UNIVERSITY
of
GLASGOW

Computing Science
Ph.D.

Dynamic Order-Sorted Term-Rewriting Systems

Brian Martin Matthews

Submitted for the degree of

Doctor of Philosophy

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

Dynamic Order-Sorted Term-Rewriting Systems

by

Brian Martin Matthews

Submitted to the Department of Computing Science
on 27th September 1996
for the degree of
Doctor of Philosophy

Abstract

This thesis considers the problems of order-sorted equational logic and its operational interpretation, order-sorted term rewriting.

Order-sorted equational logic is a well-known paradigm in the field of algebraic specification for expressing partial functions and error handling. In this thesis we review the work undertaken in this field and present a new implementation of order-sorted term rewriting, an equational theorem-proving environment known as MERILL. This implementation extends previous implementations by integrating the order-sorted rewriting process with reasoning modulo associative-commutative axioms. The implementation of this system is described and examples of its use are given.

However, the standard theory of order-sorted equational logic and term rewriting also has theoretical and pragmatic difficulties. In particular, the restriction to sort-decreasing rules limits its practical application, and this can be traced to difficulties in the semantics. There have been several approaches to overcoming these problems, which are reviewed.

The thesis develops a new approach to the semantics of order-sorted specification, via a two-tier model, called a dynamic algebra, and a corresponding dynamic equational logic. A term-rewriting theory is then associated with this semantics. This approach, dynamic order-sorted term rewriting, uses dynamic terms where the sort of terms is carried as extra information within the structure of the term. This information may change as the rewriting process uncovers further sort information of terms. Matching and unification are defined for dynamic terms, although well-sorted unification proves to be undecidable.

Dynamic rewriting is shown to be sound and complete. However, to automatically generate rewriting proofs, the Church-Rosser property is required. To establish this, completion algorithms are given, subject to two properties: well-sorted unification and the coherence property. Criteria are given which satisfy these conditions; the definition of further criteria remains an open problem. An alternative approach of constrained order-sorted rewriting, using well-sortedness constraints, is also proposed.

Examples of the use of dynamic rewriting are given and an implementation of dynamic rewriting is proposed.

Thesis Supervisor: Dr. Muffy Thomas
Title: Senior Lecturer.

Acknowledgements

I would like to thank my supervisor, Dr. Muffy Thomas for her continual encouragement and enthusiasm even at the most difficult periods in the production of this thesis. I am also especially grateful to Dr. Phil Watson who oversaw the development of the theory, providing many incisive suggestions and much constructive criticism.

This work was funded by a mature research studentship from the Science and Engineering Research Council.

I would like to thank the Computing Science Department of Glasgow University for providing a lively and stimulating environment in which to pursue this research.

I would also like to thank my colleagues at the Rutherford Appleton Laboratory for their support during my writing up period, especially Dr. Juan Bicarregui, Dr. Brian Ritchie and Dr. Stuart Robinson, who have read drafts of this thesis.

Finally I thank my parents, family and friends for their support and encouragement, and Trish, for her patience.

Contents

Chapter 1

Introduction

1.1 The Formal Methods Background

The use of mathematical modelling is becoming recognised as an important tool in the development of correct software systems. As software systems become more pervasive, it is vital that they are safe and predicatable, especially when influencing the safety of people, but also where large financial or physical losses can result. At the same time, software systems are becoming increasingly complex and this complexity must be managed. By giving a mathematical model of the system, the designer can ensure correct behaviour of the system while abstracting away from the complexity of implementation, and by developing the system within a formal method, correct software can be produced. Both parts of this process, showing that the mathematical model is correct, and that the development preserves correctness, rely on formal proof, that is deriving the result from assumptions by the application of logical inference rules.

However, there is much resistance to the adoption of formal methods, a problem which partly lies with the methods themselves. They are insufficiently mature to be easily applicable, especially in large and complex systems; proof is a difficult skill to learn and apply effectively; and tools are needed to support formal methods, especially for formal proof,

where a large volume of shallow proofs are typically needed. The precise nature of rule application within proofs is particularly suited to mechanised support.

Support for proof can lie along a spectrum, from a 'bookkeeping' role, where the human user determines the proof steps, the computer performing the accurate symbolic manipulation, and tracking dependencies; through semi-automated systems which can assist the user by searching for and suggesting directions for proof and performing simpler proofs through the use of tactics; to completely automated systems where the machine rapidly searches for and applies proof steps to generate a proof. Typically, automated systems apply a simple set of basic proof rules in an arbitrary fashion and this can result in an inefficient and fruitless search. The user of automated system has to present the problem in such a manner that the system can both find a proof if there is one, and efficiently home in on that proof.

This thesis describes an implementation of a (semi-automated) proof tool to support one formal method, *order-sorted algebraic specification.* It further describes a significant extension of the theory of order-sorted algebraic specification to more general cases, which modifies the rules and methods of proof.

1.1.1 Formal Development Methodologies

Several different broad approaches have been developed within formal methods, and three of the most common are outlined below.

Algebraic Specification. Algebraic specification languages, such as Clear [BG80], Larch [GHW85], ACT ONE [EM85], and Extended ML [ST86, KST96] describe behaviour of systems in terms of an algebra representing the data of the system and the operations as functions. This is also called *property-based* specification, as the properties of the systems are specified using first-order formulae, usually equalities, and details of representation are ignored; there is no one model which 'is' an algebraic specification, and any model which behaves in the specified way is acceptable.

Model Based Specification. Model-based specification languages, such as Z [Spi92], VDM [Jon90] or B [Abr96], can be seen as a specialisation of algebraic specification to a particular model. Basic structures such as booleans, sets, numbers, maps and sequences are supplied and are used with a notion of a 'state', which models the state of the machine, with the obvious connection to state-based programming.

Process Based Specification. Formal methods such as CCS [Mil89], CSP [Hoa85] and LOTOS [ISO88] view software as a process, or series of actions or events, rather than a model of data and operations. These events can take place in parallel, and with different actors performing them and communicating with one another, so such formalisms are particularly suited to the analysis of concurrent systems.

Within this thesis, the approach used is an algebraic specification method, with properties specified by equations, which is particularly suited to automated reasoning.

1.2 Algebraic Specification

Algebraic specification was first developed in the mid 1970's by several groups including Liskov and Zilles [LZ74], Guttag and Horning [GH78], and Goguen, Thatcher and Wright [GTW79], and is now an established technique for defining data types and their operations, and for developing software. For an introduction and references see [EMCO92, EMO92, Wir90] and for a more in depth account see [EM85].

Software is modelled as an algebra constrained by first-order formulae, usually equations, which define the properties of the system. Data objects are represented as *terms*, defined using symbols from a *signature*, which specifies *sorts* for different types of data, and *operators* for the operations on the data. These terms are interpreted in a model where the data objects are a set of atomic entities and operations are functions between entities. The behaviour is constrained by the first-order formulae which determine which algebras are satisfactory interpretations of the specification. For example, the natural numbers with the addition operator can be modelled as:

Sorts:	Nat
Operators:	$0 : \rightarrow Nat$
	$s : Nat \rightarrow Nat$
	$\_+\_ : Nat\ Nat \rightarrow Nat$
Equations:	$x + 0 = x$
	$x + s(y) = s(x + y)$

Terms such as 0, $s(0)$, $s(0) + s(0)$ represent natural numbers in any algebra, but in any valid algebra the equations must be respected, so the representation of $s(0) + s(0)$ must be that of $s(s(0) + 0)$ and also that of $s(s(0))$. Similarly, we can specify sequences of natural numbers, implicitly importing the above specification of naturals, as follows:

Sorts:	Nat, Seq	
Operators:	$\langle\rangle : \rightarrow Seq$	$head : Seq \rightarrow Nat$
	$\_\frown\_ : Nat\ Seq \rightarrow Seq$	$tail : Seq \rightarrow Seq$
Variables:	$n : Nat$, $s : Seq$	
Equations:	$head(n \frown s) = n$	$tail(n \frown s) = s$

The property-based nature of algebraic specification is clear in this example as we are not concerned with the representational details of sequences, but only that their behaviour respects the equations. When specifications are given using first-order equations, consequences can be deduced using *Equational Reasoning*. That is the application of the rules of equational logic determining the behaviour of equality.

1.2.1 The Problem of Partiality

Algebraic specification forms a powerful technique for describing systems, combining a simple presentation and syntax with deep mathematical foundations. These are straightforward when total functions are specified. However, algebraic specifications have difficulties in defining partial functions and error handling. For example, in the specification of sequences above, the terms $head(\langle\rangle)$ and $tail(\langle\rangle)$ are terms but we have no specification of their behaviour, as intuitively, head and tail are not defined upon the empty sequence. Several different approaches to these problems have been developed, including Error Algebras

and Partial Algebras.

Error Algebras, such as that of [Gog78a], and [GDLE84], allow the definition of explicit error values to represent the result of applying a function outside its domain. For example, in sequences, we could specify that $head(\langle\rangle) = underflow$, where *underflow* is an error value. However, defining such values naively can soon lead to the the specification being only modelled by trivial algebras, so error algebras keep the error values separate by splitting the specification into *ok* and *error* parts, and controlling the interaction between them, by marking operators to be safe or unsafe, variables to match *ok* or *error* terms, and equations to propagate *ok* or *error* values. Such machinery soon becomes cumbersome.

Partial Algebras, such as that of [BW82], allow operators to be modelled by partial functions in the algebra rather than total ones, the exact definition of such partial algebras having several options. Partial algebras typically require a *definedness predicate*, $D(t)$ to determine when terms are defined. Thus in the sequence example, the definedness predicates: $D(\langle\rangle)$ and $D(n \frown s)$ are added, and *head* and *tail* are declared to be partial. However, naive approaches again can lead to problems, such as trivial or non-existent initial algebras, and methods to overcome these problems require complex definitions.

1.2.2 Order-Sorted Equational Logic

One of the most successful approaches to partiality in algebraic specification is Order-Sorted Algebra[1]. Algebraic specifications are modified by splitting the set of terms into a collection of types or *sorts* and imposing a partial ordering on those sorts, which intuitively represent sets and subsets of elements in the algebra. This gives greater ease of expression, combining the strengths of error and partial algebras, without most of their difficulties. Operators are defined to be partial by specifying them to be total over subsorts; error handling can be cleanly specified by the use of error sorts, generalising the idea of *ok* and *error* sets; and the scope of equations can be restricted allowing differing properties

[1]The phrase 'order-sorted' seems to have been coined in analogy to the simpler 'many-sorted' style of specification. The phrase 'sort-ordered' is more grammatically correct and may be more aesthetically pleasing, but I shall bow to convention and use the usual terminology.

in different parts of the specification. Thus in the sequence example, *head* and *tail* are defined as total functions on the *subsort* of non-empty sequences, and thus the problem terms $head(\langle\rangle)$ and $tail(\langle\rangle)$ are not even defined.

This power and economy of order-sorted algebraic specification which makes it attractive. However, order-sorted algebra, while allowing a more flexible approach, does have subtleties which restrict its use. We will discuss this at length later in this thesis. In order to overcome the problems of order-sorted equational logic, whilst maintaining its advantages, we investigate how to modify the paradigm.

History. The awareness of the problems of specifying error values led Goguen to develop the order-sorted specification [Gog78b]. This approach, subsequently known as the *Overloaded* approach, is followed in [GM87, KKM88, MGS89, GM89, GKK90]. An alternative approach, known as *Non-overloaded*, was developed by Walther [Wal83] in the context of resolution, and Gogolla [Gog84] and followed by [Gog87, Smo86, SS86, SNGM88, Kir88, Wal89, SS89] and also [Poi90], who discusses both approaches, but tends to favour the second. Waldemann gives a survey [Wal92] giving a comparison of the two methodologies pointing out their strengths and weaknesses and Diaconescu [Dia91] gives a briefer overview of the field. Cunningham and Dick [CD85] were the first to develop order-sorted rewriting, with an independent approach subsumed by [GM89]. The problems of these methods are discussed together with possible solutions in [GM89, GKK90, WD89, IG88, GI88, Gan91b, CH91b, Wit92, Wer93] and [HKK94], the most recent work in the area, which most closely matches the work of this thesis, and are discussed at length in later sections.

1.3 Proof for Algebraic Specifications

In order to give proofs of properties of algebraic specifications, we reason using the given equations. This equational reasoning is based on the ancient mathematical principle, often know as Leibnitz's law, that if one thing is equal to another, then anything that is true of one is true of the other, that is if $P(a)$ and $a = b$ then $P(b)$. The operational interpretation of this is that subterms of a term can be replaced by equal subterms, which are instances

of some equation and the semantics of the term remains the same. Thus the required property is described as an equation, and its terms are formally manipulated by replacing equals for equals, until one term is transformed into the other. This is a simple and easily understood process. However, such proofs are typically formed by many small steps which are tedious for a human user to apply and there is a high probability of bookkeeping error. Also, the search space is large, and strategies are needed to search it efficiently. Thus it is appropriate that these proofs should be automated as much as possible.

1.3.1 Term Rewriting

Term rewriting is arguably the simplest, most appropriate and most successful technique developed to automate equational proofs. The large search space of equational replacement is restricted by *orienting* equations into *rewrite rules*, and only allowing the replacement to take place in one direction, by *matching* the left-hand side of rules to subterms and rewriting to the right-hand side. Thus in the sequence example, we can consider the equations as left-to-right rewrite rules and using $\rightarrow$ to denote a rewriting step, give the rewrites:

$$head(tail(0 \frown succ(0) \frown succ(0) \frown \langle\rangle)) \rightarrow head(succ(0) \frown succ(0) \frown \langle\rangle) \rightarrow succ(0)$$

Rewriting has the advantage over equational replacement in that it is directed, limiting the search space and proof of equality can be performed by detecting whether terms can be rewritten to the same term, a process which can easily be automated. However, care has to be taken that rewriting terminates, otherwise infinite rewriting sequences can be followed and proofs not established.

However, rewriting is not strong enough to cover all equational theories. The commutative axiom, considered as rewrite rule, does not terminate. It results in the sequence:

$$a + b \rightarrow b + a \rightarrow a + b \rightarrow b + a \rightarrow \cdots$$

leading to looping in an automated system. Commutativity frequently occurs with the Associativity axiom and together these are known as the AC axioms. These axioms commonly occur in practical problems, and it is desirable to handle them in conjunction in some other way to avoid non-termination. This can be done by rewriting with respect to

(or 'modulo') the equational theory, where equational variants are matched and rewriting is to terms which are equal modulo the equational theory. Thus the AC equations are not used as rewrite rules directly, but built into the rewriting method itself.

Rewriting in order-sorted specifications is a more straightforward extension to standard rewriting theory than rewriting modulo equations. Care has to be taken that the match formed preserves the sorts. However, as we shall see, there are restrictions to standard order-sorted rewriting theory which limit its applicability.

History. The replacement of equals for equals has always been used by mathematicians. However, the investigation of its use as an automated proof method using computers dates to the development of the Knuth-Bendix algorithm in the late 1960's [KB70]. Interest grew slowly in the 1970's with the work of Lankford [LB77] on rewriting modulo permutative theories, including commutativity, and Huet [Hue80] extended the theory to rewriting modulo left-linear theories. Huet and Oppen's survey [HO80] was followed by an explosion of interest in the 1980's and several implementations appeared incorporating these new developments, including REVE [For84], RRL [KZ89], the Larch Prover [GG89], the OBJ family [GW88] and ERIL [Dic85]. Peterson and Stickel [PS81], and Jouannaud and Kirchner [JK86] provided full accounts of rewriting modulo arbitrary equational theories. Cunningham and Dick [CD85] were the first to develop order-sorted rewriting, with an independent approach improved upon in [GJM85, SNGM88, GM89]. Order-sorted rewriting modulo equational theories is considered in [GKK90]. Research is still continuing in many of the areas of rewriting theory including termination and unification methods, rewriting and modularity, and extensions of rewriting into constraint and logic programming. For surveys see [JD90, Klo90, Pla93, Jou93].

1.3.2 Unification

Unification is the process of finding a solution to a system of equational constraints, and thus is a fundamental equational operation. Syntactic unification over homogeneous signatures is a straightforward process of finding common instances of expressions for which very efficient algorithms are known to find the 'best' or 'most general' unifier. By adding

an equational theory to the signature, the unification problem becomes equation solving in that theory, which is undecidable in general and if soluble, complex. Whilst merging an equational theory into the unification algorithm produces a more complex unification algorithm, with the possibility arising of either an infinite set of solutions or no most general unifiers, it is useful in some cases, such as the commutative theory, which is non-terminating as a rewrite rule. As the AC axioms commonly occur in practical problems, the *associative-commutative (AC)* theory has been the most studied equational theory in the context of unification. If an order-sorted signature is used, the unification algorithm becomes more complex again as checks must be made on the sorts of terms to prevent the unification of expressions of incomparable sort, which can lead to multiple most general unifiers.

History. Although described by Herbrand [Her30, Her67], the first effective algorithm for syntactic unification was given by Robinson [Rob65] as one of the inference rules of the resolution theorem proving method. Knuth and Bendix [KB70] used the same technique for completing sets of equations. Robinson's algorithm was exponential, and Paterson and Wegman [PW78] developed a linear algorithm, although it could have an expensive overhead. Martelli and Montenari [MM82] presented a new perspective on unification as solving systems of equations. In addition they developed an algorithm which maximises the use of commonalities of terms and solves equations simultaneously. Whilst not linear, it proves a more effective algorithm in most cases.

Unification over equational theories was first considered by Plotkin [Plo72], and the first effective algorithm for AC unification was devised by Stickel [Sti81], shown to be correct by Fages [Fag87]. Since then there has been much work on the development of algorithms for equational unification and for AC unification in particular, including [Fag87, For87, Kir87, Kir89, LC89, AK90, BCD90].

Unification over order-sorted signatures was first investigated by Dick [CD85]. The combination of unification in equational theories and order-sorted signatures was studied by Schmidt-Schauss [SS86, SS85, SS89]. Smolka, Goguen and Meseguer give a category theoretic account of order-sorted equational unification in [MGS89], where they present a Martelli and Montenari style algorithm, and [SNGM88] present a two stage algorithm. Waldemann also presents an algorithm for order-sorted unification [Wal89] and gives con-

ditions for unification to be finitary and unitary. Claude Kirchner [Kir88] presents Order-Sorted Equational Unification as a series of inference rules and also extends the method to signatures which have differing equational theories on different parts of the signature by giving a general mutation rule, subsequently improved by Boudet [Bou92].

The survey papers by Siekmann [Sie89], Jouannaud and Kirchner [JK91], and Baader and Siekmann [BS93] give an overview of unification theory and algorithms.

1.3.3 Completion

Term rewriting is a powerful method for automated reasoning since it restricts and directs the space through which the theorem prover can search for proofs of equalities under equational theories, compared with arbitrary equational reasoning. However, the price which is paid for this is loss of completeness: there may be proof of equalities which exist in the equational theory which cannot be found by rewriting. For rewriting systems to be complete, the Church-Rosser property is required of the rewrite rules, which establishes that for all equational proofs of equalities there is a corresponding rewriting proof. Completion seeks to establish the Church-Rosser property for terminating rewrite systems by considering when overlapping terms can be rewritten in two ways, known as Critical Pairs. This process requires unification of existing rules to generate the critical pairs, which may then be added as extra rules to the set of rules. This procedure may generate sufficient rules to re-establish completeness. However, it may alternatively generate an infinite set of new rules, or fail through the lack of termination proof for a new rule.

Completion can also be extended to rewriting modulo equational theories, with the Church-Rosser property also modified to modulo equations. In this case, unification modulo the equational theory is also required. Further, it can be extended to order-sorted theories and the combination of order-sorted signatures and rewriting modulo equational theories.

History. Completion was first introduced in a seminal paper by Knuth and Bendix [KB70]. Huet [Hue81] gives an alternative proof of the algorithm, and Peterson and Stickel and Jouannaud and Kirchner give accounts of completion modulo equations. Smolka et. al. [SNGM88]

describe completion in order-sorted theories and Gnaedig, Kirchner and Kirchner [GKK90] describe completion over order-sorted rewriting modulo equational theories. Bachmair [Bac87, Bac91] gives an alternative approach to the description of completion using inference rules and to proofs of correctness using the notion of equational proofs, which has since become the standard method of verifying completion algorithms.

1.4 Using Order-Sorted Logic

Equational reasoning can be used in several areas of computer science. In this section we give a brief account of some of these applications, with special emphasis on the use of order-sorted equational reasoning.

1.4.1 Theorem-Proving Systems

There are many existing term-rewriting based theorem-proving systems, for example RRL [KZ89], LP [GG89], ORME [Les89, Les90] and ReDuX [Bün93, WB95], which perform reasoning modulo associative-commutative operators, but over many-sorted signatures. The ERIL system [Dic87, DK88] can perform equational reasoning over order-sorted signatures, but cannot handle the associative-commutative axioms in the reasoning mechanism. ELIOS-OBJ [Gna92a] is an extension of the OBJ3 [GW88] programming environment to a prover which can perform completion, although not completion modulo equations. In this thesis we describe a term-rewriting theorem-proving system, MERILL, which incorporates both an order-sorted signature and also equational theories, in particular the AC-theory.

1.4.2 High-level Programming Languages

OBJ3 [GW88, KKM88] does handle AC-matching and rewriting over an order-sorted signature, but this system is designed to be a high-level programming language which uses term rewriting as its computation mechanism. It does not attempt completion and there-

fore does not require unification. The system has been extended to explore rewriting as a programming language, including modules and higher-order types [Gog88], and object-oriented programming [GM88]. 2OBJ [GSHH92] has been developed as a general purpose first-order logic theorem prover on top of OBJ3. A similar language is TEL [Smo88], which combines an order-sorted type system with polymorphism in a combination of logic, using Horn clauses and resolution, and functional programming, using conditional equations and rewriting.

1.4.3 Functional Programming

Functional programming (see for example [BW88]) is a declarative programming paradigm based on the lambda calculus (see [HS86]) which has aroused interest for its conceptual clarity, conciseness, and the ease of proof compared with conventional programming languages. Programs are described using systems of equations, and equational reasoning is frequently used in an ad hoc fashion to show program correctness. Rewriting has been considered for proving properties of functional programs, for example by Martin and Nipkow [MN90a]. However, the higher-order nature of functional programs makes them awkward to analyse in this fashion.

Equational reasoning has another role to play in functional programming. Functional languages often provide a powerful yet flexible type system and the type inference procedure, for example as in [Mil78], uses unification. Blott and Wadler [WB89, Blo91] have developed an extended type system for functional languages which handles overloading of operators which has been incorporated into the language Haskell [HPJE92]. This has a three layered type system, with values, types and type classes. The type classes divide the types into a hierarchical structure, similar to the sorts in an order-sorted signature, and the types within the classes have ranks similar to operators in an order-sorted signature. Nipkow and Snelting [NS91] have exploited this similarity by using order-sorted unification to perform type checking for a Haskell-style language. The current author has explored this idea further [Mat92] using order-sorted equational matching and unification to search libraries of Haskell functions via their type signature, extending the work of Rittri [Rit89, Rit93]

for the ordinary type system of functional languages.

1.5 Overview of the Thesis

The motivation behind this thesis is the development of a new automated theorem-proving system based upon order-sorted term-rewriting. However, in the development of this system, it soon became apparent that the shortcomings of the existing order-sorted algebraic specification and rewriting theory meant that the application domain of this theorem prover was strictly limited, and many useful specifications were not within the range of its application. As a consequence, it was decided to reconsider the theory of order-sorted algebra, logic, and rewriting to extend its range of application, with a view to implementing this new theory within a future version of the theorem prover.

We give a short synopsis of the content of each chapter.

In Chapter 2, the standard theory of order-sorted specification, algebra, and equational logic is discussed, contrasting the given *non-overloaded* approach with the *overloaded* approach. Unification, rewriting and completion modulo equational theories, with emphasis on the AC theory, are then introduced, and the implementation of this rewriting theory within a theorem-proving system, MERILL, is then described.

In Chapter 3, the problems associated with order-sorted specification are described, together with a discussion of why the existing theory is too restrictive. Although these problems are ultimately intractable, several of the approaches to mitigating these problems are discussed. These approaches become increasingly sophisticated: however, it is felt that none of these methods gives an entirely satisfactory account of handling the shortcomings of the existing theory.

In Chapter 4, a new two-tier algebraic semantics for order-sorted specification is defined, known as dynamic order-sorted algebra. Dynamic order-sorted equational logic is then introduced, and proven to be sound and complete with respect to the dynamic algebra. The notion of well-sorted terms in this algebra is then discussed, the sorts of terms being

dependent on the equational logic. The sorting of terms is shown to be undecidable.

In Chapter 5, dynamic order-sorted terms are introduced. These are used as an operational representation of terms which record their sorts within their structure. Dynamic substitutions are defined and some basic operations on dynamic terms are given. Finally, dynamic equations are introduced.

In Chapter 6, the options available to define dynamic order-sorted matching and dynamic order-sorted unification are explored. Decidable algorithms for constructing canonical matching and unifying substitutions are given. However, in the case of unification, such substitutions may not be well-sorted, and a discussion follows on how well-sorted unifiers can be identified.

In Chapter 7, a description of dynamic order-sorted term rewriting is given, based on three separate rewriting methods. The soundness and completeness of dynamic rewriting with respect to dynamic equational logic is proven, and termination properties of dynamic rewriting systems is discussed. Church-Rosser and confluence properties are defined, as well as similar properties modulo sorts, which use the identity of the underlying terms. Results on local confluence are given, with a view to automatically generating dynamic rewriting proofs.

In Chapter 8, the work towards automated rewriting proofs is presented. Dynamic critical pairs are defined and the critical pair lemma is proven for rewriting modulo sorts. The completion of dynamic rewriting systems to give decision procedures for dynamic order-sorted rewriting is explored, identifying two conditions for the realisation of this procedure. Some sufficient criteria for these conditions are given. An alternative approach to dynamic order-sorted rewriting is discussed, which uses sort constraints, generated during dynamic unification, as conditions on equations.

In Chapter 9, the thesis is concluded with a summary and a discussion on the future implementation and research directions. Some examples of dynamic rewriting are given which illustrate how dynamic rewriting overcomes the problems highlighted in Chapter 3, and also how dynamic rewriting could be taken further.

1.5.1 Original Content of Thesis

This thesis contains the following original content.

- The first implementation of order-sorted associative-commutative term-rewriting theory in a practical equational theorem proving system is described.
- A new semantic framework for order-sorted algebra, dynamic algebra is derived, which captures the intuition of order-sorted specification, with a simple dynamic equational logic and a notion of sorting which is semantic rather than syntactic.
- Dynamic terms, which record sort information, are defined as an operational representation of terms and their sorts.
- Decidable matching and unification algorithms for dynamic terms are derived and proven correct.
- A sound and complete rewriting theory for order-sorted algebra based on dynamic terms is developed. This rewriting theory manipulates sorts as well as operators, and does not require the *compatibility* condition for its completeness.
- Confluence, critical pair and completion results for dynamic rewriting are proved and sufficient conditions are identified for completion. Criteria are given which extend the range of application of the completion algorithm beyond the standard theory.
- A constrained rewriting method for extending dynamic rewriting is presented as an alternative approach.

Chapter 2

Order-Sorted Specification

This chapter summarises the existing work on order-sorted algebra, logic and rewriting theory. Notation and concepts used throughout this thesis are introduced in a presentation of standard theories, with discussion of the variations in the literature. This work also stands as the theoretical basis behind the implementation of associative-commutative order-sorted rewriting in the MERILL system, which is described in this chapter.

2.1 Sets and Relations

We use standard notation for first order predicate logic and set theory, with operators $\neg, \wedge, \vee, \Rightarrow, \forall, \exists$ and $\emptyset, \in, \subseteq, \subset, \supseteq, \supset, \cup, \cap$ defined in the standard fashion; additionally $\uplus$ is used for disjoint union. The notation $\mathbb{N}$ represents the set of natural numbers. A relation R on sets A, B is a set of pairs such that $R \subseteq A \times B$.

Definition 2.1. A relation $\preceq$ is a *quasi-ordering* if $\forall x \cdot x \preceq x$ (*reflexive*) and $\forall x, y, z \cdot x \preceq y \wedge y \preceq z \Rightarrow x \preceq z$ (*transitive*). It is a *partial-ordering* if it is reflexive, transitive and $\forall x, y \cdot x \preceq y \wedge y \preceq x \Rightarrow x = y$ (*antisymmetric*).

Notation 2.2. Let $\rightarrow$ be a relation. Then we use the following notation:

- $\xrightarrow{n}$ the closure up to n steps of $\rightarrow$
- $\xrightarrow{+}$ the transitive closure of $\rightarrow$
- $\xrightarrow{*}$ the transitive reflexive closure of $\rightarrow$
- $\leftarrow$ the symmetric relation of $\rightarrow$
- $\longleftrightarrow$ the symmetric closure of $\rightarrow$
- $\xleftrightarrow{+}$ the transitive symmetric closure of $\rightarrow$
- $\xleftrightarrow{*}$ the reflexive transitive symmetric closure of $\rightarrow$

A *multiset* (or *bag*) is an unordered collection of elements from some base set E, with possibly repeated occurrences, and can be formally described as a mapping from elements to the number of occurrences of the elements. The following definition follows [JL82]; equivalent definitions are given in, for example, [Pla93, HO80].

Definition 2.3. Given a set E, a multiset M over E is a mapping $E \rightarrow \mathbb{N}$. If $x \in E$ we say that $x \in M$ if and only if $M(x) > 0$. $\mathcal{M}(E)$ is the set of all finite multisets over E, that is $\forall M \in \mathcal{M}(E)$ the set $\{x | M(x) > 0\}$ is finite.

The empty multiset $\{\} \in \mathcal{M}(E)$ is the multiset such that $\forall x \in E \cdot \{\}(x) = 0$. The *sum* of two multisets, $M + N$ is $(M + N)(x) = M(x) + N(x)$. M is *included* in N, $M \subseteq N$ if $\forall x \in E \cdot M(x) \leq N(x)$. If $M \subseteq N$ then the *difference* $N - M$ is $(M - N)(x) = M(x) - N(x)$.

An ordering $<$ on E is extended to an ordering on $\mathcal{M}(E)$.

Definition 2.4. The **Dershowitz-Manna Ordering** [DM79] is defined as $M \prec\!\!\prec N$ if and only if $\exists X, Y \in \mathcal{M}(E)$ such that:

i) $\{\} \neq X \subseteq N$

ii) $M = (N - X) + Y$

iii) $\forall y \in Y \cdot \exists x \in X \cdot y < x$

So M is less than N if a finite number of elements of N can be replaced by a finite number of elements, all of which are less than an element removed from N, to result in M.

Example 2.5. Multisets over natural numbers $\mathcal{M}(I\!N)$ include:

$$\{1\} \lessdot\!\!\lessdot \{1,1\} \lessdot\!\!\lessdot \{2\} \lessdot\!\!\lessdot \{0,0,0,0,3,0,0\}$$

□

This ordering on multisets is important for two reasons. Firstly it is the maximal ordering on multisets which extends the ordering on the underlying set in a monotonic fashion.

Definition 2.6. Let $\tau : E \times E \rightarrow \mathcal{M}(E) \times \mathcal{M}(E)$. $\tau(<)$ is said to be a monotonic extension of $<$ if and only if (i). $\tau(<)$ is an ordering and (ii). τ is monotonic, that is $\prec \subseteq < \Rightarrow \tau(\prec) \subseteq \tau(<)$.

Theorem 2.7. ([JL82])

i) $\lessdot\!\!\lessdot$ is a monotonic extension of $<$.

ii) If $\tau(<)$ is an monotonic extension of $<$, and $\lessdot\!\!\lessdot \subseteq \tau(<)$ then $\lessdot\!\!\lessdot = \tau(<)$.

Secondly, $\lessdot\!\!\lessdot$ is particularly useful for termination proofs. The multiset extension preserves well-foundedness of orderings.

Theorem 2.8. [DM79] If $<$ is well-founded on E, then $\lessdot\!\!\lessdot$ is well-founded on $\mathcal{M}(E)$.

2.2 Non-Overloaded Order-Sorted Signatures

The syntax of order-sorted specifications is given by order-sorted signatures.

Signatures.

Order-sorted signatures consist of three components: the sorts, an ordering on the sorts and the operators. A *sort symbol* is a name with no structure. A *subsort declaration* is a pair of sorts of the form $s_1 \leq s_2$. An *operator* is also a name but it has an associated *rank*.

Definition 2.9. The *rank* of an operator is a pair consisting of a sequence of sorts (the *arity*) and a sort (the *coarity*). For convenience we write operator f with rank (w, s) as an

operator declaration of the form $f : s_1, \ldots, s_n \rightarrow s$ where $w = \langle s_1, \ldots, s_n \rangle$. We write $|f|$ for the length of the arity of an operator.

Intuitively, the sorts describe collections of objects and the operators functions over those collections.

Definition 2.10. An *Order-Sorted Signature*, Σ, is a triple, $(S_\Sigma, \leq_\Sigma, \mathcal{F}_\Sigma)$ where:

1. S_Σ is a set of sorts.
2. $\leq_\Sigma$ is a partial ordering on S_Σ, which is the least transitive-reflexive closure of a set of subsort declarations.
3. $\mathcal{F}_\Sigma$ is a $(w, s) \in S_\Sigma^* \times S_\Sigma$ indexed set of operators.

More fully, $\mathcal{F}_\Sigma = \{\mathcal{F}_{\Sigma(w,s)}\}_{(w,s) \in S_\Sigma^* \times S_\Sigma}$, where each $\mathcal{F}_{\Sigma(w,s)}$ is the set of operators with rank (w, s). Note that an operator can have more than one rank.

When the signature is clear in context, the Σ subscript is dropped. Also, we write $s \in \Sigma$ when $s \in S_\Sigma$ and $f \in \Sigma$ when $f \in \mathcal{F}_{\Sigma(w,s)}$, for some (w, s). The ordering on sorts is extended to sequences of sorts and to pairs in $S_\Sigma^* \times S_\Sigma$ in the obvious, pointwise and pairwise fashions.

Two sorts s and s' are *incomparable* written $s \bowtie s'$ if $s \not\leq_\Sigma s'$ and $s' \not\leq_\Sigma s$. The *meet* of two sorts s and s', written $s \wedge s'$, is defined to be the maximal elements of the set of sorts less than or equal to both: if $s'' \in (s \wedge s')$ then $s'' \leq_\Sigma s$, $s'' \leq_\Sigma s'$ and if for any $r \in S_\Sigma$ if $r \leq_\Sigma s$ and $r \leq_\Sigma s'$ then $\exists s'' \in (s \wedge s')$ for which $r \leq_\Sigma s''$.

Terms.

For each sort $s \in S_\Sigma$, assume an infinite set of variables $\mathcal{X}_s$ of sort s. The set of all variables is $\mathcal{X} = \bigcup_{s \in \Sigma} \mathcal{X}_s$.

Definition 2.11. The set of Σ-*Terms* of a sort $s \in \Sigma$ is the set constructed as follows:

1. If $x \in \mathcal{X}_{s'}$, $s' \leq_\Sigma s$, x is a (variable) Σ-Term of sort s.

2. If $f \in \mathcal{F}_{\Sigma(\langle s_1,\ldots,s_n\rangle,s')}$, $s' \leq_\Sigma s$ and $t_1,\ldots,t_n$ are Σ-Terms of sorts $s_1,\ldots,s_n$ respectively, then $f(t_1,\ldots,t_n)$ is a (compound) Σ-Term of sort s. If $n = 0$ then f is a *constant* of sort s.

We write that a term t is of sort s by $t : s$.

The set of all Σ-Terms on a set of variables $\mathcal{X}$ is denoted $T_\Sigma(\mathcal{X})$. If the set of variables $\mathcal{X} = \emptyset$ then the set $T_\Sigma(\emptyset)$ (or just T_Σ) is the set of *Ground* terms. The set of variables occurring in a term t is $Vars(t)$, and this is extended in the obvious way to sets of terms, equations, and sets of equations.

The notion of paths is introduced to access subterms within terms.

Definition 2.12. An *Path* is a possibly empty (written ϵ) finite sequence of integers, written $n_1.n_2.\cdots.n_k$. The set of paths for a term t, $O(t)$ is defined to be:

$O(t) = \{\epsilon\}$ if t is a variable or a constant
$O(f(t_1,\ldots,t_n)) = \{\epsilon\} \cup \{i.p | 1 \leq i \leq n, p \in O(t_i)\}$

Given a term t and a path $p \in O(t)$ the subterm occurring at p, $t|_p$ is $t|_\epsilon = t$ if $p = \epsilon$, and $f(t_1,\ldots,t_n)|_{i.u} = t_i|_u$ if $p = i.u$. The result of replacing a subterm of t at path $p \in O(t)$ by term s is written $t[p \leftarrow s]$.

From the definition of terms, we can also give the definition of a least sort of a term.

Definition 2.13. Given a signature Σ, the set of *least sorts* of a term t, $\mathcal{LS}(t)$ is the minimal set of sorts such that $\forall s \in \mathcal{LS}(t) \cdot t : s$, and $\forall s'$ such that $t : s'$ then $\exists s \in \mathcal{LS}(t)$ such that $s \leq s'$.

If the least sorts of terms are unique, then unification becomes finitary [SS89] and in the overloaded semantics initial algebras exist (see below). Uniqueness is ensured by the property of Regularity. There are several definitions of regularity in the literature depending on the semantics chosen; for a discussion see [Wal92]. The following is from [SNGM88].

Definition 2.14. A signature Σ is said to be *Regular* if for each $t \in T_\Sigma(\mathcal{X})$ the set $\{s|t : s\}$ has a unique minimal element, unambiguously denoted as $\mathcal{LS}(t)$.

Regularity can be decided by analysis of the signature.

Lemma 2.15. ([SNGM88],[Wal89]) A signature Σ is regular if and only if for every $f \in \mathcal{F}_\Sigma$ and every $w \in S_\Sigma^*$, the set $\{s|f : w' \rightarrow s \in \mathcal{F}_\Sigma \wedge w \leq_\Sigma w'\}$ is either empty or has a unique minimal element.

Example 2.16. Assuming a specification of naturals Nat, we give the following signature for sequences of numbers.

Sorts:	$Nat,\ NeSeq,\ Seq$	
Subsorts:	$Nat \leq NeSeq \leq Seq$	
Operators:	$\langle\rangle : \rightarrow Seq$	
	$\_@\_ : NeSeq\ Seq \rightarrow NeSeq$	$\_@\_ : Seq\ Seq \rightarrow Seq$
	$head : NeSeq \rightarrow Nat$	$tail : NeSeq \rightarrow Seq$

Here, the Naturals are regarded as sequences of length one; there are two ranks for the concatenation operator, giving different result sorts for differing arguments; and head and tail are defined only on *NeSeq*. Thus terms and their sorts include $0{:}NeSeq$, $0@\langle\rangle{:}NeSeq$. The empty sequence $\langle\rangle$ is not of sort *NeSeq* and consequently, $head(\langle\rangle)$ and $tail(\langle\rangle)$ are ill-defined terms. Note also that this signature is regular. □

Substitutions.

Definition 2.17. A Σ-*Substitution* is an endomorphism on $T_\Sigma(\mathcal{X})$ which preserves the sorts of terms and is identity on all but a finite set of variables. A substitution is thus completely defined by its effect on that set of variables, the *Domain* of a substitution σ, written $Dom(\sigma)$. Substitutions are *sort preserving*: $\forall(x : s) \in Dom(\sigma) \cdot \sigma x : s$. The *Image* of a substitution is the set of variables occurring in the range: $Im(\sigma) = \bigcup_{x \in Dom(\sigma)} Vars(\sigma x)$.

Substitutions can be constrained to apply to a set of variables. A substitution σ restricted to a set of variables $\mathcal{W} \subseteq \mathcal{X}$, is denoted $\sigma_{|\mathcal{W}}$ and defined as $\sigma_{|\mathcal{W}}(x) = \sigma(x)$ if $x \in \mathcal{W}$ and

$\sigma_{|\mathcal{W}}(x) = x$ otherwise.

A substitution σ can be written as a set of pairs $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, for all $x_i \in Dom(\sigma)$. The composition of substitutions σ and λ, written $\lambda \circ \sigma$ is the substitution given by $\forall t \cdot (\lambda \circ \sigma)t = \lambda(\sigma t)$.

Definition 2.18. ρ is a *renaming* substitution if $\forall x \in Dom(\rho) \cdot \rho x \in X$ and $s(x) = s(\rho x)$.

Definition 2.19. Terms t_1, t_2 are *alpha-equivalent*, $t_1 =_\alpha t_2$ if for some renaming ρ $\rho t_1 = t_2$

We use substitutions in the definition of subsumption and matching.

Definition 2.20. A term $t \in T_\Sigma(\mathcal{X})$ *matches* a term t' if there is a Σ-substitution σ such that $\sigma t = t'$. In this case t *subsumes* t', (t' *is an instance of* t), and we define the *subsumption ordering* $t \preceq t'$. We extend this to substitutions. For substitutions σ and τ, if $\exists \lambda$ such that $\forall x \in \mathcal{X}.\lambda(\sigma(x)) = \tau(x)$, then $\sigma \preceq \tau$. This can be restricted to a set of variables $\mathcal{W} \subseteq \mathcal{X}$; $\sigma \preceq_E^{\mathcal{W}} \tau$ if and only if there $\exists \lambda$ such that $\forall x \in \mathcal{W} \cdot \lambda(\sigma(x)) =_E \tau(x)$.

One interesting observation on the subsumption ordering, in the light of the later presentation of *dynamic matching*, is the following lemma.

Lemma 2.21. If Σ is regular, and $t \preceq t'$, then $\forall p \in O(t)$, $\mathcal{LS}(t|_p) \geq_\Sigma \mathcal{LS}(t'|_p)$.

Proof (2.21). Assume that $t \preceq t'$, so there is a σ such that $\sigma t = t'$. Proof is by induction on the height of subterms. If $t|_p = x : s \in \mathcal{X}$, then $\sigma x = t'|_p$ and $\mathcal{LS}(t'|_p) \leq s$. If $t|_p = c$, a constant, then $t|_p = t'|_p$ and the proposition holds. Consider a subterm, at path p of length $n+1$. Let $t|_p = f(t_1, \ldots, t_n)$, and $t'|_p = f(t'_1, \ldots, t'_n)$. By induction hypothesis, $\mathcal{LS}(t_i) \geq \mathcal{LS}(t'_i)$, $f(t'_1, \ldots, t'_n)$ is a term of sort $\mathcal{LS}(t|_p)$ and thus $\mathcal{LS}(t|_p) \geq \mathcal{LS}(t'|_p)$. □

2.3 Non-Overloaded Order-Sorted Algebras

The denotation of terms over an order-sorted signature is given by *Order-Sorted Algebras.* This section follows [SNGM88], and the algebra it describes is *Non-Overloaded*[1].

Definition 2.22. Given signature Σ, an *(Non-overloaded) Order-Sorted Algebra*, $\mathcal{A}_\Sigma$, (Σ-*Algebra*) is a pair $(S_\Sigma^{\mathcal{A}}, \mathcal{F}_\Sigma{}^{\mathcal{A}})$ such that:

1. $S_\Sigma^{\mathcal{A}}$ is a set of sets indexed by S_Σ; for each $s \in S_\Sigma$ there is a set $s^{\mathcal{A}} \in S_\Sigma^{\mathcal{A}}$, the *carrier* of s. The carrier of $\mathcal{A}$ is the union of the carriers: $C^{\mathcal{A}} = \bigcup_{s \in S} s^{\mathcal{A}}$.

2. If $s \leq_\Sigma s'$ then $s^{\mathcal{A}} \subseteq s'^{\mathcal{A}}$.

3. For each operator $f \in \mathcal{F}_\Sigma$ there is a function $f^{\mathcal{A}} : D_f^{\mathcal{A}} \rightarrow C^{\mathcal{A}} \in \mathcal{F}_\Sigma{}^{\mathcal{A}}$, where $D_f^{\mathcal{A}} \subseteq (C^{\mathcal{A}})^{|f|}$ is the domain of $f^{\mathcal{A}}$. Thus each operator has a unique denotation.

4. If $f \in \mathcal{F}_{\Sigma(<s_1 \ldots s_n>,s)}$ and $a_i \in s_i^{\mathcal{A}}$ for $i = 1 \ldots i$ then $(a_1, \ldots, a_n) \in D_f^{\mathcal{A}}$ and $f^{\mathcal{A}}(a_1, \ldots, a_n) \in s^{\mathcal{A}}$. If $c \in \mathcal{F}_{\Sigma(\Lambda,s)}$ then $c^{\mathcal{A}} \in s^{\mathcal{A}}$.

A signature is denoted by a class of algebras, which are related using Homomorphisms.

Definition 2.23. Given signature Σ and Σ-algebras $\mathcal{A}$ and $\mathcal{B}$ a Σ-*homomorphism* $h : \mathcal{A} \rightarrow \mathcal{B}$ is a mapping between the algebras which satisfies the following conditions.

1. $h(s^{\mathcal{A}}) \subseteq s^{\mathcal{B}}$ for every sort $s \in S_\Sigma$.
2. $h(D_f^{\mathcal{A}}) \subseteq D_f^{\mathcal{B}}$ for every operator $f \in \mathcal{F}_\Sigma$.
3. $h(f^{\mathcal{A}}(a_1, \ldots a_n)) = f^{\mathcal{B}}(h(a_1), \ldots, h(a_n))$ for all $f \in \mathcal{F}_\Sigma$, and $(a_1, \ldots a_n) \in D_f^{\mathcal{A}}$.

Σ-algebras and Σ-homomorphisms together form a category, $\mathbf{OSAlg}_\Sigma$. A homomorphism $h : \mathcal{A} \rightarrow \mathcal{B}$ is an *isomorphism* if there is a homomorphism $h' : \mathcal{B} \rightarrow \mathcal{A}$ such that $h' \circ h = id_{\mathcal{A}}$ and $h \circ h' = id_{\mathcal{B}}$, where $id_{\mathcal{A}}$ and $id_{\mathcal{B}}$ are the identity homomorphisms for $\mathcal{A}$ and $\mathcal{B}$.

[1] This is the terminology used in [Wal92]. Mosses objects to this designation as misleading, as syntactic overloading is permitted in this paradigm, and prefers to call this style *Universal Order-Sorted Algebra* [Mos92]. However, we use the term Non-Overloaded to contrast it with the Goguen style semantics.

2.3.1 Term and Initial Algebra

Definition 2.24. Given signature Σ, and Σ-sorted set of variables $\mathcal{X}$, the *Free Term-Algebra* $\mathcal{T}_{\Sigma,\mathcal{X}}$ on $\mathcal{X}$ is given by the following construction on $T_\Sigma(\mathcal{X})$.

1. $s^{\mathcal{T}_{\Sigma,\mathcal{X}}} = \{t | t \in T_\Sigma(\mathcal{X}) \text{ and } t : s\}$
2. $D_f^{\mathcal{T}_{\Sigma,\mathcal{X}}} = \{(t_1,\ldots,t_n) | f(t_1,\ldots,t_n) \in T_\Sigma(\mathcal{X})\}$
3. $f^{\mathcal{T}_{\Sigma,\mathcal{X}}}(t_1,\ldots,t_n) = f(t_1,\ldots,t_n)$

If the set of variables $\mathcal{X} = \emptyset$, then the term algebra $\mathcal{T}_{\Sigma,\emptyset}$ is formed by ground terms.

Thus terms form an algebra for the signature, although care is taken to distinguish between the well-sorted terms $T_\Sigma(\mathcal{X})$ which are syntactic entities, and the free term algebra $\mathcal{T}_{\Sigma,\mathcal{X}}$, which is a denotation of the signature. This algebra is canonical in the following sense.

Definition 2.25. If $\mathcal{A}$ is a Σ-algebra, and $\mathcal{V} \subseteq \mathcal{X}$ a set of variables, a $(\mathcal{V},\mathcal{A})$-*assignment* is a mapping $\nu : \mathcal{V} \rightarrow \mathcal{A}$ such that $\forall x \in \mathcal{V}.\nu(x : s) \in s^{\mathcal{A}}$. The *denotation* of a term t in $\mathcal{A}$, with assignment ν on $Vars(t)$, written t^ν, is a mapping $\mathcal{T}_{\Sigma,\mathcal{V}} \rightarrow \mathcal{A}$ given by:

$$\begin{array}{rcll} x^\nu & = & \nu(x) & \text{if } x \in \mathcal{X}, \\ f(t_1,\ldots,t_n)^\nu & = & f^{\mathcal{A}}(t_1^\nu,\ldots,t_n^\nu) & \text{otherwise.} \end{array}$$

For any $(\mathcal{V},\mathcal{A})$-assignment ν, the denotation function $\_^\nu : \mathcal{T}_{\Sigma,\mathcal{V}} \rightarrow \mathcal{A}$ is a homomorphism. In particular, if $\mathcal{V} = \emptyset$ there is only the trivial $(\emptyset,\mathcal{A})$-assignment. There is a unique homomorphism from $\mathcal{T}_{\Sigma\emptyset}$ to $\mathcal{A}$ and thus $\mathcal{T}_{\Sigma,\emptyset}$ is the *Initial Object* in $\mathbf{OSAlg}_\Sigma$, unique up to isomorphism, known as the *Initial Algebra.*

2.4 Equations

Properties of specifications are defined using equations.

Definition 2.26. A Σ-*Equation* is a triple (Y,t,t') where $Y \in \mathcal{X}$ and $t,t' \in T_\Sigma(\mathcal{X})$, such that $Vars(t) \cup Vars(t') \subseteq Y$, written $\forall Y.t = t'$. If the variable set $Y = Vars(t) \cup Vars(t')$,

then we write $t = t'$.

A signature is combined with a set of equations to give a specification.

Definition 2.27. A *Specification* consists of a pair (Σ, E) where Σ is an order-sorted signature and E a set of equations.

Example 2.28. Example 2.16 can be extended to a specification of sequences as follows:

Variables:	$n : Nat,\ s_1, s_2, s_3 : Seq,\ ns : NeSeq$	
Equations:	$\forall s_1.\langle\rangle @ s_1 = s_1$	$\forall s_1.s_1 @ \langle\rangle = s_1$
	$\forall s_1, s_2, s_3.(s_1 @ s_2) @ s_3 = s_1 @ (s_2 @ s_3)$	
	$\forall n.tail(n) = \langle\rangle$	$\forall n.head(n) = n$
	$\forall n, s_1.tail(n @ s_1) = s_1$	$\forall ns, s_1.head(ns @ s_1) = head(ns)$
	$\forall ns, s_1.tail(ns @ s_1) = (tail(ns)) @ s_1$	

Note that the *head* and *tail* operators are defined only on non-empty sequences and this is reflected in the sorts of the variables in their defining equations. □

A specification restricts the class of models to those with the specified properties, using the notion of validity.

Definition 2.29. Given a specification $\mathcal{S} = (\Sigma, E)$, an equation $\forall Y.t = t'$ is *Valid* in an Σ-algebra $\mathcal{A}$ written $\mathcal{A} \models \forall Y.t = t'$ as defined in the following:

$$\mathcal{A} \models t = t' \Leftrightarrow \forall (Y, \mathcal{A}) - assignments\ \nu \cdot t^\nu = t'^\nu$$

$\mathcal{A}$ is a *Model* for $\mathcal{S}$ (or $\mathcal{A}$ is a $\mathcal{S}$-*Algebra*) if every $\forall Y.t = t' \in E$ is valid in $\mathcal{A}$.

The class of models for $\mathcal{S}$ and their homomorphisms form a category, $\mathbf{OSAlg}_{\mathcal{S}}$.

Given a specification $\mathcal{S}$, its equational theory is the class of equations which are valid in all models of $\mathcal{S}$. An equation $\forall Y.t = t'$ is *valid* under a specification $\mathcal{S}$, written $\mathcal{S} \models \forall Y.t = t'$, if $\forall Y.s = t$ is valid in every model of $\mathcal{S}$. Care is taken with the variable sets of equations to avoid difficulties with empty sets, as illustrated in the following example.

Example 2.30. Consider the following specification.

Sorts:	*Bool*, *Void*	
Operators:	$false : \rightarrow Bool$	$true : \rightarrow Bool$
	$f : Void \rightarrow Bool$	
Equations:	$f(x : Void) = true$	$f(x : Void) = false$

There are no $(\{x : Void\}, \mathcal{T}_{\Sigma,\emptyset})$-assignments, as $Void^{\mathcal{T}_{\Sigma,\emptyset}}$ is empty. Hence, the given equations are trivially valid in $\mathcal{T}_{\Sigma,\emptyset}$, which is thus a model. However, for every $(\{x : Void\}, \mathcal{T}_{\Sigma,\emptyset})$-assignment, ν, $true^{\nu} = false^{\nu}$. But $true \neq false$ in $\mathcal{T}_{\Sigma,\emptyset}$. □

This problem occurs because of quantification over empty sets. To avoid this difficulty, we keep track of which sorts are empty.

Definition 2.31. A sort $s \in S_\Sigma$ is *Inhabited*, if there is a ground term of sort s. A signature Σ is inhabited if every $s \in S_\Sigma$ is inhabited. The set of inhabited sorts of Σ is denoted $Inhab_\Sigma$.

The inhabitedness of signatures is decidable, and given by the following algorithm.

Lemma 2.32. Given a signature Σ:

1. If $c : \rightarrow s \in \mathcal{F}_\Sigma$ then s is inhabited.
2. If $f : s_1 \dots s_n \rightarrow s \in \mathcal{F}_\Sigma$ and $s_1, \dots, s_n$ are inhabited, then s is inhabited.
3. If $s' \leq s$ and s' is inhabited, then s is inhabited.

Proof (2.32). Immediate from the definition of ground terms of sort s. □

To manipulate the variable set carried by equations, we use the following lemma.

Lemma 2.33. ([Wal92]) Given a Σ-equation $\forall Y.t = t'$. If $s \in S_\Sigma$ is inhabited, or there is a $y : s' \in Y$ such that $s' \leq s$, then for every Σ-algebra $\mathcal{A}$:

$$\mathcal{A} \models \forall (Y \cup \{x : s\}).t = t' \Leftrightarrow \mathcal{A} \models \forall Y.t = t'$$

The problem of empty models is considered in more detail in [GM89, GKK90, Wal92].

2.4.1 Order-Sorted Equational Logic

The *Well-Sorted Congruence Generated by E*, denoted as $s =_E t$, is the smallest relation containing E generated by the deduction rules in Figure 2.1. Under these laws a set of equations E presents an *equational theory.*

Reflexivity. $E \vdash_\Sigma \forall Y.t =_E t$ if t is a Σ-term.

Symmetry. $$\frac{E \vdash_\Sigma \forall Y.s =_E t}{E \vdash_\Sigma \forall Y.t =_E s}$$

Transitivity. $$\frac{E \vdash_\Sigma \forall Y.s =_E t,\ E \vdash_\Sigma \forall Y.t =_E u}{E \vdash_\Sigma \forall Y.s =_E u}$$

Congruence. $$\frac{E \vdash_\Sigma \forall Y.s_1 =_E t_1, \cdots, E \vdash_\Sigma, \forall Y.s_n =_E t_n}{E \vdash_\Sigma \forall Y.f(s_1, \ldots, s_n) =_E f(t_1, \ldots, t_n)}$$ if $f(s_1, \ldots, s_n)$ and $f(t_1, \ldots, t_n)$ are Σ-terms.

Instantiation. $$\frac{E \vdash_\Sigma \forall Y.s = t}{E \vdash_\Sigma \forall X.\sigma s =_E \sigma t}$$ if $\sigma : Y \to T_\Sigma(X)$ is a Σ-substitution, and $s = t \in E$.

Figure 2.1: Rules for Non-Overloaded Order-Sorted Equational Deduction.

Example 2.34. The associative-commutative (AC) equational theory over a single-sorted signature with a single operator "+" is presented by the two equations:

Commutativity: $x + y = y + x$

Associativity: $(x + y) + z = x + (y + z)$

□

Definition 2.35. For a term t the *congruence class* of t under an equational theory E is the set of terms which are equal to t under that theory: $[t]_E = \{t' | \forall Y.t =_E t'\}$.

It can be shown that the models of a specification and the equational theories generated by the equational logic are equivalent. If an equation can be deduced via the rules, it is

valid in all models (*Soundness*), and if an equation is valid in all models, it can be deduced using the rules (*Completeness*).

Theorem 2.36. ([SNGM88]) Order-sorted equational logic is sound and complete:

1. **Soundness.** $E \vdash_\Sigma \forall Y.s =_E t \Rightarrow \mathcal{S} \models \forall Y.s = t$
2. **Completeness.** $\mathcal{S} \models \forall Y.s = t \Rightarrow E \vdash_\Sigma \forall Y.s =_E t$

This justifies the use of equational deduction to generate consequences of a specification.

2.4.2 Quotient Algebra

The free term algebra is not a model for specifications. However, the quotient term algebra for a specification can be constructed upon the free term algebra. Intuitively, this construction identifies those terms which are provably equal.

Definition 2.37. Given a specification $\mathcal{S} = (\Sigma, E)$, we give the *Quotient Term Algebra* over a set of variables $\mathcal{X}$, written $\mathcal{T}_{\mathcal{S},\mathcal{X}}$ by the following construction.

1. $\forall s \in S_\Sigma \cdot s^{\mathcal{T}_{S,\mathcal{X}}} = \{[t]_E | t \in T_\Sigma(\mathcal{X}) \text{ and } t : s\}$
2. $\forall f \in \mathcal{F}_\Sigma \cdot D_f^{\mathcal{T}_{S,\mathcal{X}}} = \{([t_1]_E, \ldots, [t_n]_E) | f(t_1, \ldots, t_n) \in T_\Sigma(\mathcal{X})\}$
3. $f^{\mathcal{T}_{S,\mathcal{X}}}([t_1]_E, \ldots, [t_n]_E) = [f(t_1, \ldots, t_n)]_E$

Theorem 2.38. Given a specification $\mathcal{S} = (\Sigma, E)$, a $\mathcal{S}$-algebra $\mathcal{A}$, and a $(\mathcal{V}, \mathcal{A})$-assignment ν over a set of variables $\mathcal{V}$, the denotation function $\_^\nu : \mathcal{T}_{\mathcal{S},\mathcal{V}} \rightarrow \mathcal{A}$ is a homomorphism. The ground quotient term algebra $\mathcal{T}_{\mathcal{S},\emptyset}$ is the initial model of the category $\mathbf{OSAlg}_\mathcal{S}$.

The initial algebra can be characterised as that which has "no junk" and "no confusion".

- *No Junk.* Every element in the algebra is a denotation of some ground term. Extra elements in the algebra which do not denote some ground term are 'junk'.
- *No Confusion.* Elements are identified in the algebra if and only if they are provably

equal using equational logic. Elements which are identified, but not provably so, are 'confused'.

2.5 Overloaded Order-Sorted Semantics

Goguen and Meseguer [GM87, GM89] give a different approach to order-sorted semantics, also used in [GKK90]. This approach is known as *Overloaded Semantics* as it allows a more general overloading of operators.

Definition 2.39. Given an Order-Sorted Signature Σ with the *Monotonicity* condition:

$$f \in \mathcal{F}_{\Sigma(w_1,s_1)} \text{ and } f \in \mathcal{F}_{\Sigma(w_2,s_2)} \text{ and } w_1 \leq_\Sigma w_2 \text{ then } s_1 \leq_\Sigma s_2$$

Then an *Overloaded* Σ*-Algebra*, $\mathcal{A}$ is defined to be a pair $(S_\Sigma^{\mathcal{A}}, \mathcal{F}_\Sigma{}^{\mathcal{A}})$ such that :

1. $S_\Sigma^{\mathcal{A}} = \{ s^{\mathcal{A}} \mid s \in S_\Sigma \}$.
2. If $s \leq_\Sigma s'$ then $s^{\mathcal{A}} \subseteq s'^{\mathcal{A}}$.
3. For each operator f in $\mathcal{F}_{\Sigma(\langle s_1,\ldots,s_n\rangle,s)}$ there is a function $f^{\mathcal{A}} : s_1^{\mathcal{A}}, \ldots, s_n^{\mathcal{A}} \rightarrow s^{\mathcal{A}}$.
4. If $f \in \mathcal{F}_{\Sigma(w_1,s_1)}$ and $f \in \mathcal{F}_{\Sigma(w_2,s_2)}$ and $w_1 \leq_\Sigma w_2$ then $f^{\mathcal{A}} : w_1^{\mathcal{A}} \rightarrow s_1^{\mathcal{A}}$ equals $f^{\mathcal{A}} : w_2^{\mathcal{A}} \rightarrow s_2^{\mathcal{A}}$ on $w_1^{\mathcal{A}}$.

Thus in overloaded semantics, each operator symbol is denoted by *family* of functions in the algebra, one for each rank. This properly extends the many-sorted approach to algebraic specification as defined in for example [GTW79].

Example 2.40. Given the signature:

Sorts:	$A\ B\ C$	
Subsorts:	$A \leq C,\ B \leq C$	
Operators:	$a : \rightarrow A$	$f : A \rightarrow A$
	$b : \rightarrow B$	$f : B \rightarrow B$

Given a denotation in an overloaded algebra $\mathcal{A}$ such that $A^{\mathcal{A}} \cap B^{\mathcal{A}} \neq \emptyset$, and an element

$e \in A^{\mathcal{A}} \cap B^{\mathcal{A}}$, then $f^{\mathcal{A}}_{A \to A}(e)$ may not be the same as $f^{\mathcal{A}}_{B \to B}(e)$. In the non-overloaded semantics, there is only one possible value for $f^{\mathcal{A}}(e)$. □

However, in overloaded algebras, only coherent signatures can be used.

Definition 2.41. An order-sorted signature Σ is *filtered* if for all distinct pairs of sorts $s, s' \in S_\Sigma$, and $(s, s') \in (\leq \cup \leq^{-1})^+$, then there is a sort $s'' \in S_\Sigma$ such that $s, s' \leq s''$. It is *coherent* if it is filtered and regular.

Without this restriction, the notion of Isomorphism between algebras breaks down and there is no initial algebra. Meseguer and Goguen claim that in practice signatures can be easily made coherent. They also show that a slightly different set of the laws of equational logic is sound and complete with respect to overloaded algebras [GM89], for coherent signatures. However, this has some unexpected consequences, which may not be as the user intended.

Example 2.42. In Example 2.40, add the equation $a = b$. In the overloaded semantics, $f(a) = f(b)$ does not follow as f's denotation may not be the same on each sort. □

The two approaches to the semantics of order-sorted equational logic are very similar, and in most natural examples produce the same results, although differences do occur in subtle ways. The overloaded semantics is more general and as Goguen and Meseguer emphasise, properly extends many-sorted algebra, and also is suited for modelling inheritance in object-oriented languages. However, it is subject to more restrictions, and can have unexpected consequences. For a full discussion of the differences see [GM89, Poi90, Dia91, GD92, Wal92]. Poigné [Poi91] gives an interesting approach which can describe both in the same framework by partitioning the set of sorts, and insisting that the operators are consistent on the partitions; non-overloading uses the trivial partition of the whole set of sorts. The approach used in this thesis is the non-overloaded semantics similar to [Gog84, Smo86, SNGM88, Kir88, Wal89, SS89], and is called the *Standard Model* or *Standard Semantics.*

2.6 Order-Sorted Associative-Commutative Unification

In this section, we consider unification modulo equational theories. Given a set of equations E, we extend the subsumption ordering to the equational theory of E.

Definition 2.43. Given equational theory E, a Σ-term t *E-matches* a Σ-term t' written $t \preceq_E t'$ if there is an *E-matching* substitution σ such that $t' =_E \sigma(t)$. We extend this to substitutions. For Σ-substitutions σ and τ, if there exists a substitution λ such that $\forall x \in \mathcal{X}.\lambda(\sigma(x)) =_E \tau(x)$, then $\sigma \preceq_E \tau$. This can be restricted to a set of variables $\mathcal{W} \subseteq \mathcal{X}$. $\sigma \preceq_E^{\mathcal{W}} \tau$ if there exists a substitution λ such that $\forall x \in \mathcal{W} \cdot \lambda(\sigma(x)) =_E \tau(x)$. Substitutions, σ and τ are equal with respect to E and set of variables $\mathcal{W} \subseteq \mathcal{X}$, written $\sigma =_E^{\mathcal{W}} \tau$ if and only if $\sigma \preceq_E^{\mathcal{W}} \tau$ and $\tau \preceq_E^{\mathcal{W}} \sigma$.

We can now define order-sorted equational unification.

Definition 2.44. Given an order-sorted specification (Σ, E), an *E-Unifier* of Σ-terms t and t' is a Σ-substitution σ such that $\sigma(t) =_E^{\mathcal{W}} \sigma(t')$, where $Vars(t = t') \in \mathcal{W}$.

An E-Unifier of a set of equations Γ is a Σ-substitution σ such that $\forall(t = t') \in \Gamma \cdot \sigma(t) =_E^{\mathcal{W}} \sigma(t')$, where $Vars(\Gamma) \in \mathcal{W}$. The set of E-unifiers of a set of equations Γ is denoted $U_E(\Gamma)$.

In general, $U_E(\Gamma)$ is an infinite set of substitutions. However, for many equational theories a set which characterises all unifiers of a equation set Γ can be given.

Definition 2.45. Given equational theory E, a *Complete set of E-unifiers* of a set of equations Γ on $Vars(\Gamma) \subseteq \mathcal{W} \subseteq \mathcal{X}$, $CSU_E^{\mathcal{W}}(\Gamma)$, is a set of Σ-substitutions such that:

1. $\forall \sigma \in CSU_E^{\mathcal{W}} \cdot Im(\sigma) \cap \mathcal{W} = \emptyset$
2. $CSU_E^{\mathcal{W}}(\Gamma) \subseteq U_E(\Gamma)$
3. $\forall \tau \in U_E(\Gamma) \cdot \exists \sigma \in CSU_E^{\mathcal{W}}(\Gamma).\sigma \preceq_E^{\mathcal{W}} \tau$

In addition, a complete set of unifiers is known as *minimal*, denoted $\mu CSU_E^{\mathcal{W}}(\Gamma)$, if it is a complete set and $\forall \sigma, \tau \in \mu CSU_E^{\mathcal{W}}(\Gamma) \cdot \ \sigma \preceq_E^{\mathcal{W}} \tau \Rightarrow \sigma =_E^{\mathcal{W}} \tau$.

The problem of unifying t and t' over an equational theory E is that of finding a minimal complete set of E-unifiers for $\{t \stackrel{?}{=} t'\}$. Such sets of E-unifiers may not exist ([FH83]), and if they do, they may be infinite, as for example the associative theory [Plo72]. However, the commutative and associative-commutative theories do have finite most general sets of minimal unifiers [Fag87, For87], although they can be very large: Domenjoud [Dom92] demonstrates that $x+x+x+x = y_1+y_2+y_3+y_4$ has 34 359 607 481 minimal AC-unifiers.

2.6.1 Conditions for Unification

Before discussing the generation of order-sorted AC-unifiers, we give the conditions on specifications which ensure finite sets of unifiers in the order-sorted AC theory.

Combination of Theories. It is not straightforward to combine unification algorithms for equational theories (including the empty theory). However, Yelick [Yel85] defines two conditions on the equational theory which ensure a straightforward combination.

Definition 2.46. A set of equations E is *Collapse Free* if there is no $t = t' \in E$ such that $t \in \mathcal{X}$ or $t' \in \mathcal{X}$. A set of equations E is *Regular* if for all $t = t' \in E$, $Vars(t) = Vars(t')$.

Both the commutative and associative-commutative theories are collapse-free and regular.

In an order-sorted signature, different equational theories could apply to different parts of the signature. For example, an operator could be AC over some sorts, but only commutative over others, as considered in [Kir88, Bou92]. However, in this thesis we ignore this and assume an equational theory applies to all well-formed terms.

Regularity. Consider the following unification problem.

Example 2.47. Given the signature.

Sorts:	$A\ B$	
Operators:	$c : \rightarrow A$	$f : A \rightarrow A$
	$c : \rightarrow B$	$f : B \rightarrow B$

and the unification problem $\{x : A \stackrel{?}{=} y : B\}$. $\{x : A \mapsto y : B\}$ is not sort-preserving. However, the following infinite set of substitutions are all sort-preserving and minimal:

$$\{x : A \mapsto f^n(c), y : B \mapsto f^n(c)\} \textit{ where } n = 0, 1, 2, \ldots$$

This infinite set of minimal unifiers occurs because in all Σ-algebras the sets representing A and B intersect on $f^n(c)$ and so the equation $x : A = y : B$ can have meaningful solutions, but this is not reflected in the syntax as the terms $f^n(c)$ have no unique least sort. □

This problem is caused because the signature is not regular[2]. By restricting the unification problem to regular signatures the following theorem holds.

Theorem 2.48. ([SNGM88]) If Σ is regular, unification over the empty theory is finitary.

Sort Compatible Theories. We consider whether the commutative or associative-commutative theory applies to all well-formed terms.

Example 2.49. Given the signature:

Sorts:	$A\ B$	
Operators:	$a : \rightarrow A$	$b : \rightarrow B$
	$\_+\_ : A\ B \rightarrow A$	

and '+' is commutative. However, $y : B + x : A$ is not a Σ-Term. □

The initial model of the specification does not respect the commutativity axiom for '+'; the commutativity axiom cannot even be presented as an equation. The following construction tests whether the AC theory is well-defined.

Definition 2.50. The set of commutativity equations C for a signature Σ and operator $+$ is as follows. For each $s_1, s_2 \in S_\Sigma$, generate new variables $x_1 : s_1, x_2 : s_2$ and construct the terms $t_1 = x_1 + x_2$, $t_2 = x_2 + x_1$. If $t_1, t_2 \in T_\Sigma(\mathcal{X})$ then $t_1 = t_2 \in C$; if $t_1, t_2 \notin T_\Sigma(\mathcal{X})$ then $t_1 = t_2 \notin C$. If for some s_1, s_2, t_1 (respectively t_2) $\in T_\Sigma(\mathcal{X})$ and t_2 (respectively t_1) $\notin T_\Sigma(\mathcal{X})$ then $C = \emptyset$. If $C \neq \emptyset$ then $+$ is commutative with the specification (Σ, C).

[2]This is not to be confused with the regularity of the equational theory as defined above.

The set of associativity axioms A is as follows. For each $s_1, s_2, s_3 \in S_\Sigma$, generate new variables $x_1 : s_1, x_2 : s_2, x_3 : s_3$ and construct the terms $t_1 = (x_1 + x_2) + x_3$, $t_2 = x_2 + (x_1 + x_3)$. If $t_1, t_2 \in T_\Sigma(\mathcal{X})$ then $t_1 = t_2 \in A$; if $t_1, t_2 \notin T_\Sigma(\mathcal{X})$ then $t_1 = t_2 \notin A$. If for some s_1, s_2, t_1 (respectively t_2) $\in T_\Sigma(\mathcal{X})$ and t_2 (respectively t_1) $\notin T_\Sigma(\mathcal{X})$ then $A = \emptyset$. If $A \neq \emptyset$ then $+$ is associative with the specification (Σ, A). If $A \neq \emptyset$ and $C \neq \emptyset$ then $+$ is associative-commutative with specification (Σ, AC) where $AC = A \cup C$.

However, this is not sufficient to ensure a well-sorted congruence.

Example 2.51. Given the signature:

Sorts:	$A\ B$	
Subsorts:	$B \leq A$	
Operators:	$f : B \to B$	$\_+\_ : A\ A \to A$
	$b : \to B$	$\_+\_ : A\ B \to B$
	$a : \to A$	

and "+" is commutative then:

$$C = \{x_1 : A + x_2 : A = x_2 : A + x_1 : A,\ x_1 : A + x_2 : B = x_2 : B + x_1 : A\}$$

$\mathcal{LS}(x : B + y : A) = A$, $\mathcal{LS}(y : A + x : B) = B$, so $f(a + b) \in T_\Sigma(\mathcal{X})$, $f(b + a) \notin T_\Sigma(\mathcal{X})$. □

Equational variants need to be valid in every context, given by the sort-compatibility condition on equational theories, as discussed by Schmidt-Schauss [SS86, SS89].

Definition 2.52. An equational theory given by equations E is *Sort-Compatible* with signature Σ if $\forall s, t \in T_\Sigma(\mathcal{X}) \cdot s =_E t \Rightarrow \mathcal{LS}(s) = \mathcal{LS}(t)$.

Example 2.53. Given the signature:

Sorts:	$A\ B\ C$	
Subsorts:	$B \leq A, C \leq A$	
Operators:	$\_+\_ : A\ A \to A$	$\_+\_ : B\ C \to C$
	$\_+\_ : C\ B \to C$	

and "+" commutative then $x : B + y : C =_E y : C + x : B$, $\mathcal{LS}(x : B + y : C) = C$ and $\mathcal{LS}(y : C + x : B) = C$. The signature is sort-compatible with "+" commutativity. □

For a equational theory to be sort-compatible with a signature, the sorts and arities of the operators must form a model for the equational theory. For regular signatures, sort-compatibility is decidable.

Theorem 2.54. ([SS89]) Given a regular signature Σ, and a set of equations E, if for all equations $s = t \in E$ and for all renaming substitutions ρ

$$\mathcal{LS}(\rho(s)) = \mathcal{LS}(\rho(t))$$

then E is sort-compatible with respect to Σ.

For a finite specification, sort-compatibility is decidable.

Corollary 2.55. The commutativity of $+ \in \mathcal{F}_\Sigma$ is sort-compatible with signature Σ, if the partial function $S_+ : S_\Sigma\ S_\Sigma \rightarrow S_\Sigma$ given by $S_+(s, s') = \mathcal{LS}(x : s + y : s')$ for any $x, y \in \mathcal{X}$ of sorts s, s', where $\exists\_ + \_ : w \rightarrow r \in \mathcal{F}_\Sigma \cdot \langle s, s' \rangle \leq_\Sigma w$, is commutative.

The associativity and commutativity of $+ \in \Sigma$ is sort-compatible with Σ, if the function $S_+ : S_\Sigma\ S_\Sigma \rightarrow S_\Sigma$ given by $S_+(s, s') = \mathcal{LS}(x : s + y : s')$ for any $x, y \in \mathcal{X}$ of sorts s, s' where $\exists\_ + \_ : w \rightarrow r \in \mathcal{F}_\Sigma . \langle s, s' \rangle \leq_\Sigma w$ is an Abelian Semigroup over S_Σ.

Example 2.56. [Examples 2.51 and 2.53 revisited.] In Example 2.51 , $S_+(B, A) = A$ and $S_+(A, B) = B$ and so commutativity is incompatible. In Example 2.53:

$$\begin{aligned} S_+(A,B) &= A = S_+(B,A) \\ S_+(A,C) &= A = S_+(C,A) \\ S_+(C,B) &= A = S_+(B,C) \end{aligned}$$

and so the function S_+ is commutative. □

Example 2.57. Given the signature:

Sorts:	$A\ B$
Subsorts:	$B \leq A$
Operators:	$\_+\_ : A\ A \rightarrow B \qquad \_+\_ : B\ B \rightarrow A$

where "+" is associative and commutative. The function $S_+ : S_\Sigma\ S_\Sigma \rightarrow S_\Sigma$ given by $S_+(s, s') = \mathcal{LS}(x : s + y : s')$ does not form an Abelian semigroup over S_Σ. Consider

$$(x_1 : A + x_2 : A) + x_3 : B =_E x_1 : A + (x_2 : A + x_3 : B)$$

in this case $\mathcal{LS}((x_1 : A + x_2 : A) + x_3 : B) = A$ and $\mathcal{LS}(x_1 : A + (x_2 : A + x_3 : B)) = B$. Hence this specification is not AC-sort-compatible. □

If the sort-compatible property holds, then the different semantics of [GM89] and [SNGM88] result in the same unifiers; if not they can be different [Wal89].

Example 2.58. Given the signature:

Sorts:	$A\ B\ C$	
Subsorts:	$B \leq A, C \leq A$	
Operators:	$f : B \rightarrow B$	$\_+\_ : A\ A \rightarrow A$
	$f : C \rightarrow C$	$\_+\_ : B\ C \rightarrow C$
	$c : \rightarrow C$	$\_+\_ : C\ B \rightarrow B$
	$b : \rightarrow B$	

where "+" is commutative. This is a well-defined commutative theory but not sort-compatible. Consider unifying $f(x_1 : B + y_1 : C) = f(y_2 : C + x_2 : B)$. In the non-overloaded semantics, there is a single most general unifier $\{x_1 \rightarrow x_2, y_1 \rightarrow y_2\}$. But in the overloaded semantics, there are no most general unifiers as a model $\mathcal{A}$ may exist where $f^{\mathcal{A}}(x_1^\nu : B +^{\mathcal{A}} y_1^\nu : C) \neq f^{\mathcal{A}}(y_2^\nu : C +^{\mathcal{A}} x_2^\nu : B)$ for some $\mathcal{A}$-assignment ν on the variables. □

2.6.2 An Abstract Unification Algorithm.

In the view of unification originally developed by Herbrand [Her30, Her67] and rediscovered by Martelli and Montenari [MM82], generating minimal unifiers is seen as a set of rules

which transform a set of equations while preserving the minimal unifiers. The equations are transformed into solved forms.

Definition 2.59. An equation is in *Solved Form* if it is of the form $x = t$ where $x \in \mathcal{X}_s$ is a variable of sort s, and $t : s \in T_\Sigma(\mathcal{X})$ such that $x \notin Vars(t)$.

(1) Decomposition.

$$\frac{\{\{f(t_1,\ldots,t_n)\stackrel{?}{=}f(t'_1,\ldots,t'_n)\}\cup U\}\cup C}{\{\{t_1\stackrel{?}{=}t'_1,\ldots,t_n\stackrel{?}{=}t'_n\}\cup U\}\cup C} \text{ if } f\in\mathcal{F}_{\Sigma d}$$

(2) Conflict.

$$\frac{\{\{f(t_1,\ldots,t_n)\stackrel{?}{=}g(t'_1,\ldots,t'_m)\}\cup U\}\cup C}{C} \text{ if } f,g\in\mathcal{F}_{\Sigma d} \text{ and } f\neq g$$

(3) Trivial Equation.

$$\frac{\{\{t\stackrel{?}{=}t\}\cup U\}\cup C}{\{U\}\cup C}$$

(4) Occurs Check.

$$\frac{\{\{x\stackrel{?}{=}t\}\cup U\}\cup C}{C} \text{ if } x\in\mathcal{X},\ t\notin\mathcal{X} \text{ and } x\in Vars(t).$$

(5) Eliminate.

$$\frac{\{\{x : s\stackrel{?}{=}t : s'\}\cup U\}\cup C}{\{\{x\stackrel{?}{=}t\}\cup\{x\mapsto t\}(U)\}\cup C} \text{ if } x\notin Vars(t),\ x\in Vars(U) \text{ and } s'\leq s$$

(6) Mutation.

$$\frac{\{\{f(t_1,\ldots,t_n)\stackrel{?}{=}g(t'_1,\ldots,t'_m)\}\cup U\}\cup C}{MUT_E(\{f(t_1,\ldots,t_n)\stackrel{?}{=}g(t'_1,\ldots,t'_m)\}\cup U)\cup C} \text{ if one of } f,g\in\mathcal{F}_{\Sigma E}$$

(7) Intersect.

$$\frac{\{\{x : s\stackrel{?}{=}y : s'\}\cup U\}\cup C}{\{\{x\stackrel{?}{=}z : s'',\ y\stackrel{?}{=}z : s''\}\cup U|s''\in s\wedge s'\}\cup C} \text{ if } x,y\in Vars(U) \text{ and } s'\bowtie s$$

(8) Remove.

$$\frac{\{\{x : s\stackrel{?}{=}y : s'\}\cup U\}\cup C}{C} \text{ if } x,y\in Vars(U) \text{ and } s'\bowtie s \text{ and } s'\wedge s=\emptyset$$

(9) Abstract.

$$\frac{\{\{x : s\stackrel{?}{=}f(t_1,\ldots,t_n)\}\cup U\}\cup C}{\{\{z_1 : s_1\stackrel{?}{=}t_1,\ldots,z_n : s_n\stackrel{?}{=}t_n, x\stackrel{?}{=}f(z_1,\ldots,z_n)\}\cup U|s_1,\ldots,s_n\to s'\in\Sigma^s_f\}\cup C}$$

if $\mathcal{LS}(f(t_1,\ldots,t_n))\not\leq s$ and
$\Sigma^s_f=\{w>s'|f : w\to s'\in\Sigma\wedge s'\leq s\wedge w \textit{ maximal}\}$

Figure 2.2: Rules for Order-Sorted Equational Unification

A most general unifier can easily be inferred from a set of equations all in solved form.

Theorem 2.60. Given a set of equations Γ all in solved form, $\{x_1 = t_1, \ldots, x_n = t_n\}$ say, such that $x_i \notin \mathit{Vars}(t_j)$ for $i, j = 1, \ldots n$, $\mu CSU_E^{\mathcal{W}}(\Gamma)$ is given by $\sigma = \{x_n \mapsto t_n, \ldots, x_1 \mapsto t_1\}$ where $\mathcal{W} = \{x_1, \ldots, x_n\}$.

We give a set of transformation rules in Figure 2.2 for order-sorted equational unification similar [Kir88] to give an abstract algorithm which does not assume any particular control. In [SS86] and [MGS89, SNGM88] the algorithms first generate an unsorted solution and then analyse the sort structure to find the order-sorted unifiers, and this may result in performing that unnecessary work in the case of sort clash. Giving the algorithm as transformation rules allows the development of alternative strategies which interleave the structural, equational and sort decomposition rules.

The rules transform sets of *candidate* sets; in the rules other candidate sets are represented by C, and other equations in the same unifier by U. To solve a unification problem $t \stackrel{?}{=} t'$ we transform the set $\{\{t \stackrel{?}{=} t'\}\}$. The transformations preserve the unifiers of the candidates. We assume that the signature is inhabited, and regular, and the equational theory is regular, collapse free and sort-compatible.

The set of operators in $\mathcal{F}_\Sigma$ is divided into two sets: $\mathcal{F}_{\Sigma E}$ which appear in the equations defining the equational theory, and $\mathcal{F}_{\Sigma d}$ for other *free symbols.* The unifier of terms with the same free symbol at root is the composition of the unifiers of the corresponding subterms and, if the top free symbols differ, no unifier is possible. If the two terms are the same then the unifier is trivial. If a variable occurs on both sides of an equation circularities occur and no unifier is possible. If a variable occurs in the left hand side of a solved form, it can be eliminated from other equations in that candidate.

Rules 1-5 alone reduce the sets of equations to equations of the form:

1. $f(t_1, \ldots, t_n) = g(t'_1, \ldots, t'_m)$ where one of $f, g \notin \mathcal{F}_d$, or

2. $x = t$ where $x \in \mathcal{X}$ and $x \notin \mathit{Vars}(t)$.

MUT_E in rule 6 is a specialised operation for the particular equational theory. For theories other than AC see the surveys of Siekmann [Sie89], and Jouannaud and Kirchner [JK91].

Commutative mutation is simple, and is given as a transformation rule in Figure 2.3.

The mutation operation for AC-theories is complex and cannot be simply described in terms of a simple inference rule. The central part of the algorithm is the solution of a set of linear Diophantine equations to form a basis. Solutions to AC unification are then generated by considering subsets of this basis. This has an interesting consequence for order-sorted solutions. New variables are introduced in the subset resolution phase of the algorithm, and should be sorted. However, at this stage the sorts of variables cannot be given: this is established by the sort-resolution rules 7-9. To overcome this, we adopt a solution similar to that of Meseguer, Goguen and Smolka [MGS89]. A new sort *Top* is introduced such that $\forall s \in S_\Sigma \cdot c \leq Top$. All new variables are said to be of this sort and later weakened to be of a declared sort, if possible. The AC mutation operation is not discussed further in this thesis; for more details see, for example, [Sti81, Fag87, For87, LC89, AK90, BCD90].

(6a) Commutative Mutation.

$$\frac{\{\{s_1 + s_2 \stackrel{?}{=} t_1 + t_2\} \cup U\} \cup C}{\{\{s_1 \stackrel{?}{=} t_1, s_2 \stackrel{?}{=} t_2\} \cup U\} \cup \{\{s_1 \stackrel{?}{=} t_2, s_2 \stackrel{?}{=} t_1\} \cup U\} \cup C} \quad \text{if } + \text{ commutative (only)}$$

Figure 2.3: Commutative Mutation Rule

Rules 7-9 carry out the conversion of equations of the form $x = t$, where $x \in \mathcal{X}$, into solved forms, which may result in more than one unifier.

Example 2.61. ([CD85]) Given the signature for lists with an append function:

Sorts:	*List NeList*	
Subsorts:	$NeList \leq List$	
Operators:	$\_@\_ : List\ List \rightarrow List$	$\_@\_ : NeList\ List \rightarrow NeList$
	$\_@\_ : List\ NeList \rightarrow NeList$	$\_@\_ : NeList\ NeList \rightarrow NeList$

unify $\{x : NeList \stackrel{?}{=} y : List \ @ \ z : List\}$. The Abstract Rule can be applied in two ways, either using the second rank for @, and substitute y for a variable of sort *NeList*, or use the third rank to replace z similarly. Thus there are two unifying substitutions.

$$\{x \mapsto (y_1 : NeList)@z,\ y \mapsto y_1\}$$

$$\{x \mapsto y@(z_1 : NeList),\ z \mapsto z_1\}$$

Using the fourth rank to replace both variables would be subsumed by either of the above and therefore would not form a minimal solution. □

The Intersect rule handles the case where two variables are of incomparable sorts. They are equated in alternative unifiers to a variable of a maximal common subsort. The Remove rule applies when there are no common subsorts of the variables, resulting in a type clash and the candidate is rejected. The Abstract rule deduces feasible sorts from the signature for subterms where $x \stackrel{?}{=} f(t_1, \ldots, t_n)$ is not a solved-form.

2.7 Order-Sorted Term Rewriting

Rewrite rules give an operational interpretation of equations to automate equational proofs, by replacing one side of a rule with the other in terms. However, some equational theories are not suitable for treatment as rewrite systems. A common example is commutativity, which generates a non-terminating rewrite relation: the commutative axiom will rewrite the ground term $a + b$ indefinitely: $a + b \rightarrow b + a \rightarrow a + b \rightarrow \cdots$. Similarly associativity may require an infinite set of rules to generate a decision procedure [PS81]. Alternative methods have been developed to handle such theories. One approach, ordered or unfailing completion, delays the orientation of rules until their application to particular instances [HR87, MN90b, Bac91]. Another approach builds the equational theory into the rewrite relation [Plo72, LB77, BL79, Hue80, PS81, JK86, GKK90, Bac91], and in this section we sketch this approach, giving results for order-sorted term-rewriting modulo an equational theory.

Definition 2.62. An *order-sorted rewrite rule* is an order-sorted equation $\forall Y.l \rightarrow r$ where $Vars(r) \subseteq Vars(l)$. A *rewriting system modulo* E is a specification $\mathcal{R} = (\Sigma, E \cup R)$ where E is a set of equations and R is a set of rules.

Definition 2.63. A rewriting system $\mathcal{R} = (\Sigma, E \cup R)$ defines a *rewriting relation* on Σ-terms. t *rewrites* to t', written $t \rightarrow_R t'$ if and only if there is some path p of t, a rule $l \rightarrow r \in R$, and a matching substitution σ such that $t|_p = \sigma l$ and $t' = t[p \leftarrow \sigma r]$.

$\mathcal{R}$ also defines a *rewriting relation modulo* E on Σ-terms. t *rewrites modulo* E to t', written $t \rightarrow_{R.E} t'$ if and only if there is some path p of t, a rule $l \rightarrow r \in R$, and a E-matching substitution σ such that $t|_p =_E \sigma l$ and $t' = t[p \leftarrow \sigma r]$.

$\mathcal{R}$ defines a *rewriting relation on E-equivalence classes* $\rightarrow_{R/E}$ defined as: $=_E o \rightarrow_R o =_E$

For each of these rewriting relations, if there is no t' such that $t \rightarrow t'$, then t is in *normal form* with respect to that relation. If $t \xrightarrow{*} t'$ and t' is in normal form, then we write $t' = t\downarrow$.

Clearly $\rightarrow_{R.E} \subseteq \rightarrow_{R/E}$, although $\rightarrow_{R.E}$ is a more tractable relation. To have a decidable rewrite system, these relations should terminate.

Definition 2.64. A rewrite relation R is *terminating* if there exist no infinite chains of Σ-terms $t_1 \rightarrow_R t_2 \rightarrow_R t_3 \rightarrow_R \cdots$.

Termination is commonly established by *termination orderings*, which are well-founded orderings on terms, by demonstrating that the rewriting relation $\rightarrow_R$ (or $\rightarrow_{R.E}$, or $\rightarrow_{R/E}$) is contained within some termination ordering $>_t$, $\rightarrow_R \subseteq >_t$, and thus is well-founded also. There is an extensive literature on termination (see for many useful results [Der87]) and termination orderings, including the Knuth-Bendix Ordering [KB70], Polynomial Orderings [BL87] and the Recursive Path Ordering [Der87]. For the purposes of this thesis, we shall assume the existence of a termination ordering $>_t$ on terms.

Rewriting generates a symmetric relation known as Equational Replacement.

Definition 2.65. Equational Replacement. Two terms t and t' replace one another in rewriting system $\mathcal{R} = (\Sigma, E \cup R)$, written $t \longleftrightarrow_{\mathcal{R}} t'$ if there is an equation $l = r \in E \cup R$, a path p and a substitution σ such that $t|_p = \sigma(l)$ and $t' = t[p \leftarrow \sigma(r)]$

This relation is intended to be equivalent to equality. However, it is not in general the case that $u =_{\mathcal{R}} t$ if and only if $u \leftrightarrow^*_{\mathcal{R}} t$ for order-sorted rewriting.

Example 2.66. Smolka's Example [SNGM88]. Given rewriting system $\mathcal{R}$:

Sorts:	$A\ B$	
Subsorts:	$A \leq B$	
Operators:	$a : \rightarrow A$	$b : \rightarrow B$
	$a' : \rightarrow A$	$f : A \rightarrow A$
Rules:	$a \rightarrow b$	$a' \rightarrow b$

This system is confluent and terminating using rewriting over the empty theory. It is provable that $f(a) =_{\mathcal{R}} f(a')$. However, $f(a) \not\rightarrow^*_{\mathcal{R}} f(a')$. □

Thus order-sorted rewriting is incomplete. The term $f(b)$ is not a well-formed $\mathcal{R}$-term and the rewriting system cannot replace through it. The problem is caused by replacing a term of a lower sort by one of a higher. Thus the right-hand side cannot always be substituted for the left. However, the rewriting relation is complete if all rules are *compatible* [SNGM88].

Definition 2.67. A term rewriting system $\mathcal{R}$ is *compatible* if $\forall t \in T_\Sigma(\mathcal{X})$, if $t \rightarrow_{R.E} t'$ then $t' \in T_\Sigma(\mathcal{X})$.

Informally, this ensures that term-rewriting can never form ill-sorted terms. This condition is decidable by means of a test on the rewrite rules of $\mathcal{R}$ [Wal90].

Theorem 2.68. Birkhoff's Theorem for Order-sorted Rewriting.
If $\mathcal{R}$ is a compatible rewriting system then:

$$R \cup E \vdash_\Sigma \forall \mathcal{X} \cdot t_1 = t_2 \text{ if and only if } t_1 \overset{*}{\longleftrightarrow}_{\mathcal{R}} t_2$$

To use the rewrite relation for automatic proofs, we would like to establish the Church-Rosser property. If we have established the Church-Rosser property, then to prove automatically that an equality $R \cup E \vdash_\Sigma t_1 = t_2$, then we only need rewrite $t_1 \overset{*}{\longrightarrow}_{\mathcal{R}} t_1\!\downarrow$, $t_2 \overset{*}{\longrightarrow}_{\mathcal{R}} t_2\!\downarrow$ and then test whether $t_1\!\downarrow =_E t_2\!\downarrow$.

Definition 2.69. A rewrite relation $\rightarrow_{R.E}$ is *Church-Rosser modulo* E if for any Σ-terms, t and t', $t \overset{*}{\longleftrightarrow}_{\mathcal{R}} t'$, there are Σ-terms u, v such that $t \overset{*}{\longrightarrow}_{R.E} u$ and $t' \overset{*}{\longrightarrow}_{R.E} v$ and $u =_E v$.

This is established by showing equivalence of this property to confluence.

Definition 2.70. A rewrite relation $\rightarrow_{R.E}$ is *confluent modulo* E if $\forall t, t_1, t_2 \in T_\Sigma(\mathcal{X})$, such that $t_1 \ {}_{R.E}\overset{*}{\leftarrow} t \overset{*}{\rightarrow}_{R.E} t_2$ then $\exists t_1', t_2' \in T_\Sigma(\mathcal{X})$ such that $t_1 \overset{*}{\rightarrow}_{R.E} t_1' =_E t_2' \ {}_{R.E}\overset{*}{\leftarrow} t_2$ (see Figure 2.4a).

If we restrict ourselves to compatible rewrite systems, then the equivalence of confluence and the Church-Rosser properties can be established.

Definition 2.71. A rewrite relation $\rightarrow_{R.E}$ is *locally confluent modulo* E if $\forall t, t_1, t_2 \in T_\Sigma(\mathcal{X})$, such that $t_1 \ {}_{R.E} \leftarrow t \rightarrow_{R.E} t_2$ then $\exists t_1', t_2' \in T_\Sigma(\mathcal{X})$ such that $t_1 \overset{*}{\rightarrow}_{R.E} t_1' =_E t_2' \ {}_{R.E}\overset{*}{\leftarrow} t_2$ (see Figure 2.4b).

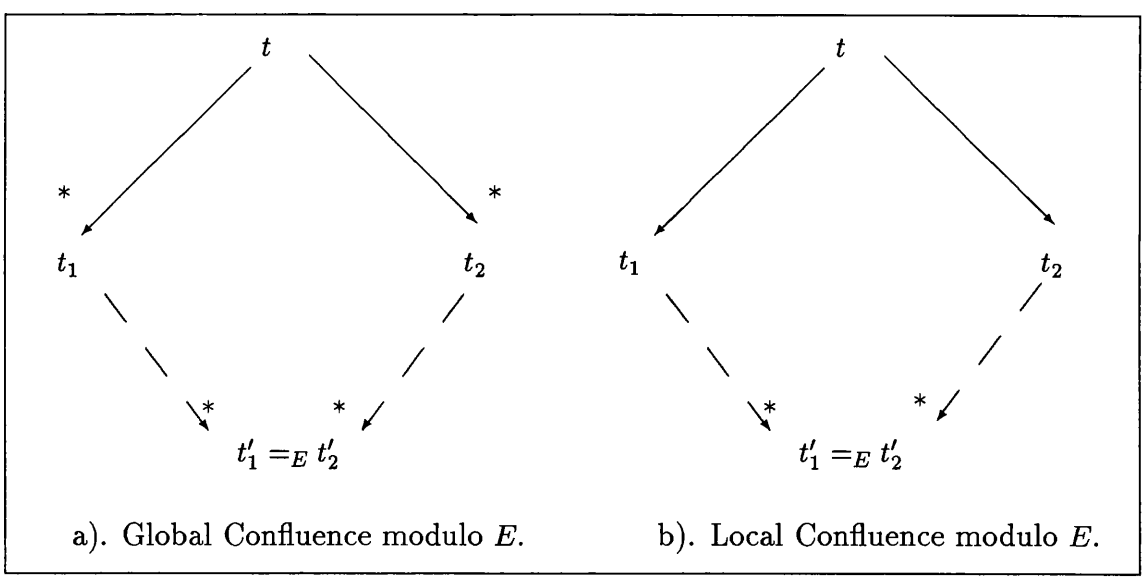


Figure 2.4: Confluence Conditions for the Rewrite Relation.

Also, in rewriting modulo an equational theory the following properties are important.

Definition 2.72. A rewrite relation $\rightarrow_{R.E}$ is *coherent modulo* E if $\forall t, t_1, t_2 \in T_\Sigma(\mathcal{X})$, such that $t_1 \ {}_{R.E}\overset{*}{\leftarrow} t =_E t_2$ then $\exists t_1', t_2' \in T_\Sigma(\mathcal{X})$ such that $t_1 \overset{*}{\rightarrow}_{R.E} t_1' =_E t_2' \ {}_{R.E}\overset{*}{\leftarrow} t_2$. (see Figure 2.5a).

Definition 2.73. A rewrite relation $\rightarrow_{R.E}$ is *locally coherent modulo* E if $\forall t, t_1, t_2 \in T_\Sigma(\mathcal{X})$, such that $t_1 \ {}_{R.E} \leftarrow t =_E t_2$ then $\exists t_1', t_2' \in T_\Sigma(\mathcal{X})$ such that $t_1 \overset{*}{\rightarrow}_{R.E} t_1' =_E t_2' \ {}_{R.E}\overset{*}{\leftarrow} t_2$.

(see Figure 2.5b).

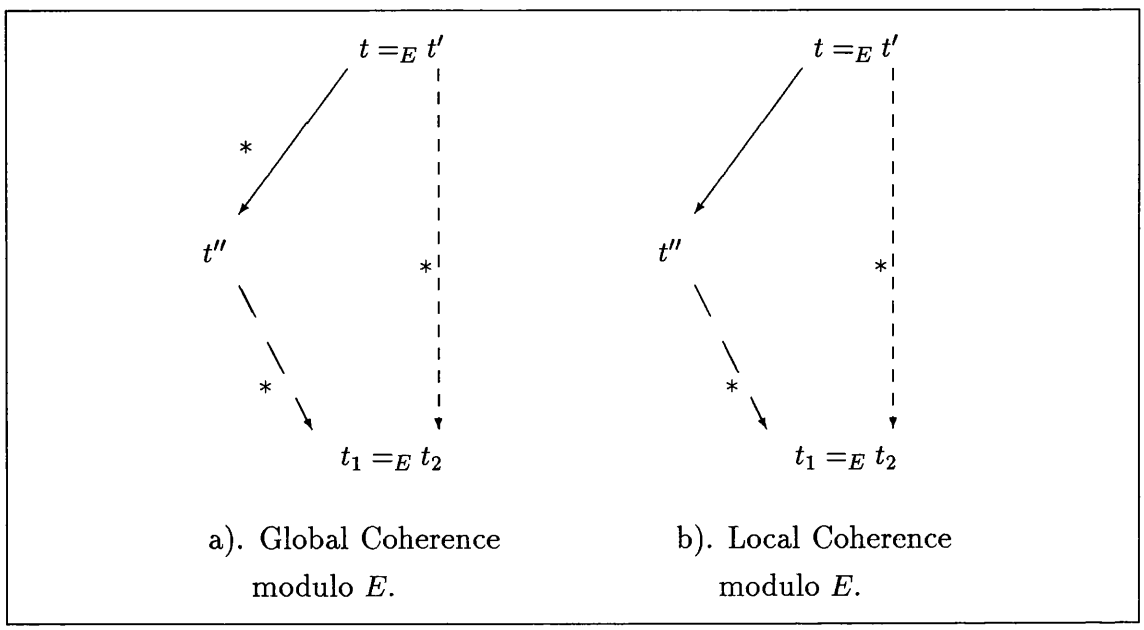


Figure 2.5: Coherence Conditions for the Rewrite Relation.

The following lemma, a modification of Newman's Lemma [New42], establishes the relationship between these properties.

Lemma 2.74. A terminating rewrite relation $\rightarrow_{R.E}$ is confluent and coherent modulo E if and only if it is locally confluent and locally coherent modulo E.

In order to demonstrate the local confluence of the rewrite relation, we consider all possible peaks that can occur within $\overset{*}{\longleftarrow}_{R.E}$, that is terms which can be rewritten by rewrite rules in two different ways. Some peaks are formed by rewrites occurring at incomparable paths, or at so called variable overlaps, and such peaks can always be replaced by a rewriting proof. However, peaks formed by overlapping rewrite rules are more complex; such peaks are instances of critical pairs.

Definition 2.75. Given a set of rewrite rules R, and equational theory E, a *critical pair* of variable disjoint rules $l \rightarrow r$, $g \rightarrow d$ is defined to be $(\sigma(r), \sigma(l[p \leftarrow d]))$ where there exists some path p in l and Σ-substitution σ such that $\sigma(l|_p) =_E \sigma(g)$, that is the terms E-Unify. A critical pair (t, u) is *trivial* if there exists t', u' such that $t \overset{*}{\longrightarrow}_{R.E} t' =_E u' {}_{R.E}\overset{*}{\longleftarrow} u$. $CP_E(R)$ is the set of all critical pairs of R under the equational theory E.

However, in the order-sorted case, it is not sufficient to consider critical pairs.

Example 2.76. ([SNGM88]) Consider the following compatible rewriting system:

Sorts:	$A\ B$	
Subsorts:	$A \leq B$	
Operators:	$a : \rightarrow A$	$f : A \rightarrow A$
	$b : \rightarrow B$	$f : B \rightarrow B$
Rules:	$a \rightarrow b$	$f(x : A) \rightarrow x : A$

There are no non-trivial critical pairs of these rewrite rules. However, it is not locally confluent as $a \leftarrow f(a) \rightarrow f(b)$, but a and $f(b)$ do not rewrite to the same term. □

To establish the critical pair lemma, we have to show that the set of rewrite rules is (weakly) sort decreasing, The following definition is given by Schmidt-Schauss [SS89].

Definition 2.77. A rewriting system $\mathcal{R}$ is *weakly sort-decreasing* if for every $\mathcal{R}$-term t of sort s, such that $t \rightarrow_{\mathcal{R}} u$ then $u \rightarrow^*_{\mathcal{R}} v$ where v is of sort s.

A stronger condition is sort-decreasingness.

Definition 2.78. A rewriting system $\mathcal{R}$ is *sort-decreasing* if for every $\mathcal{R}$-term t of sort s, and $t \rightarrow_{\mathcal{R}} u$ then u is also of sort s.

This is decidable, with an algorithm given by Waldemann [Wal90], and more simply in a regular signature [SNGM88]. Clearly, if a set of rules is sort-decreasing, it is compatible.

The following result is an extension of the key result of Knuth and Bendix [KB70].

Lemma 2.79. Critical Pair Lemma. ([SNGM88, GKK90]). Let R be a sort-decreasing rewriting system. Then R is locally confluent modulo E if and only if all critical pairs of R are trivial.

Thus if we can construct a set of rewrite rules, equivalent to the initial set of rules, where

all critical pairs are trivial, then we have a locally confluent, and thus a confluent and Church-Rosser rewrite relation. Thus, to prove equivalence of term in the algebra, we need only test whether they rewrite to identical normal forms.

(1) Delete	$\dfrac{(E \cup \{t = t'\}, R, S)}{(E, R)}$	if $t =_{AC} t'$
(2) Reduce	$\dfrac{(E \cup \{t' = t\}, R, S)}{(E \cup \{t' = u\}, R, S)}$	if $t \rightarrow_{R.AC} u$
(3) Orient	$\dfrac{(E \cup \{t = t'\}, R, S)}{(E, R \cup \{t \rightarrow t'\}, S)}$	if $t >_t t'$ and $t \rightarrow t'$ is sort-decreasing.
(4) Deduce	$\dfrac{(E, R, S)}{(E \cup \{t_1 = t_2\}, R, S)}$	if $t_1 = t_2 \in CP_{AC}(R)$
(5) Extend	$\dfrac{(E, R, S)}{(E, R, S \cup \{l \rightarrow r\})}$	if $l = r \in EXT_{AC}(R)$
(6) Simplify Right	$\dfrac{(E, R \cup \{l \rightarrow r\}, S)}{(E, R \cup \{l \rightarrow r'\}, S)}$	if $r \rightarrow_{R.AC} r'$
	$\dfrac{(E, R, S \cup \{l \rightarrow r\})}{(E, R, S \cup \{l \rightarrow r'\})}$	if $r \rightarrow_{R.AC} r'$
(7) Simplify Left	$\dfrac{(E, R \cup \{l \rightarrow r\}, S)}{(E \cup \{l' = r\}, R, S)}$	if $l \rightarrow_{R.AC} l'$ by $g \rightarrow d$ where $l \overset{e}{\triangleright} g$.

Figure 2.6: Rules for Standard Order-sorted AC Completion.

This is basis of the completion procedure for order-sorted AC rewriting, where all critical pairs are generated in turn and tested to see if they are trivial. This is given in Figure 2.6 as a set of inference rules on triples consisting of a set of equations and two sets of rules. The rules can be applied in any order to give an abstract completion algorithm. Note that $\overset{e}{\triangleright}$ is the well-founded strict encompassment ordering: $t_1 \overset{e}{\triangleright} t_2$ if $\exists \sigma$ and path $p \in O(t_1)$ such that $t_1|_p \simeq \sigma t_2$, and σ is not a variable renaming.

Three other conditions have to be met for this to generate a confluent set of rewrite rules. Firstly, the rules must be terminating. This is ensured by supplying a termination ordering $>_t$ on terms, and proving that each rule entered into R is within this ordering, generating a termination proof as completion proceeds. Secondly, as we are using order-sorted

signatures, we must show that each rule is sort-decreasing. Finally, we have to establish the local coherence of rewriting modulo the AC theory. This is established by means of AC-extensions.

Definition 2.80. Given a rewrite rule $l \rightarrow r$ within the associative-commutative theory AC, then if $(x + y) + x = x + (y + z) \in AC$, for some AC operator $+$, and $x + y$ order-sorted AC-unifies with l, then $l + z \rightarrow r + z$ is an *extended rule.* The extension is AC-trivial if there is a rule $g \rightarrow d \in R$ and substitution σ such that $l + z =_{AC} \sigma g$ and $r + z \xrightarrow{*}_{R.AC} \sigma d$. The set of all non-AC-trivial AC-Extensions of R is written $EXT_{AC}(R)$.

In an order-sorted AC system, there may be axioms $(x + y) + x = x + (y + z) \in AC$, for different sorts, and each may generate extensions, differing from unsorted AC-completion. Peterson and Stickel [PS81] and Bachmair [Bac91] prove that this set of extended rules added to the set of rewrite rules is sufficient to prove local coherence of AC rewriting.

Completion starts with the set axioms and the empty set of rules and extensions $(E, \emptyset, \emptyset)$, and by the application of the above rules, a sequence of Equation set/Rule set/Extension set triples are generated, $\{(E_i, R_i, S_i)\}_{i=1,2,\ldots}$. Extensions are kept as a separate set of rules as they have to be protected from being rewritten by the rule which they extend. The final triple is written $(E_\infty, R_\infty, S_\infty)$. This process can result in one of three states.

1. **Success.** All critical pairs are generated and oriented, and no more non-trivial critical pairs remain. The rules terminate in the state $(\emptyset, R_\infty, S_\infty)$. The resulting set of rules $R_\infty \cup S_\infty$ form a decision procedure for equations $t_1 = t_2$; if $t_1 \xrightarrow{*}_{R_\infty \cup S_\infty} t_1{\downarrow} =_{AC} t_2{\downarrow}_{R_\infty \cup S_\infty} \xleftarrow{*} t_2$ then the equality holds; otherwise it does not hold.

2. **Failure.** There is an (irreducible) equation $t_1 = t_2 \in E_i$ for some i, which is unorientable, as $t_1 \not>_t t_2$ and $t_2 \not>_t t_1$, in the termination ordering, or $t_1 >_t t_2$ but $t_1 \rightarrow_t t_2$ is not sort-decreasing. The rules terminate in a failure, with $E_i \neq \emptyset$, and no decision procedure results. The user can attempt to provide a more suitable termination ordering, or no such ordering may exist.

3. **Non-termination.** There is no i such that $E_i = \emptyset$ and R_i has no non-trivial critical pairs, so completion does not terminate and the set $R_\infty \cup S_\infty$ is infinite. In this case,

there is a semi-decision procedure for $t_1 = t_2$ under the algebra; if the equality holds, there is some i such that $t_1 \xrightarrow{*}_{R_i \cup S_i} t_1 \downarrow =_{AC} t_2 \downarrow \xleftarrow{*}_{R_i \cup S_i} t_2$.

2.8 Implementing Order-Sorted Equational Reasoning

The non-overloaded order-sorted rewriting theory has been implemented within MERILL[3], a general purpose order-sorted equational reasoning system, developed at the C.L.R.C. Rutherford Appleton Laboratory (RAL) and the Dept of Computing Science at Glasgow University[4]. In this section we give a brief overview of this implementation. A more detailed description of the use of MERILL appears in [Mat93b]; a brief description appears in [Mat93a].

The development of MERILL was inspired by the ERIL (*Equational Reasoning : an Interactive Laboratory*) system developed by Jeremy Dick at RAL and Imperial College, London [Dic85, Dic87], which used order-sorted logic in a practical equational theorem prover. However, the design of ERIL became obsolete: it was considered too slow, and lacked AC operators. It was decided to implement a new system which would retain the major features of ERIL, and have a similar 'menu-driven' user interface whilst being significantly faster and having new facilities. Like its predecessor, MERILL is designed to support order-sorted reasoning. However, unlike ERIL, it also incorporates reasoning modulo commutative and associative-commutative equations. It is also comparable with the Helios-OBJ [Gna92a] system, which supports AC-rewriting, and order-sorted completion, but not order-sorted completion modulo equations; MERILL remains unique in that respect.

MERILL is written in Standard ML [HMT88, Rea89, Pau96], chosen due to the elegance associated with functional programming languages, coupled with the availability of efficient implementations. Standard ML's module system also makes the system easy to modify to incorporate new extensions and experiments in equational reasoning techniques.

[3] Available via anonymous ftp from the University of Glasgow ftp.dcs.glasgow.ac.uk (130.209.240.50)

[4] Partly supported by the SERC/DTI IEATP project "Verification Techniques for LOTOS Specifications" between RAL, Glasgow University and Royal Holloway and Bedford New College.

Part of the philosophy of MERILL is that the user is in control. Thus the system has few built in assumptions and does little reasoning in the background. This also means that the user has to enter the object syntax explicitly as the system has no built in assumptions, for example about the form of variables.

2.8.1 The Organisation of the MERILL System

The organisation of MERILL is given in Figure 2.7. The central database of the system is divided into three major components: the signature, the equality sets and the environment. These mutually dependent data sets combine to provide the raw material for the tools, and provide storage for their results. Tools include rewriting, unification and completion.

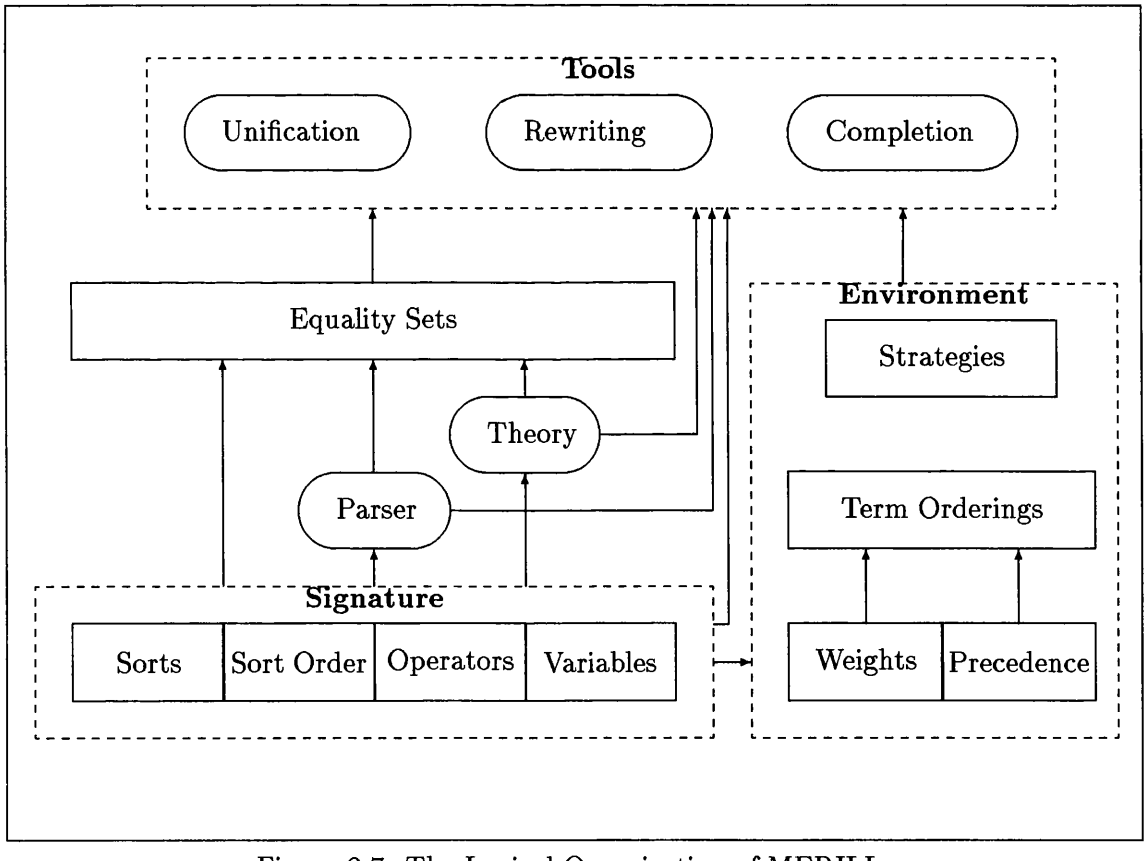


Figure 2.7: The Logical Organisation of MERILL.

We discuss the features of MERILL with reference to the specification in Figure 2.8, of basic arithmetic over the natural numbers. This example shows the distinctive features of order-sorted equational logic as implemented in MERILL. It has a hierarchy of sorts

and subsorts, multiple ranks for functions and sorted variables. Further, the addition and multiplication functions are both declared to be AC.

Sorts:	$Bool$, nat, $zero$, $posnat$	
Subsorts:	$zero < nat$, $posnat < nat$	
Operators:	$ff : \rightarrow Bool$	$tt : \rightarrow Bool$
	$0 : \rightarrow zero$	$succ : nat \rightarrow posnat$
	$\_ > \_ : nat\ nat \rightarrow Bool$	$pred : posnat \rightarrow nat$
	$\_ + \_ : nat\ nat \rightarrow nat$	$\_ * \_ : nat\ nat \rightarrow nat$
	$\_ + \_ : zero\ zero \rightarrow zero$	$\_ * \_ : posnat\ posnat \rightarrow posnat$
	$\_ + \_ : nat\ posnat \rightarrow posnat$	$\_ * \_ : zero\ nat \rightarrow zero$
	$\_ + \_ : posnat\ nat \rightarrow posnat$	$\_ * \_ : nat\ zero \rightarrow zero$
Variables:	$n, n1, n2 : nat$, $p : posnat$	
Equations:	$pred(succ(n)) = n$	$n1 + n2 = n2 + n1$
	$n + 0 = n$	$(n + n1) + n2 = n + (n1 + n2)$
	$n + (succ(n1)) = succ(n + n1)$	$n1 * n2 = n2 * n1$
	$n * 0 = 0$	$(n * n1) * n2 = n * (n1 * n2)$
	$(succ(n)) * n1 = (n * n1) + n1$	
	$n * (n1 + n2) = (n * n1) + (n * n2)$	
	$0 > n = ff$	$p > 0 = tt$
	$succ(n) > succ(n1) = n > n1$	

Figure 2.8: An Example Specification of the Naturals.

2.8.2 Signatures in MERILL

The signature defines the object language the user wishes to use. This comprises four components: the sorts; a sort ordering over the declared sorts; operators together with their ranks; and sorted variables. Unlike other term-rewriting systems, which have predefined conventions about variables and operator symbols, MERILL insists that the user declares all the sort, operator and variable symbols before use.

Sorts and Sort-Orderings. Sorts are declared as names. The user then declares a set of pairs of sorts as subsort declarations, and the subsort relation is calculated as the least transitive reflexive closure containing the declarations. This ordering is stored in an efficient manner; no repeated or reflexive pairs are stored and no pair is stored which could be deduced using the transitive closure of the relation. Circularities are also prevented.

There are also two predefined sorts available, `Top` and `Bottom`, the first of which is larger than any other sort, the second smaller.

Operators. Operators are declared in a mixfix format, that is they can have a concrete syntax of a sequence of separate symbols with the arguments interspersed between them; the positions of arguments denoted by underscores in declarations. Also, the rank of the operator is declared. Examples include:

```
_ + _ :  int int -> int              (* infix, binary operator + *)
_ + _ :  nat nat -> nat              (* overloaded *)
a :  int                             (* constant *)
b :  -> int                          (* constant *)
f :  int int -> int                  (* function of form f( _ , _ ) *)
f(_) :  int -> int                   (* due to arity this is a different operator *)
if _ then _ else _ :  bool s s -> s  (* a mixfix operator *)
_ _ :  s s -> s                      (* Juxtaposition as an operator *)
```

The user can also declare that operators are associative, commutative, or both, by declaring the attributes `COMM ASSOC` after the operator declaration. Thus we may have:

```
_ + _ :  nat nat -> nat (COMM ASSOC)   (* + is AC *)
_ * _ :  matrix matrix -> matrix       (ASSOC) (* matrix mult assoc only *)
```

Note that currently a declaration will apply over the whole of the sort structure, wherever the terms are well-defined. MERILL automatically checks the compatibility of the declaration, using the algorithm given in Corollary 2.55, and will delete the declaration if it is incompatible with the signature, by generating a representation of this equational theory which is also used in completion to construct associative extensions.

Figure 2.9 gives an example screen from MERILL showing operators being added as in the example in Figure 2.8, showing a typical example of the menu-driven style of the system's interface. The top of the screen shows the operators already within the system with their ranks. The user has selected the '`a`' item from the menu, to add new operators. Each rank is added separately; here we declare the greater-than operator, and two ranks of the multiplication operator. The addition operator is annotated '`(ASSOC COMM)`' to denoting

```
-----------------------------------OPERATORS-------------------------------------
 1       0 :  -> zero
 2       succ( _ ) :  nat -> posnat
 3       pred( _ ) :  posnat -> nat
 4       _ + _ :  nat nat -> nat (ASSOC COMM)
                :  zero zero -> zero (ASSOC COMM)
                :  nat posnat -> posnat (ASSOC COMM)
                :  posnat nat -> posnat (ASSOC COMM)
 5       tt :  -> Bool
 6       ff :  -> Bool
-------------------------(h - help, Control-C - Interrupt)-----------------------
                                                Operator Options
                                                 a   Add Operators
                                                 d   Delete Operators
                                                 e   Equational Theory
                                                >>   a
Enter Operators:
>> _ > _ :  nat nat -> Bool
>> _ * _ :  nat nat -> nat
>> _ * _ :  posnat posnat -> posnat
```

Figure 2.9: Adding operators to MERILL.

that it has been has been declared to be AC.

Variables. Classes of variable names are similarly declared with their sort within the variables menu. There are two types of variable name declarations which can be entered, *whole variables* and *variable prefixes.* The first are variable names in their own right. The second form a class of variable names, each with the declared prefix. Example variable declarations include:

```
x* :  int     (* all strings beginning with an x are sort int vars *)
fre* :  S     (* all strings beginning with a fre are sort S vars *)
fred :  int   (* but fred is interpreted as a int var *)
joe :  S      (* and joe is a sort S var *)
```

In the above example '`freda`' would be a sort `S` variable, while '`fred`' is of sort `int`. Variable names can be the same as sort names and can overlap with operator forms, but operator forms take precedence.

From the declarations of operators and variables, the system builds a parser of terms, handling the mixfix forms to try to produce an unambiguous parse, using a method similar

to that in [Voi86]. As it is implemented, the parser chooses one particular parse, which may not be the desired parse; this could be easily modified to allow the user to choose between a list of possible parses. Brackets can always be used to disambiguate terms.

Tests on the Signature. Three tests are available to check the properties of the given signature. Unlike the sort-compatibility test mentioned earlier, these are optionally called by the user.

Inhabitedness. The inhabitedness test determines whether all the declared sorts have ground terms, either declared within them or inferred from the current signature, as in Definition 2.31, and displays which sorts are inhabited and which are not.

Monotonicity. The monotonicity test checks operator declarations to see if any pair breaks the monotonicity requirement, as given in Definition 2.39.

Regularity. The Regularity test determines whether the current signature is regular and displays those pairs of operator arities which break the condition defined in Definition 2.14. This is done pairwise. A finite monotonic signature is Regular if for all operators f which have ranks $w_1 \rightarrow s_1$ and $w_2 \rightarrow s_2$ and if there is some w_0 such that $w_0 \leq w_1$ and also $w_0 \leq w_2$, then there is some $w \leq w_1, w_2$ such that f has rank $w \rightarrow s$ and $w_0 \leq w$. This definition extends to a non-monotonic signature by adding the extra condition that if there is a rank $w' \rightarrow s'$ such that $w_0 \leq w' \leq w_1, w_2$, then $s \leq s'$.

2.8.3 Equality Sets

The user can declare equalities which are collected within *equality sets*. This allows separation of equalities which can be manipulated separately. There are four varieties of equality which can be declared within MERILL, each with a different role.

Equality type	Format of Equality	Role
Equations	$t_1 = t_2$	Axioms which cannot be used for rewriting.
Rewrite Rules	$t_1 => t_2$	Rules used for rewriting.
Conjectures	$t_1 =? = t_2$	Passive equalities used as a query, declared true when rewritten to identity.
Conditionals	$e_1, \ldots, e_n ==> e$	Used for rewriting, but not completion.

By dividing equalities into sets the interaction between them can be controlled. For example, an equality can be rewritten by a specified set of rules, or critical pairs can be generated between two sets of rules and the resulting equations placed in a third. This allows the user to control the reasoning process more tightly than keeping all equalities together.

2.8.4 The MERILL Environment

The environment allows the selection of term orderings and completion strategies which are independent of the nature of the signature used. To use these orderings the user may have to set precedences and weights on operators.

Global Orderings. The global term ordering is a noetherian ordering on terms, generic on operators, which controls the ordering of rewrite rules during completion. Several global term orderings have been implemented.

User RPO - Left Status. A version of the Recursive Path Ordering [Der87] which uses the precedences on operators given previously by the user and all operators are of left lexicographic status.

User RPO - Right Status. A version of the Recursive Path Ordering which uses the precedences on operators given previously by the user and all operators are of right lexicographic status.

User RPO - Multiset Status. A version of the Recursive Path Ordering which uses the precedences on operators given previously by the user and all operators are of multiset lexicographic status.

User KBO. A version of the Knuth-Bendix Ordering [KB70], where all operators have their weights and precedences predefined by the user.

User AKBO. A version of the Knuth-Bendix Ordering by Steinbach [Ste90], where all operators have their weights and precedences predefined, which in addition can handle AC-operators. This method is subject to three conditions.

1. C and AC-Operators have a multiset status, others left-lexicographic.
2. The weight of all AC-operators is zero.
3. AC-Operators are minimal in the precedence ordering.

Automated KBO. The Incremental Knuth-Bendix Ordering as described by Dick, Kalmus and Martin [DKM90], which automatically decides whether a Knuth-Bendix ordering exists for the current set of rules as they are oriented. If a rule can be oriented in either direction, then the local ordering is used; if the manual local ordering is used, the user has control. The implementation of this ordering is due to Cropper at Glasgow University [Cro92]. It can handle small sets of rules well, but care must be taken as in certain cases the matrices used internally to calculate the existence of an ordering can grow exponentially.

In addition, by not selecting any of the above, the local ordering will be used for ordering rules. This is particularly useful for manual ordering of rules in completion when no ordering is applicable or easy to set up, although there is no proof of termination.

Local Ordering. The Local Ordering is invoked when the Global Term ordering has insufficient information to determine the orientation of an equation, which can be in either direction. Note that this is not the same as *unorientability*, where the Global Ordering cannot order the equation in either direction. The available Local Orderings are:

by-size Puts the term with the largest number of operators on the left-hand side.

manual Prompts the user to orient the rule.

2.8.5 Tools in MERILL

MERILL allows the explicit unification of terms. It uses the algorithm of Clausen and Fortenbacher [CF89] for solving sets of linear Diophantine equations, and the algorithm of Lincoln and Christian [LC89] for generating AC-Unifiers, modified as described in Section 2.6 for order-sorted signatures. This algorithm assumes that checks for regularity and sort-compatibility have already been performed.

Other tools available for operating on terms and equalities include: rewriting of both terms and equations, as described in Section 2.7; the generation of critical pairs, and three order-sorted completion algorithms.

Knuth-Bendix Completion. An implementation of the Knuth-Bendix completion algorithm [KB70] adapted to order-sorted signatures is available. This does not use any associative-commutative declarations of operators. The user can also attempt to use the Knuth-Bendix algorithm as a semi-decision procedure. Any conjecture equalities can be reduced in parallel with the completion process: they are proven when reduced to an identity.

Huet's Completion Algorithm. This is an implementation of Huet's Left-linear completion algorithm [Hue81] adapted to order-sorted signatures. It handles associative-commutative operators without using special unification algorithms, but is restricted to left-linear rewrite rules only, a left-linear rule being one with no repeated variables in its left-hand term. Huet's completion algorithm is similar to the Knuth-Bendix completion algorithm, with the exception that it will generate an error and halt if a non left-linear rule is generated.

Peterson and Stickel's Completion Algorithm. This is a combined implementation of Peterson and Stickel's commutative and associative-commutative completion algorithms [PS81], as sketched in Section 2.7. This generates critical pairs by using AC unification and also generates extended rules for the associative axiom. As the AC-unification algorithm is computationally very expensive, the maximum amount of orientation and interreduction of rules are performed in this algorithm before new critical pairs are generated.

All three of these completion algorithms can result in the three outcomes as in Section 2.7.

Local strategies determine the order in which rules are chosen for consideration in the Knuth-Bendix Algorithm. A fair strategy is one where all possible critical pairs are generated and considered for orientation. Currently available strategies are:

by-size Picks equations according to the number of operators occurring in both sides, the smallest first; a fair strategy.

by-age Picks equations according to the length of time that they have been in the system, the oldest first; a fair strategy.

manual Prompts the user to select which equation to consider next; an unfair strategy.

The strategy used should be 'fair', especially when the algorithm is being used as a semi-decision procedure to some set of conjectures. However, the unfair manual strategy can be useful if the user knows which equation to orient next.

To give an example of the nature of these completion algorithms, we complete the example in Figure 2.8 using AC-completion. The organisation of the algorithm is given in Figure 2.10. Boxes represent equality sets, solid arrows movement of equalities during completion, and dashed arrows rewriting of equalities in one set by another set. Thus the algorithm is data driven, in a similar style to ERIL and also to ORME [Les90]; the control of the algorithm is determined by the presence of equalities in these sets. We select three sets from the equality sets database: A contains the unoriented equations; T is a set of temporary rewrite rules; and R is the final target set to hold the rewrite rules. In addition, another set H can hold conjectures to be considered in parallel with the completion process.

The strategy described in this diagram is to move all axioms from A, orienting them through the term ordering O and placing them in T, also checking whether the equation is sort-decreasing. All equality sets are rewritten by T and R, (rewriting by T and R on themselves omitted in the diagram for clarity). Then, using the selection strategy from the environment, a single rule is moved from T to R, generating all critical pairs between that rule and members of R, including itself. Non-trivial critical pairs are then placed in A, and the cycle begins again. Termination occurs when the sets A and T are empty; the set R then contains a confluent set.

To run this we first need to set up a term ordering. Only one of the implemented orderings

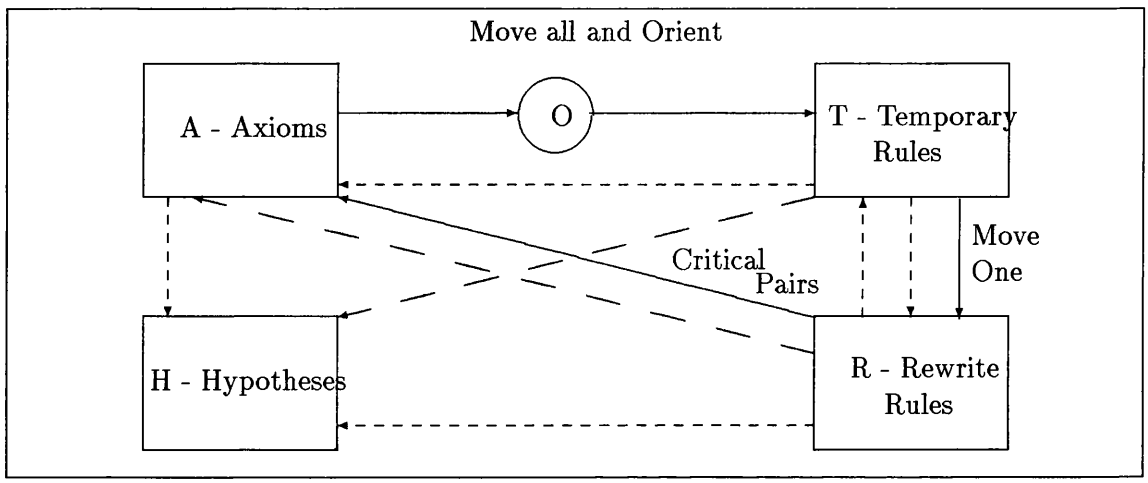


Figure 2.10: Running AC-Completion.

accepts AC-operators; the Associative Knuth-Bendix ordering. However, the rule set given is not suitable for this ordering and we use a manual ordering, and assume termination.

```
-----------------------------------Rewrite Rules-----------------------------------
10 Equalities
 1   pred(succ(n)) => n
 2   n + 0 => n
 3   n * 0 => 0
 4   0 > n => ff
 5   p > 0 => tt
 6   n + (succ(n1)) => succ( n + n1 )
 7   (succ(n)) > (succ(n1)) => n > n1
 8   (succ(n))* n1 => ( n * n1 )+ n1
 9   n *( n1 + n2 ) => ( n * n1 )+( n * n2 )
 10  ((succ(n))* n1 )* n1 => ( n1 * n1 )+( n1 *( n * n1 )) (*)
-------------------------(h - help, Control-C - Interrupt)-------------------------
                                          Equality Set Options
                                           a   Add Equations
                                           d   Delete Equations
                                           o   Orient Equations
                                          >>
```

Figure 2.11: Complete Set of Rewrite Rules for Arithmetic on Naturals.

Figure 2.11 gives the results of running this strategy upon the example in Figure 2.8 from an actual MERILL session. In this case no new non-trivial pairs were created. However, rule 10 is a new extended rule, and is marked with an asterisk, added to ensure confluence of AC-rewriting. This rule is generated by superposing rules on the associative laws of the equational theory generated by the annotations on operators.

Chapter 3

The Problems with the Standard Semantics

In this chapter, we discuss of the problems associated with the standard theory of order-sorted specification, and consider several approaches given in the literature to their mitigation. It is these shortcomings, and those of the proposed solutions which motivates the theoretical work described in the rest of this thesis.

3.1 The Problems of these Semantics

The approach to order-sorted specification based on syntactic sorts and well-sorted term algebra, given in the previous chapter, has several problems which have already been hinted at in Example 2.66, and also discussed in [CH91b], which note following defects of order-sorted equational logic.

1. The congruence relation induced from equational axioms may not be sound.
2. Replacement of equals for equals may not be equivalent to equational deduction.
3. The Church-Rosser property is not equivalent to Confluence for the rewrite relation.

4. The Critical Pair Lemma no longer holds.

The reason Chen and Hsiang give for these problems is worth quoting (*op. cit. p.3*):

> *When establishing the equivalence of two terms through equational replacements, the substitution may force the introduction of terms which are not defined in the term algebra from the sort declaration. These 'ill-formed' terms should have semantic meaning since they are equivalent to well-defined terms. Furthermore, two terms whose equivalence can be established should belong to the same sorts since they apparently have the same meaning.*

To expand on this, consider the following example.

Example 3.1. ([CH91b]) Given the specification $\mathcal{S}$ as follows

Sorts:	$B\ C\ A$	
Subsorts:	$B \leq A, C \leq A$	
Operators:	$a :\rightarrow B$	$c :\rightarrow C$
	$b :\rightarrow B$	$f : B \rightarrow A$
Equations:	$a = c$	$b = c$

The ground quotient term algebra $\mathcal{T}_{\mathcal{S},\emptyset}$, has the following sets denoting sorts.

$$\begin{aligned} B^{\mathcal{T}_{\mathcal{S},\emptyset}} &= \{[a],[b]\} \\ &= \{\{a,b,c\}\} \\ C^{\mathcal{T}_{\mathcal{S},\emptyset}} &= \{[c]\} \\ &= \{\{a,b,c\}\} \\ A^{\mathcal{T}_{\mathcal{S},\emptyset}} &= \{[a],[b],[c],[f(a)],[f(b)]\} \\ &= \{\{a,b,c\},\{f(a),f(b)\}\} \end{aligned}$$

Whilst this is the correct initial interpretation with respect to the unsorted algebra, it has consequences that the user may not have expected. In particular, there are three points which raise questions about the soundness of this method.

The first point is that $a = c$ but $f(c) \neq_E f(a)$ which illustrates that since $f(c)$ is an ill-formed term, the congruence rule, while sound, does not give results as expected.

The second point uses the notion of equational replacement. In this example, whilst $f(a) =_E f(b)$ by the deduction rules, it is not the case that $f(a) \longleftrightarrow^*_E f(b)$ as this would require the use of $f(c)$ in an intermediate step. This illustrates that even expected consequences involving well-formed terms may not be deducible operationally as they may involve steps that use ill-formed terms.

The third point is if we consider the equations as left to right rewrite rules, the system is confluent, but there is no rewriting proof of $f(a) \longleftrightarrow^*_E f(b)$. □

Other problems with this methodology occur when equations are used as rewrite rules to do automated deduction. The completeness of rewriting is lost and the critical pair lemma no longer applies unless the restriction applies that the rewrite rules used are *sort-decreasing.* Exceptions to this restriction are common and natural, as in the following example.

Example 3.2. Consider the following (partial) specification (again from [CH91b]):

Sorts:	Int, Nat
Subsorts:	$Nat \leq Int$
Operators:	$\_*\_ : Nat\ Nat \rightarrow Nat$
	$\_*\_ : Int\ Int \rightarrow Int$
	$square : Int \rightarrow Nat$
Variables:	$x : Int$, $n : Nat$
Equations:	*(equations for* $\_*\_$ *omitted)*
	$square(x) = x * x$

The natural way to orient this equation is from left to right as $square(x) \rightarrow x*x$. However, this is a sort increasing rule and is thus disallowed. □

In this case, sort-increasing rules could be avoided, for example by the use of an auxiliary operator $abs : Int \rightarrow Nat$ for the absolute value of an integer, but such a specification is not as clear and succinct. Even if an initial set of rules is sort-decreasing, Knuth-Bendix

Completion may generate rules which are not, causing the completion process to fail.

If we allow such rules in specifications, problems occur as in Example 2.66. Also, consider the following specification from [Wer93], similar to the above.

Example 3.3. Consider the following specification:

Sorts:	Int, Nat	
Subsorts:	$Nat \leq Int$	
Operators:	$0 : \rightarrow Nat$	$succ : Nat \rightarrow Nat$
	$square : Int \rightarrow Nat$	$succ : Int \rightarrow Int$
	$\_*\_ : Int\ Int \rightarrow Int$	$\lvert\_\rvert : Int \rightarrow Nat$
Variables:	$x : Int$, $n : Nat$	
Rules:	$\lvert n\rvert \rightarrow n$	$square(x) \rightarrow x * x$

This specification has no critical pairs, but there is the following overlap at a variable path:

$$succ(x * x) \leftarrow succ(square(x)) \leftarrow |succ(square(x))| \rightarrow |succ(x * x)|$$

which is not convergent. □

While we emphasise that the deduction rules are sound and complete with respect to the given definition of algebra, the standard proof and model theory is not strong enough to capture the *desirable* consequences of specifications. This criticism applies to both the non-overloaded and the overloaded semantics. There have been several attempts to overcome these problems, which shall be considered in the rest of this chapter.

3.2 Syntactic Alterations

A style of approach to the problems associated with the standard order-sorted theory is to modify the signatures of problematic specifications.

Adding a Top Sort. One approach is to add a new top sort and modify the signature accordingly [SS89, SNGM88].

Definition 3.4. Given an Order-Sorted Signature Σ, the $\top$-*Augmented signature* $\Sigma^{\top}$ is given by:

1. $S_{\Sigma^{\top}} = S_{\Sigma} \cup \{\top\}$
2. $\leq_{\Sigma^{\top}} = \leq_{\Sigma} \cup \{(s, \top) | s \in S_{\Sigma^{\top}}\}$
3. $\forall f \in \mathcal{F}_{\Sigma} \cdot rank_{\Sigma^{\top}}(f) = rank_{\Sigma}(f) \cup \{(\top^{\alpha(f)}, \top)\}$

In $\top$-Augmented signatures all terms are well-formed, thus equivalence is sound with respect to the logic; there are no ill-formed terms through which to make deductions. However, the restriction to sort-decreasing rules remains. Also this style reintroduces the problem which order-sorted specification was designed to avoid: any ill-defined term can be used as it now is well-defined. This method does not discriminate between those terms with well-defined *denotations* and those without.

Adding Extra Operators. Gnaedig, Kirchner, and Kirchner [GKK90] propose another solution to the problem of non-sort-decreasing rules. When such a rule is encountered, a new sort and a new operator are added to the signature and the rule is split into new sort-decreasing rules.

Example 3.5. Given the following specification:

Sorts:	s, s', s''	
Subsorts:	$s' < s''$, $s < s''$	
Operators:	$a : \rightarrow s''$	$g : s'' \rightarrow s'$
	$f : s'' \rightarrow s$	$h : s'' \rightarrow s''$
Variables:	$x : s''$	
Rules:	$h(x) \rightarrow f(x)$	$h(x) \rightarrow g(x)$

Completion generates the critical pair $f(x) = g(x)$ which has terms of incomparable sorts. As all sorts are non-empty, there is an inhabited intersection of s and s', and $f(a)$ and $g(a)$ must be members of this intersection by the critical pair. A new sort $s \frown s'$ and a new operator k are thus added which conservatively extend the specification.

Sorts:	$s,\ s',\ s'',\ s^\frown s'$	
Subsorts:	$s^\frown s' < s' < s'',\ s^\frown s' < s < s''$	
Operators:	$a : \rightarrow s''$	$g : s'' \rightarrow s'$
	$f : s'' \rightarrow s$	$h : s'' \rightarrow s''$
	$k : s'' \rightarrow s^\frown s'$	
Variables:	$x : s''$	
Rules:	$h(x) \rightarrow k(x)$	$f(x) \rightarrow k(x)$
	$g(x) \rightarrow k(x)$	

□

Gnaedig, Kirchner, and Kirchner give a theorem which can be used to alter the specification when a pair $p = q$ has some specialisations ρ, $LS(\rho(p)) \leq LS(\rho(q))$ which can be oriented and for other specialisations ρ, $LS(\rho(p)) \bowtie LS(\rho(q))$. This theorem is stated incorrectly in [GKK90] as it does not change the sort ordering; we give a corrected version.

Theorem 3.6. Assume that the sort set does not contain a void sort and that completion generates a non-sort-decreasing pair $\forall X.p = q$, such that $X = \vec{x} \cup \vec{y} \cup \vec{z}$, where $\vec{x} = Var(p) \cap Var(q)$, and $\vec{y} = Var(p) - \vec{x}$, $\vec{z} = Var(q) - \vec{x}$. Let $S_0 = \{s_\rho | \rho \text{ is such that } LS(\rho(p)) \bowtie LS(\rho(q))\}$ be a new set of sort symbols, $\leq_0 = \{s_\rho \leq LS(\rho(p)), s_\rho \leq LS(\rho(q)) | s_\rho \in S_0\}$ be a set of new sort order declarations, and $\Sigma_0 = \{f_\rho : \sigma(\rho(\vec{x})) \rightarrow s_\rho | s_\rho \in S_0 \text{ and } f_\rho \notin \Sigma\}$ be a new set of operator declarations. Then let:

$$
\begin{aligned}
S' &= S \cup S_0 \\
\leq' &= \leq \cup \leq_0 \\
\Sigma' &= \Sigma \cup \Sigma_0 \\
\Gamma' &= \Gamma \cup \{\forall \rho(\vec{x}).\rho(p) = f_\rho(\rho(\vec{x})) | f_\rho \in \Sigma_0\} \cup \{\forall \rho(\vec{x}).\rho(q) = f_\rho(\rho(\vec{x})) | f_\rho \in \Sigma_0\}
\end{aligned}
$$

where Γ is the set of rules, pairs and axioms generated so far. Then:

- $(S', \leq', \Sigma')$ is coherent
- Γ' is a sort-decreasing equational term-rewriting system, and
- $\mathcal{T}_{\Sigma',\Gamma'}(X)$ is a conservative extension of $\mathcal{T}_{\Sigma,\Gamma}(X)$.

This method has the advantages of being simple, and straightforward to implement. However, it does obscure the signature with intersection sorts and extra operators. Moreover, it is of limited applicability. In Example 3.2 this method results in the new equations:

$$\begin{aligned} square(x) &= sq(x) \\ x * x &= sq(x) \end{aligned}$$

for some new operator $sq : Int \rightarrow Nat$ and nothing is gained.

Retracts. The method of retracts is described in [GJM85, GM89], and vigorously advocated in [GD92]. This method allows a restricted class of ill-sorted terms to appear in rewriting and is implemented in the rewriting system and programming language OBJ-3 [KKM88, GW88]. Here, an order-sorted specification $(\Sigma, \mathcal{E})$ is conservatively extended into $(\Sigma^{\otimes}, \mathcal{E}^{\otimes})$ as follows:

1. $\forall s, s' \in \Sigma$ such that $s \leq s'$, add the *retracts* $r_{s,s'} : s' \rightarrow s$ to $\mathcal{F}_{\Sigma}$.
2. For each retract add to $\mathcal{E}$ the retract equation: $r_{s,s'}(x : s) = x$

By using retract operators, ill-formed terms can be transformed into well-formed ones, although marked by the retracts as being 'odd'. Then, by the application of the retract equations with the rules the term may be rewritten into a well-formed term. For example, in the specification of sequences of Example 2.28, the term $head(tail(1@2))$ is ill-formed. The use of retracts can convert this into $head(r_{NeSeq,Seq}(tail(1@2)))$ and can be rewritten:

$$head(r_{NeSeq,Seq}(tail(1@2))) \rightarrow head(r_{NeSeq,Seq}(2)) \rightarrow head(2) \rightarrow 2$$

OBJ-3 automatically inserts retracts at parse time. In [GD92] examples are given which give equations on the retracts to give further properties between sorts, such as converting between alternative representations of an abstract datatype.

Retracts are undoubtedly a useful operational tool to allow the efficient use of ill-sorted terms in rewriting. However, the method of retracts has not been used in completion theory. This would be an interesting research area. For example, one could envisage a rule which takes a sort-increasing rule $l \rightarrow r$ and replaces it with a sort-preserving rule $l \rightarrow I(r)$

where $I = r_{s_0,s_1} \circ r_{s_1,s_2} \circ \cdots \circ r_{s_{n-1},s_n}$, $s_0 = \mathcal{LS}(l)$, $s_n = \mathcal{LS}(r)$ and $s_i \leq s_{i+1}$ for some path $s_0 \ldots s_n$ through the sort lattice. This is similar to the work of Ganzinger discussed below.

3.3 Using a Many-Sorted Signature

Ganzinger [Gan89, Gan91b] presents a different approach by not considering completion over an order-sorted signature, but by converting the signature into an equivalent many-sorted signature. To do this he defines a translation function between signatures.

Definition 3.7. Let Σ^{OS} be an order-sorted signature. Its *translation* into a many-sorted signature $\Sigma = (S, \Omega)$ is defined as follows:

1. The sorts in S are the sorts in S^{OS}.
2. If $f : s_1 \ldots s_n \rightarrow s_0$ is an operator in Σ^{OS}, then $f_{s_1 \ldots s_n \rightarrow s_0} : s_1 \ldots s_n \rightarrow s_0 \in \Omega$.
3. If $s \leq_b s'$ in Σ^{OS}, then $i_{s \subset s'} \in \Omega$.

where $s \leq_b s'$ if $s \leq s'$ and $\nexists s''$ such that $s \leq s'' \leq s'$ and $i_{s \subset s'} : s \rightarrow s'$ is the injection from s into s'.

The injection functions represent the subsort relation in the many-sorted algebra, and functions are separated into a different function for each rank. Using this translation, there is a pair of mappings from $T_{\Sigma^{OS}}(X)$ to $T_{\Sigma}(X)$ and vice-versa. These mappings are not inverses; more than one many-sorted terms map to an order-sorted term, and the function $\lambda : T_{\Sigma^{OS}}(X) \rightarrow T_{\Sigma}(X)$, produces the term with the lowest possible sort in an arbitrary total extension of $\leq_{\Sigma^{OS}}$. Ganziger defines a canonical rewrite system LP on Σ to convert a Σ-term to the Σ-term with the 'lowest parse' of the order-sorted term it represents.

Using this translation, an order-sorted equational presentation E is converted into a many-sorted presentation $E^{\#}$ which has the same equational theory.

$$\lambda(t_1 = t_2) = I_{s_1 \subset s}(\lambda(t_1)) = I_{s_2 \subset s}(\lambda(t_2))$$

where $I_{s_i \subset s}$ is the minimum composition of injection functions from s_i to s and s is some

minimal supersort of the order-sorted terms t_1 and t_2. This function is naturally extended to sets of (conditional) equations.

$$E^{\#} = LP \cup IN \cup \lambda(E) \text{ where}$$
$$IN = \{i_{s \subset s'}(x) = i_{s \subset s'}(y) \Rightarrow x = y | i_{s \subset s'} \in \Omega\}$$

In the rewriting theory of this system, rewriting by a set of order-sorted sort-decreasing rules R is equivalent to rewriting by the many-sorted rules $R_{\#} = \lambda(R_S) \cup LP$ where R_S is the set of rules generated by adding all specialisations of R, renaming the variables of the rules by variables of lower sorts. Thus the many-sorted completion algorithm can be used.

However, a many-sorted conditional rule is used to simulate a non-sort-decreasing rule. Thus in Example 3.2,the sort-increasing rule is converted to the conditional equation:

$$x * x = i_{Nat \subset Int}(n) \Rightarrow x * x = n$$

Conditional equations which have variables in their right-hand sides which do not occur in the left of this kind are not usually admitted. However, a deterministic valuation for n can be found by normalising $x * x$ and matching the result to $i_{Nat \subset Int}(n)$. Ganzinger gives a brief account of how the completion algorithm is modified to allow for such rules and gives further references to his own work for more details of how this is done.

Thus the problem of non-sort-decreasing rules can be handled by switching to the many-sorted algebra, which while detailed is conceptually straightforward and can be mechanised. However, this method uses a conditional completion algorithm, which is an involved process adding complexities of its own. Also, in coverting to a many-sorted signature, the elegance of order-sorted specification is lost.

3.4 Rewriting through Ill-Sorted Terms

A further class of approaches allows the use of ill-sorted terms in the rewriting process, unmediated by retracts, and with little modification to the algebra.

Rewriting in Connected Algebras. Gallier and Isakowitz [GI88] propose a new formalism for order-sorted algebra which stands between the overloaded and non-overloaded approaches. Confusingly, they call this approach *Overloaded Order-sorted Algebra* even though it introduces some of the features of the non-overloaded approach into the overloaded formalism. Gallier and Isakowitz's algebra depends on sorts being connected; a better name would be *connected order-sorted algebra*, which is used here.

Definition 3.8. Let $\equiv = (\leq \cup \leq^{-1})^+$ be the equivalence class induced by the partial ordering on sorts $\leq$. Sorts s and s' are *connected* if $s \equiv s'$.

Definition 3.9. Given an order-sorted signature Σ, a Σ-*Connected-Order-Sorted Algebra* $\mathcal{A}$ is a Σ-overloaded-order-sorted algebra such that given function symbol $f \in \mathcal{F}_{\Sigma(w,s)} \cap \mathcal{F}_{\Sigma(w',s')}$, where $w \equiv w'$ and $s \equiv s'$ then for every $x \in A_w \cap A'_w$, $f_{\mathcal{A}}^{w \to s}(x) = f_{\mathcal{A}}^{w' \to s'}(x)$.

A connectivity condition on signatures ensures initial algebras; if the argument sorts for two ranks of an operator are connected then so should the result sort. They also have a coherence condition to ensure that congruences over algebras are well-behaved. By developing this algebra, they avoid the non-intuitive consequences of the fully overloaded semantics mentioned above. More importantly in this thesis, the rewriting theory for this algebra has the following property.

Lemma 3.10. Given a rewriting system $u \simeq_{\mathcal{R}} t$ if and only if $u \leftrightarrow^*_{\mathcal{R}} t$.

$\simeq_{\mathcal{R}}$ is the least *overloaded order-sorted congruence of* R, a definition of congruence which forms congruences using connected ranks of an operator. This result states that rewriting is sound and complete for proving equational congruences. Thus any rewriting sequence which begins and ends with well-formed terms is a valid inference, *even if* the intermediate steps pass through ill-formed terms. This result has very important consequences as the sort-decreasing condition on rules is not needed to ensure the soundness of rewriting. Further, Gallier and Isakowitz state (*op. cit. p.15*):

> *Since the notion of structure presented in* [non-overloaded order-sorted algebra] *seems to be stronger than that of Overloaded Order-sorted Algebra* [that is Connected

Algebra] *it follows that our results on rewriting also hold* [on their] *structures.*

Gallier and Isakowitz do not present results covering completion, and rewriting modulo equational theories. More fundamentally, their theory on rewriting only covers rewriting sequences begun and terminated by well-sorted terms. However, the status of sequences terminating in ill-sorted terms is not considered. Such terms may occur in a completion procedure or be generated in a critical pair and should have a semantics and proof theory. Thus while this work provides the interesting pointer of ignoring the sort-decreasing restriction, further investigation is required.

Dynamic Sorting. Watson and Dick [WD89] take an approach similar to that of Gallier and Isakowitz, allowing computation through ill-sorted terms. They note that problems arise when there is a discrepancy between the syntactic sort of a term, determined by the signature alone, and the semantic sort, which is the minimal (named) set of denotations to which the denotation of the term belongs in any algebra. This is determined by an interaction between the signature and the equations of the specification. Watson and Dick give the following definition for a semantic sort.

Definition 3.11. Given an order-sorted specification $\mathcal{S} = (\Sigma, \mathcal{E})$, with a regular signature Σ, the *semantic sort*, $SS(t)$ of a term t is defined to be

$$SS(t) = \bigcap_{u =_{\mathcal{E}} t} \mathcal{LS}(u)$$

Given any rewrite rule, both sides must have the same semantic sort and recording the semantic sort of terms, rewriting is sound. Thus the result of Gallier and Isakowitz above is reproduced, except the intermediate 'ill-sorted' terms will be well-sorted using the semantic sort. However, the semantic sort of a term is undecidable in general, and thus so is matching. Consequently, they give a weaker definition, which they associated with the Knuth-Bendix Completion algorithm.

Definition 3.12. Given an order-sorted specification $\mathcal{S} = (\Sigma, \mathcal{E})$, with a regular signature Σ, the *approximate least sort*, $ALS(t)$, of a term t at stage $(n+1)$ of the Knuth-Bendix

procedure is the greatest lower bound of in the complete sort lattice of:

i). $ALS(t, n)$

ii). $\{ALS(u, n) | u \leftrightarrow^* t\}$

Watson and Dick use this to develop a Knuth-Bendix completion procedure. However, their work is sketchy. For example, they do not consider whether semantic sorts are allowed to occur in matching substitutions. Also their deduction system is not sound as the following example shows.

Example 3.13. Given the specification $\mathcal{S}$ as follows:

Sorts:	$A\ B\ C$
Subsorts:	$B \leq A, C \leq A$
Operators:	$b : \rightarrow B$ $\quad$ $c : \rightarrow C$
	$f : B \rightarrow A$
Equations:	$b = c$

Under either semantics for order-sorted specifications, the equation $f(b) = f(c)$ is not a valid consequence of this specification as the term $f(c)$ is ill-sorted. However, using the inference rules for dynamic sorting as given in Watson and Dick [WD89], the following inferences can be made. Equations are represented by a sort and two terms: $< S, t_1, t_2 >$, where S is the greatest lower bound of all the sorts that the t_1, t_2 have been proven to have. As equations are derived, this sort can move down the sort hierarchy and thus the sorts of equations are *dynamic.*

$$\dfrac{\dfrac{< B, b, b > \quad < \top, b, c >}{< B, b, c >} \qquad \dfrac{< C, c, c > \quad < \top, b, c >}{< C, b, c >}}{\dfrac{< B \cap C, b, c >}{< A, f(b), f(c) >}}$$

where $\top$ is the top sort. The final inference step is since $B \cap C \subseteq B$ and $f \in \mathcal{F}_{\Sigma(<B>,A)}$. Thus in this system, $f(b) = f(c)$ is a valid consequence of our specification, and thus the dynamic rules are unsound with respect to the standard models. □

Nevertheless, the idea in this paper is an intriguing: use the equational theory to define the sort of a term, and change this sort during deduction using equational reasoning.

3.5 Changing the Semantics

The method proposed by Chen and Hsiang [CH91b] follows from the approaches of Watson and Dick, and Gallier and Isakowitz by using syntactically ill-formed, but semantically meaningful terms. They redefine the meaning of an order-sorted algebra, and build up three layers of equational reasoning. They start by defining all possible terms which have a valid meaning in a specification; the $\mathcal{I}$-Terms.

Definition 3.14. The set of $\mathcal{I}$-Terms with respect to a specification $(\Sigma, \mathcal{E})$ is inductively defined as follows. For each $s \in S_\Sigma$, let

$$\begin{aligned} T_{\mathcal{I}}(\mathcal{F}_\Sigma, \mathcal{X})^0_s &= T_\Sigma(\mathcal{X}) \\ T_{\mathcal{I}}(\mathcal{F}_\Sigma, \mathcal{X})^{i+1}_s &= T_{\mathcal{I}}(\mathcal{F}_\Sigma, \mathcal{X})^i_s \cup \{u[p \leftarrow \sigma r] \mid u[p \leftarrow \sigma l] \in T_{\mathcal{I}}(\mathcal{F}_\Sigma, \mathcal{X})^i_s, \\ &\qquad l = r \in \mathcal{E},\ \forall x_{s'} \in \mathit{Vars}(l) \cdot \sigma(x_{s'}) \in T_{\mathcal{I}}(\mathcal{F}_\Sigma, \mathcal{X})^i_{s'}\} \end{aligned}$$

The set of all the $\mathcal{I}$-Terms of sort s, written as $T_{\mathcal{I}}(\mathcal{F}_\Sigma, \mathcal{X})_s$, is $\bigcup_{i \geq 0} T_{\mathcal{I}}(\mathcal{F}_\Sigma, \mathcal{X})^i_s$. The set of all the $\mathcal{I}$-Terms $T_{\mathcal{I}}(\mathcal{F}_\Sigma, \mathcal{X})$ is $\bigcup_{s \in S_\Sigma} T_{\mathcal{I}}(\mathcal{F}_\Sigma, \mathcal{X})_s$

Using this definition of ill-sorted terms, Chen and Hsiang develop deduction rules and replacement of equals for equals ($\longleftrightarrow_{\mathcal{I},\mathcal{E}}$), using the following definition of substitution

Definition 3.15. A substitution σ is a *$\mathcal{I}$-substitution* if $\forall x \in \mathit{Dom}\ \sigma$ if $x : s$ then $\sigma(x) \in T_{\mathcal{I}}(\mathcal{F}_\Sigma, \mathcal{X})_s$.

$\mathcal{I}$-deduction and $\mathcal{I}$-replacement prove to be equivalent. They then give the following semantics of the $\mathcal{I}$-Term algebra.

Definition 3.16. Let $\mathcal{S} = (\Sigma, \mathcal{E})$ be an order-sorted specification. An *$\mathcal{I}$-Algebra* $\mathcal{A}$ consists of a pair $(S_\Sigma^{\mathcal{A}}, \mathcal{F}_\Sigma{}^{\mathcal{A}})$ such that :

1. $S_\Sigma^{\mathcal{A}}$ is a set of sets indexed on the S_Σ, the *carriers* of Σ.
2. If $s \leq_\Sigma s'$ then $s^{\mathcal{A}} \subseteq s'^{\mathcal{A}}$.
3. $C_{\mathcal{A}} = \cup\{s^{\mathcal{A}} | s \in S_\Sigma\}$.
4. If $f \in \mathcal{F}_{\Sigma(<s_1 \ldots s_n>,s)}$ and if $a_i \in s_i^{\mathcal{A}}$ for $i = 1 \ldots i$ then $(a_1, \ldots, a_n) \in D_f^{\mathcal{A}}$ and $f^{\mathcal{A}}(a_1, \ldots, a_n) \in s^{\mathcal{A}}$.
5. If u is a Σ-term of sort s and $u \overset{*}{\longleftrightarrow}_{\mathcal{I},\mathcal{E}} t$, then for all $\mathcal{A}$-assignments $\alpha, \bar{\alpha}(t) \in s^{\mathcal{A}}$.

Chen and Hsiang develop the theory of $\mathcal{I}$-rewriting, which does have the desired properties missing from the normal order-sorted rewriting. However, it is undecidable whether a term is an $\mathcal{I}$-term, and thus, $\mathcal{I}$-matching, and $\mathcal{I}$-rewriting are also undecidable. To produce a decidable operational semantics for this theory they introduce a new concept intermediate between that of $\mathcal{I}$-deduction and the standard Σ-deduction, $\mathcal{W}$-deduction. Here, substitutions are restricted to well-formed substitutions, using Σ-terms, but we can still rewrite to $\mathcal{I}$-terms, similar to Gallier and Isakowitz, although the rewriting sequence does not have to begin or end with a Σ-term.

Chen and Hsiang prove that $\mathcal{W}$-deduction and $\mathcal{W}$-replacement are equivalent to $\mathcal{I}$-deduction and $\mathcal{I}$-replacement. However, $\mathcal{W}$-rewriting is not the same as $\mathcal{I}$-rewriting, unless the system is *sort-convergent*, a weaker condition than sort-decreasingness, but which is still decidable.

Definition 3.17. A Σ-rewrite step, $t \rightarrow_{\Sigma,R} u$ is a $\mathcal{I}$-rewrite step between Σ-terms u and t using rule $l \rightarrow r$ and a Σ-substitution σ such that $\mathcal{LS}(\sigma(r)) \leq \mathcal{LS}(\sigma(l))$.

Definition 3.18. A term-rewriting system R is *sort-convergent* if for any $l \rightarrow r \in R$ and any Σ-substitution σ, there is a Σ-term u such that $\sigma(l) \overset{*}{\longrightarrow}_{\Sigma,R} u \ {}_{\Sigma,R}\overset{*}{\longleftarrow} \sigma(r)$.

Chen and Hsiang show that this is sufficient to ensure the equivalence of the Church-Rosser theorem and confluence for well-sorted rewriting and also the following proposition.

Theorem 3.19. If R is terminating, sort-convergent and $\mathcal{W}$-confluent, then it is Σ-confluent.

To show confluence of $\mathcal{W}$-rewriting, the confluence of Σ-rewriting can be used, and this is

established for sort-convergent systems via a critical pair lemma for $\mathcal{W}$-rewriting. They use this to give a completion procedure using *sort enrichment* to extend the signature with new sorts and operators similar to [GKK90], whenever instances of rules are discovered which are not sort-convergent.

Thus Chen and Hsiang extend the semantics of order-sorted equational logic to cover ill-sorted but semantically meaningful terms and develop a rewriting theory for it. This is a most interesting approach, but also has several deficiencies.

The definition of order-sorted $\mathcal{I}$-algebra is unsatisfactory. Clause 5 of this definition, which gives a semantics for ill-sorted terms, links the rewriting theory with the semantics; it demands a denotation for ill-sorted terms precisely when they are reachable through rewriting. It would be better to give a definition of the $\mathcal{I}$-algebra which separates the denotation from the rewriting theory.

The restriction of sort decreasing rules is weakened to that of sort-convergent rules. It would be desirable to weaken this restriction still further, or remove it all together.

The completion procedure uses sort-enrichment, similar to [GKK90], and suffers from the same disadvantages. It would be preferable to produce a completion procedure without it.

Chen and Hsiang find that $\mathcal{I}$-rewriting is undecidable and so use $\mathcal{W}$-rewriting, which employs syntactic matching. Some intermediate rewriting technique could be used which uses some partial semantic information, as in Watson and Dick, to approximate more closely the ideal of $\mathcal{I}$-rewriting.

3.6 Further Approaches using Semantic Sorts

Two further approaches explore the notion of terms being well-sorted via the equational theory.

Using Term-Sort Declarations. Schmidt-Schauss [SS89] explores the notion of *Term-Sort Declarations*, a concept dating back to Goguen's original work [Gog78b]. A Term-Sort Declaration is an explicit assertion of the sort of terms.

Definition 3.20. A *Term-Sort Declaration* over a signature Σ is a pair $t : s$ where $t \in T_\Sigma(\mathcal{X})$ and $s \in S_\Sigma$. An order-sorted signature with declarations is a pair $(\Sigma, \mathcal{D})$ where $\mathcal{D}$ is a set of term-sort declarations over Σ. For convenience, we include the set $\{f(x_1 : s_1, \ldots, x_n : s_n) : s | f : s_1 \ldots s_n \rightarrow s \in \mathcal{F}_\Sigma\}$ within $\mathcal{D}$.

Example 3.21. A specification of the Evens can be given as:

Sorts:	*Nat, Even*
Subsorts:	$Even \leq Nat$
Operators:	$0 : \rightarrow Even$
	$succ : Nat \rightarrow Nat$
Declarations:	$succ(succ(x : Even)) : Even$

□

However, unification over specifications with term declarations is undecidable in general. With [Wit92] proposes a method by which a class of term-sort declarations can be accommodated. With first extends rewriting to allow rewriting through syntactically ill-sorted terms. He considers substitutions which are *semantically well-sorted* as follows.

Definition 3.22. Given a set of Σ-equations E, the *E-Semantical Sorts* of a term t is:

$$\text{Sorts}_E(t) = \bigcup_{t' =_E t} \text{Sorts}_\sigma(t')$$

A (Σ, E)-substitution σ is such that $Sorts_E(\sigma x) \leq s(x)$.

Then $t \Rightarrow_{\Sigma,R} t[u \leftarrow \sigma r]$ in the *E_R-rewrite relation* if there is a rewrite rule $l \rightarrow r \in R$, $u \in O(t)$ such that $t|_u = \sigma l$ for some (Σ, E)-substitution.

Using this rewrite relation, a critical pair lemma can be given, but With notes (*op. cit p.399*): *this theorem is mainly of theoretical interest because in general it is undecidable*

whether a substitution is semantically well-sorted.

With then gives a more powerful method than that given in [SS89] for showing that specifications with term-sort declarations are weakly sort-decreasing. He first discusses introducing new term-sort declarations into a specification to convert the signature into a weakly sort-decreasing one. This is also given in [SS89].

Definition 3.23. If $t : s$ is a term-sort declaration, $l \rightarrow r \in R$, and if there is a Σ-substitution such that $\sigma(t|_p) = \sigma l$ for some non-variable path $p \in O(t)$, then the pair $(\sigma t, \sigma t[p \leftarrow r])$ is called the *critical sort relation.*

A critical sort relation (t, t') is satisfied if $\mathcal{LS}(t') \leq \mathcal{LS}(t)$. It is weakly satisfied if $t' \rightarrow_R^* t''$ and $\mathcal{LS}(t'') \leq \mathcal{LS}(t)$.

Theorem 3.24. If Σ is a regular signature, and R a term-rewriting system, then $\rightarrow_R$ is weakly sort-decreasing if and only if all critical sort relations are weakly satisfied.

This gives rise to a method of establishing the weakly sort-decreasing property. With thus goes on to investigate a criterion for deciding unification in particular cases.

Definition 3.25. A *critical overlap* between term-sort declarations $t_1 : s_1$ and $t_2 : s_2$ is given by σt_1 where σ is such that $\sigma(t_1|_p) = \sigma t_2$ for $p \in O(t_1)$. A critical overlap is solved if $\sigma t_1 : s_1 \in \mathcal{D}$, if $p \neq \lambda$ or $\sigma t_1 : \{s_1, s_2\} \in \mathcal{D}$, $p = \lambda$.

Theorem 3.26. If Σ is a signature with term-sort declarations, unification is decidable and finitary if every critical overlap between term-sort declarations is solved.

This gives rise to a mechanism for generating new term-sort declarations which may make unification decidable; this process may fail to terminate. Finally, With considers an open problem of Schmidt-Schauss's and states that unification is decidable for *linear signatures*, that is signatures with sort declarations which have no repeated variables. This result, and the extension to *semi-linear signatures* is also given in [SA93].

With's approach is interesting as he first tries to give a semantic approach to order-sorted

term rewriting. He notes that this is not decidable and thus restricts his attention to extending existing theory to show that for linear signatures with term-sort declarations, this theory is decidable, using the new techniques that he has developed.

Another Semantic Approach. Werner [Wer93] gives another, semantic approach to the order-sorted rewriting, while still keeping to the non-overloaded algebra. He defines extended terms to be the well-formed terms constructed without considering sorts, and the Σ-terms to be the syntactically well-sorted. Like Chen and Hsiang, he extends the set of well-sorted terms to those which are *semantically* well-sorted, defined as follows. The relation $\overset{*}{\Leftrightarrow}_R^{\Sigma}$ is the equational replacement relation over all extended terms, restricted to using Σ-substitutions.

Definition 3.27. ([Wer93], Definition 2.) Given signature Σ and rules R, $T_{\Sigma,R}(\mathcal{X})_s$ is the set of all (Σ, R)-Terms (semantically well-sorted) of sort s:

$t \in T_{\Sigma,R}(\mathcal{X})_s$ if and only if $\exists t' \in T_{\Sigma}(\mathcal{X})_s$ such that$t \overset{*}{\Leftrightarrow}_R^{\Sigma} t'$

$T_{\Sigma,R}(\mathcal{X}) = \bigcup_{s \in S_\Sigma} T_{\Sigma,R}(\mathcal{X})_s$.

This definition again captures exactly those terms which can have sorts semantically. Werner then goes on to develop the rewriting relations $\rightarrow^{\Sigma}$ and $\rightarrow^{\Sigma,R}$ which are rewriting over extended terms using Σ-substitutions and (Σ, R)-substitutions respectively, and gives the following interesting result.

Theorem 3.28. ([Wer93], Theorem 11.) Given $t, t', t'' \in T_{\Sigma,R}(\mathcal{X})$, $p \in O(t)$, and a binary relation $\rightarrow^A$ on (Σ, R)-terms such that $\rightarrow^{\Sigma} \subseteq \rightarrow^A \subseteq \rightarrow^{(\Sigma,R)}$, the following holds.

1. **Strong Compatibility** If $t' \rightarrow^A t''$ then $t[p \leftarrow t'] \in T_{\Sigma,R}(\mathcal{X})$ if and only if $t[p \leftarrow t''] \in T_{\Sigma,R}(\mathcal{X})$.
2. **Completeness** $t' \overset{*}{\leftrightarrow}^A t''$ if and only if $t' \overset{*}{\Leftrightarrow}_R^{\Sigma} t''$ if and only if $t' =_R t''$.

Thus any rewriting relation between $\rightarrow_R^{\Sigma}$, $\rightarrow_R^{(\Sigma,R)}$ on extended terms is strong enough to represent the equational theory. However, the critical pair lemma does not hold for $\rightarrow_R^{\Sigma}$ unless R is sort decreasing.

He notes that (Σ, R)-unification is undecidable as is whether an extended term is a (Σ, R)-term, and goes on to develop the special case of $\rightarrow_R^{(\Sigma,R)}$ where Σ is range unique.

Definition 3.29. Signature Σ is *range unique* if whenever $f : s_1 \dots s_n \rightarrow s \in \mathcal{F}_\Sigma$ and $f : s'_1 \dots s'_n \rightarrow s' \in \mathcal{F}_\Sigma$, then $s = s'$.

Definition 3.30. If Σ is range unique, a *T-substitution* σ, is a (σ, R)-substitution such that:

1. if $\sigma(x : s) = f(t_1, \dots t_n)$ then there is a s' such that $f : \omega \rightarrow s' \in \mathcal{F}_\Sigma$ and $s' \leq s$;

2. if $\sigma(x : s) = y : s'$

T-rewriting $\rightarrow_R^T$ is $\rightarrow_R^{\Sigma,R}$ restricted to T-substitutions.

We have $\rightarrow_R^\Sigma \subseteq \rightarrow_R^T \subseteq \rightarrow_R^{(\Sigma,R)}$ and so the theorem above applies. Further, Werner demonstrates that T-unification is decidable and thus completion is possible over range-unique signatures. He then goes on to give a construction which can convert any specification into an equivalent range-unique one.

Werner's approach is interesting for several reasons. He recognises the importance of having all terms available, including ill-formed ones, and then uses a semantic description of the well-sorted ones, although he also recognises that it is sufficient in this case to use Σ-substitutions. The above quoted theorem shows that it is sufficient to use any rewriting relation between the 'syntactic' and the 'semantic', and $\rightarrow_R^T$ is one such rewriting relation. He also points out the undecidability of semantic unification.

Nevertheless there are problems with Werner's approach. He uses the standard non-overloaded semantics. As we have seen this semantics does not capture the intended meaning of order-sorted specification, although Werner's rewriting systems do, so Werner's approach is unsound with respect to the standard approach. Also, the semantic information on the sorts of terms which is deduced during rewriting is not recorded or reused in other circumstances, which seems a waste of the expressive power of order-sorted rewriting. Further, the decidable relation that he gives is only valid over range-unique signatures, which

is a only small class of order-sorted signatures. His transformation of any signature into a range-unique one suffers from the syntactic complexity of other transformation systems that we have seen before.

3.7 Decorated Rewrite Rules

Hintermeier, Kirchner and Kirchner [HKK93, HKK94] have independently developed a method for order-sorted computations which is similar to the dynamic sorting scheme presented later in this thesis. As in this thesis, they define an extended class of terms, which they call *decorated terms*, where each node of a term is annotated with a set of sorts recording which sorts the term is valid in. They proceed to define matching, unification, rewriting and completion in terms of these decorated terms in a fashion superficially similar to that presented in this thesis, and we shall note similarities and differences to the presentation given in this thesis in subsequent chapters. However, we do point out here that their work differs significantly from the work in this thesis in the following respects:

- They use the Galactic Algebra paradigm of Mégrelis [Még92] to provide a semantic basis for their work. This is a more expressive but more complex scheme than the embedded algebras used in this thesis. The work reported in this thesis is considered more intuitive and closer to the existing work on order-sorted algebras.

- The form of matching used is quite different. They use an exact syntactic matching of decorated terms which matches sorts exactly, and allows sort variables to appear in sort decorations. This thesis defined a matching based on an approximation of sorts, which is strictly more powerful.

- The form of unification used is also different for the same reasons as the matching above. This thesis also allows ill-sorted unifiers to occur in certain circumstances.

- Only one kind of rewriting is used, where two forms are defined in this thesis, one for sorts (static and dynamic) and one for terms. However, to give the same rewriting power and also to overcome the strict syntactic nature of the decorated term matching

and unification, they also define a parallel set of rewrite rules which only rewrite the sort decorations, a set of rules which are conditional. Thus in this method, more rewrite rules are generated, with consequently a higher chance of divergence in completion.

- There is no analogue to the 'constrained' method described in Chapter 8.

Thus this independently developed method has a quite different style and methodology to the method described in this thesis, even though many of the results are similar. This paper has been useful and inspiring to the work presented in this thesis.

3.8 Summary

In this chapter the problems associated with the standard order-sorted theory and various approaches to their resolution. No approach seems entirely satisfactory, although Gallier and Isakowitz, Watson and Dick, Chen and Hsiang, With, Werner, and Hintermeier, Kirchner and Kirchner all seem to have interesting partial solutions. It is clear from Watson and Dick and Chen and Hsiang that the semantics of order-sorted algebra need rethinking. The appearance of ill-sorted terms in reasoning steps points to the need to give such terms meaning, which With and Werner point out requires a new approach to the sorting of terms, including syntactically ill-sorted terms. Chen and Hsiang, and Hintermeier, Kirchner and Kirchner give approaches towards a new semantics for order-sorted signatures. However, each of these has their difficulties and none of these solutions seems entirely satisfactory. There is scope for a new approach to semantics and operational interpretation of order-sorted specifications, which takes inspiration from all the approaches discussed in this chapter. In the following Chapters 4-8 we discuss a new approach to order-sorted algebra, equational logic, and rewriting.

Chapter 4

A Semantics for Dynamic Order-sorted Equational Logic

4.1 Introduction

Order-sorted equational logic is a powerful extension to unsorted equational logic, and allows the elegant handling of partial functions and error values. However, the paradigm is limited by restrictions on the equational theories which can be defined. We would like to remove these restrictions, while retaining the elegance and power of order-sorted reasoning.

Of the existing approaches devised to overcome these restrictions, dynamic order-sorted logic [WD89] is particularly attractive since it has a strong intuitive basis. However, this method and others are unsound with respect to the standard semantics of order-sorted equational logic. In this is chapter, a reformulation of the semantics of order-sorted equational logic is given. This approach is similar to the *Standard Semantics* sketched in Chapter 2, but allows for sorts of terms to be determined using the equational theory.

There are three alternative views of specifications: the Model theory; the Equational Logic and the Rewriting theory, shown in Figure 4.1. The Model Theory describes an abstract

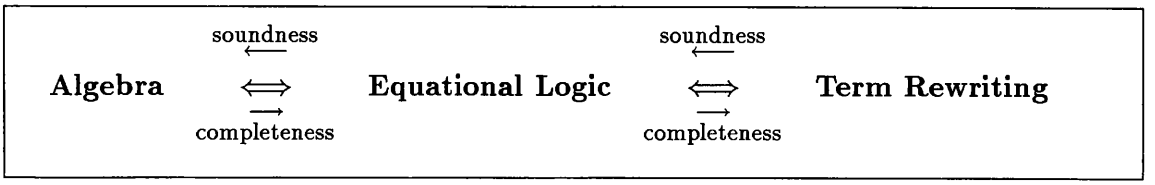


Figure 4.1: Views on (Algebraic) Specifications

mathematical entity which may be non-constructive; the Proof Theory, or Logic describes the system in terms of logical rules for deriving true theorems by symbolic inference and proof; rewriting, or more generally replacing equals by equals, is an operational view of the system which describes the system in terms of computations. The three views can be seen as decreasing in abstractness, but increasing in computability. They should be developed independently and soundness and completeness results provided to show their equivalence. This chapter will give the first and second views for order-sorted specifications; the following chapters will explore the operational semantics via term rewriting.

The difficulties with order-sorted equational specifications arise from the same source: the equational axioms have models which would naturally give denotations to terms which are ill-sorted in the standard definition and thus have no valid meaning. As discussed in Chapter 3, attempts to overcome this problem in [GI88, WD89, CH91b, Wit92, Wer93, HKK93] rely on deduction through ill-sorted terms. These terms have no denotation and these deductions are unsound in the standard model. Thus the standard model is not sufficiently powerful for full order-sorted deduction.

In Example 3.13, in the dynamically sorted system, $f(b) = f(c)$ is a valid consequence of the specification, although it is not a valid consequence of the specification in the standard semantics, and thus the dynamic rules are unsound with respect to the standard model. Our solution to this problem is not to discard the idea of dynamic sorting, but to change the notion of a model.

The problem is that $b = c$ is satisfied in the model, but $f(b) = f(c)$ is not as $f(c)$ is an ill-formed term, and therefore not available in the domain of discourse. Thus the congruence rule, while sound, does not produce the expected result, although in *any* model which satisfies the given equational theory $b = c$ there will *always* be a satisfactory denotation for $f(c)$. Since $b = c$, in any $\mathcal{S}$-model $\mathcal{A}$, $b^{\mathcal{A}} = c^{\mathcal{A}}$ and thus as $f^{\mathcal{A}}$ is a well-defined function on

$B^{\mathcal{A}}$, we know that $f^{\mathcal{A}}(c^{\mathcal{A}})$ has a value which is the same as $f^{\mathcal{A}}(b^{\mathcal{A}})$. But the definition of satisfaction of equations prevents the expression of equations valid in the model since such equations involve ill-sorted terms.

Our solution is to denote ill-formed terms in valid models in a controlled fashion. Such terms take the same denotation as any well-formed term to which they are equal. Classically, the initial algebra allows no junk and no confusion. This system allows 'junk' in the model of the sorts, but only if it is identified with a non-junk element.

4.2 Signatures and Terms

To give a semantics for Dynamic Order-Sorted Specifications in which ill-sorted terms have valid denotations, ill-sorted terms need to be available in the domain of discourse. Consequently, order-sorted signatures are embedded within unsorted ones so unsorted terms are available. We modify the definitions of order-sorted signatures given in Chapter 2.

Definition 4.1. An *Unsorted Signature*, $\overline{\Sigma}$ is a pair, $(\overline{\mathcal{F}}_{\overline{\Sigma}}, \bar{\alpha})$ where:

1. $\overline{\mathcal{F}}_{\overline{\Sigma}}$ is a set of operators.
2. $\bar{\alpha} : \overline{\mathcal{F}}_{\overline{\Sigma}} \rightarrow I\!N$ is an *arity* function.

Assume an infinite set of variables $\mathcal{X}$. The set of unsorted terms is as follows.

Definition 4.2. Unsorted Terms.
The set of unsorted terms $\overline{T}_{\overline{\Sigma}}(\mathcal{X})$ is the set of $\overline{\Sigma}$-Terms, the least set such that:

1. If $x \in \mathcal{X}$, $x \in \overline{T}_{\overline{\Sigma}}(\mathcal{X})$
2. If $c \in \overline{\mathcal{F}}_{\overline{\Sigma}}$ and $\bar{\alpha}(c) = 0$ then $c \in \overline{T}_{\overline{\Sigma}}(\mathcal{X})$.
3. If $f \in \overline{\mathcal{F}}_{\overline{\Sigma}}$, $\bar{\alpha}(f) = n$ and $t_1, \ldots t_n \in \overline{T}_{\overline{\Sigma}}(\mathcal{X})$ then $f(t_1, \ldots t_n) \in \overline{T}_{\overline{\Sigma}}(\mathcal{X})$.

If a term has no variables it is *ground*. The set of ground unsorted terms is denoted $\overline{T}_{\overline{\Sigma}}$.

Order-sorted signatures introduce sorts and an ordering on sorts. A *sort symbol* is a name with no structure; a *subsort declaration* is a pair of sorts of the form $s_1 \leq s_2$. We construct order-sorted signatures from the definition of unsorted signatures.

Definition 4.3. An *Order-Sorted Signature* Σ built on an unsorted signature $\overline{\Sigma}$, is a triple, $(S_\Sigma, \leq_\Sigma, \mathcal{F}_\Sigma)$ where:

1. S_Σ is a set of sorts.

2. $\leq_\Sigma$ is a partial ordering on S_Σ, the least transitive-reflexive closure of a set of subsort declarations.

3. $\mathcal{F}_\Sigma$ is the set $\overline{\mathcal{F}}_{\overline{\Sigma}}$ together with a function $rank_\Sigma : \overline{\mathcal{F}}_{\overline{\Sigma}} \rightarrow I\!P(S_\Sigma^* \times S_\Sigma)$ such that if $\forall f \in \overline{\mathcal{F}}_{\overline{\Sigma}} \cdot (w, s) \in rank_\Sigma(f)$ then $|w| = \bar{\alpha}(f)$.

For convenience, we write $f : s_1, \ldots, s_n \rightarrow s$ when $(\langle s_1, \ldots, s_n \rangle, s) \in rank(f)$. When the signature is clear in context, the Σ can be dropped. By abuse of notation, we refer to $s \in \Sigma$ when $s \in S_\Sigma$ and $f \in \Sigma$ when $f \in \mathcal{F}_\Sigma$. Also, we extend the ordering on sorts to sequences of sorts and to pairs in $S_\Sigma^* \times S_\Sigma$ in the obvious, pointwise pairwise fashion.

Assume that the set of variables $\mathcal{X}$ can be partitioned into a set of variables $\mathcal{X}_s$ for each sort $s \in S_\Sigma$. The sort of a variable x is given by its sort assignment: $s(x) = s$ if and only if $x \in \mathcal{X}_s$.

Given this definition of an order-sorted signature Σ and a total sort assignment to variables $s : \mathcal{X} \rightarrow S_\Sigma$, we define the set of syntactically well-sorted terms $T_\Sigma(\mathcal{X}) \subseteq \overline{T}_{\overline{\Sigma}}(\mathcal{X})$.

Definition 4.4. Sorted Terms (Σ-Terms).
The set of Σ-Terms of a sort $s \in \Sigma$, $T_s(\mathcal{X})$ is the least set constructed as follows:

1. If $x \in \mathcal{X}$, $s(x) = s'$ and $s' \leq_\Sigma s$, then x is a (variable) Σ-Term of sort s.

2. If $f \in \mathcal{F}_\Sigma$, $(\langle s_1, \ldots, s_n \rangle, s') \in rank(f)$, $n \geq 0$, $s' \leq_\Sigma s$ and $t_1, \ldots, t_n$ are Σ-Terms of sorts $s_1, \ldots, s_n$ respectively, then $f(t_1, \ldots, t_n)$ is a (compound) Σ-Term of sort s. If $n = 0$ then f is a *constant* of sort s.

The set of all Σ-terms $T_\Sigma(\mathcal{X}) = \bigcup_{s \in S_\Sigma} T_s(\mathcal{X})$. The set of ground Σ-terms is denoted T_Σ. If a term $t \in T_s(\mathcal{X})$ then it is denoted $t : s$.

4.3 Unsorted Algebras

Algebras for unsorted signatures are defined in a standard fashion. See for example [EM85] for a full treatment of such algebras.

Definition 4.5. An *Unsorted Algebra*, $\bar{A}_{\overline{\Sigma}}$, of an unsorted signature $\overline{\Sigma}$ (also known as a $\overline{\Sigma}$-Algebra) consists of a pair $(\bar{A}, \overline{\mathcal{F}}^{A}_{\overline{\Sigma}})$ such that :

1. $\bar{A}$ is the *carrier* set of $\bar{A}_{\overline{\Sigma}}$.
2. For each operator f in $\overline{\mathcal{F}}_{\overline{\Sigma}}$ there is a corresponding function $f^{\mathcal{A}} : \bar{A}^{\bar{\alpha}(f)} \rightarrow \bar{A}$ in $\overline{\mathcal{F}}^{A}_{\overline{\Sigma}}$.

Thus a signature is denoted by a class of algebras. The algebras of a signature are related using Homomorphisms.

Definition 4.6. Given an unsorted signature $\overline{\Sigma}$ and two $\overline{\Sigma}$-algebras $\bar{\mathcal{A}}$ and $\bar{\mathcal{B}}$ an *Unsorted $\overline{\Sigma}$-Homomorphism* $\bar{h} : \bar{\mathcal{A}} \rightarrow \bar{\mathcal{B}}$ is a mapping between the algebras such that:

$$\bar{h}(f^{\bar{\mathcal{A}}}(a_1, \ldots a_n)) = f^{\bar{\mathcal{B}}}(\bar{h}(a_1), \ldots, \bar{h}(a_n)) \text{ for all } f \in \overline{\mathcal{F}}_{\overline{\Sigma}}, \text{ and } (a_1, \ldots a_n) \in \bar{A}^{\bar{\alpha}(f)}$$

It is trivial to show that $\overline{\Sigma}$-algebras and $\overline{\Sigma}$-homomorphisms form a category, $\mathbf{Alg}_{\overline{\Sigma}}$.

A homomorphism $\bar{h} : \bar{\mathcal{A}} \rightarrow \bar{\mathcal{B}}$ is an *isomorphism* if there is a homomorphism $\bar{h}' : \bar{\mathcal{B}} \rightarrow \bar{\mathcal{A}}$ such that $\bar{h}' \circ h = \bar{id}_{\mathcal{A}}$ and $\bar{h} \circ \bar{h}' = \bar{id}_{\mathcal{B}}$, where $\bar{id}_{\mathcal{A}}$ ($\bar{id}_{\mathcal{B}}$) is the identity homomorphism for $\bar{\mathcal{A}}$ ($\bar{\mathcal{B}}$).

One algebra is of special significance: the *Term-Algebra.*

Definition 4.7. Given an unsorted signature $\overline{\Sigma}$, and a set of variables $\mathcal{X}$, the free unsorted *Term-Algebra* $\bar{T}_{\overline{\Sigma},\mathcal{X}}$ on $\mathcal{X}$ is defined to be the following construction on $\overline{T}_{\overline{\Sigma}}(\mathcal{X})$.

1. $\bar{\mathcal{A}}^{\bar{\mathcal{T}}_{\overline{\Sigma},\mathcal{X}}} = \overline{T}_{\overline{\Sigma}}(\mathcal{X})$.
2. $f^{\bar{\mathcal{T}}_{\overline{\Sigma},\mathcal{X}}}(t_1,\ldots,t_n) = f(t_1,\ldots,t_n)$

Note that if the set of variables $\mathcal{X}$ is empty, then the term algebra $\bar{\mathcal{T}}_{\overline{\Sigma},\emptyset}$ is formed by ground terms. This algebra is canonical of all $\overline{\Sigma}$-algebras in the following sense.

Definition 4.8. If $\bar{\mathcal{A}}$ is a $\overline{\Sigma}$-algebra, and $\mathcal{V} \subseteq \mathcal{X}$ a set of variables, a $(\mathcal{V},\bar{\mathcal{A}})$-*assignment* is a mapping $\bar{\nu} : \mathcal{V} \rightarrow \bar{A}$. The *denotation* of a term t in an $\overline{\Sigma}$-algebra $\bar{\mathcal{A}}$ and under an assignment $\bar{\nu}$ on $Vars(t)$, written $t^{\bar{\nu}}$, is a mapping from $\bar{\mathcal{T}}_{\overline{\Sigma},\mathcal{V}}$ to $\bar{\mathcal{A}}$ given by:

$$\begin{aligned} [\![x]\!]_{\bar{\nu}} &= \bar{\nu}(x) \\ [\![f(t_1,\ldots,t_n)]\!]_{\bar{\nu}} &= f^{\bar{\mathcal{A}}}([\![t_1]\!]_{\bar{\nu}},\ldots,[\![t_n]\!]_{\bar{\nu}}) \end{aligned}$$

Thus for any $(\mathcal{V},\bar{\mathcal{A}})$-assignment $\bar{\nu}$, the denotation function $[\![\_]\!]_{\bar{\nu}} : \bar{\mathcal{T}}_{\overline{\Sigma},\mathcal{V}} \rightarrow \bar{\mathcal{A}}$ is a homomorphism. In particular, as a corollary of this result, if $\mathcal{V} = \emptyset$ there is only the trivial $(\emptyset,\bar{\mathcal{A}})$-assignment. Thus there is a unique (up to isomorphism) homomorphism from $\bar{\mathcal{T}}_{\overline{\Sigma},\emptyset}$ to $\bar{\mathcal{A}}$ and $\bar{\mathcal{T}}_{\overline{\Sigma},\emptyset}$ is the *Initial Object* in $\mathbf{Alg}_{\overline{\Sigma}}$, and called *the Initial Algebra.*

4.4 Order-Sorted Algebra

An Order-Sorted Algebra can be defined as embedded within an Unsorted Algebra.

Definition 4.9. Given an order-sorted signature Σ built upon an unsorted signature $\overline{\Sigma}$, an *Order-Sorted* Σ*-Algebra* $\mathcal{A}$ is an unsorted $\overline{\Sigma}$-algebra $\bar{\mathcal{A}}$ with the additional properties:

1. For each $s \in S_\Sigma$ there is a subset $A_s \subseteq \bar{A}$
2. If $s \leq_\Sigma s'$ then $A_s \subseteq A_{s'}$
3. If $f : s_1,\ldots,s_n \rightarrow s$ and $a_1 \in A_{s_1},\ldots,a_n \in A_{s_n}$ then $f^{\mathcal{A}}(a_1,\ldots,a_n) \in A_s$.

Thus the Σ-Algebra $\mathcal{A}$ is *embedded* within the $\overline{\Sigma}$-algebra $\bar{\mathcal{A}}$, or simply $\mathcal{A}$ is in $\bar{\mathcal{A}}$.

There are several comments which can be made concerning this definition:

- We don't need to state properties about the domains and ranges of functions; these details are imported from the embedding unsorted algebra.
- This definition is consistent with the *non-overloaded* interpretation of the standard semantics, as the embedding unsorted algebra enforces the use of one function to represent each operator, rather than a family of functions as in the standard overloaded semantics.
- One unsorted $\overline{\Sigma}$-algebra can embed many different Σ-algebras.
- There is *always* a $\overline{\Sigma}$-algebra for any order-sorted signature. Let $\bar{A} = \bigcup_{s \in S_\Sigma} T_s$.
- As we assume a ground term for every $s \in S_\Sigma$, the set A_s cannot be empty.

Again we can relate the algebras of an order-sorted signature using homomorphisms.

Definition 4.10. Given an order-sorted Σ built on an unsorted signature $\overline{\Sigma}$ and two Σ-algebras $\mathcal{A}$ and $\mathcal{B}$ on $\overline{\Sigma}$-algebras $\bar{\mathcal{A}}$ and $\bar{\mathcal{B}}$ respectively an *Order-Sorted Σ-Homomorphism* $h : \bar{\mathcal{A}} \rightarrow \bar{\mathcal{B}}$ is a mapping between the $\overline{\Sigma}$-algebras such that

$$\forall s \in S_\Sigma \cdot h(A_s) \subseteq B_s$$

Σ-algebras and Σ-homomorphisms together form a category, $\mathbf{OSAlg}_\Sigma$.

Similarly, we can define the order-sorted term-algebra.

Definition 4.11. Given an order-sorted signature Σ on $\overline{\Sigma}$, and a set of variables $\mathcal{X}$, with a sort assignment s, the free order-sorted *Term-Algebra* $\mathcal{T}_{\Sigma,\mathcal{X}}$ on $\mathcal{X}$ is the unsorted term algebra $\bar{\mathcal{T}}_{\Sigma,\mathcal{X}}$ which has been divided into subsets such that:

$$s^{\mathcal{T}_{\Sigma,\mathcal{X}}} = \{t | t \in T_\Sigma(\mathcal{X}) \text{ and } t : s\}$$

This is clearly a Σ-algebra as defined above. If the set of variables $\mathcal{X}$ chosen is empty, then $\mathcal{T}_{\Sigma,\emptyset}$ is formed by the ground terms. This Σ-algebra is canonical in the following sense.

Definition 4.12. If $\mathcal{A}$ is a Σ-algebra in $\bar{\mathcal{A}}$, and $\mathcal{V} \subseteq \mathcal{X}$ a set of variables, with sort assignment s, a sorted $(\mathcal{V}, \mathcal{A})$-*assignment* is a mapping $\nu : \mathcal{V} \rightarrow \mathcal{A}$ such that $\forall x \in \mathcal{V} . \nu(x : s) \in \mathcal{A}^s$.

The *denotation* of a term t in a Σ-algebra $\mathcal{A}$ and under an assignment ν on $Vars(t)$, written t^ν, is a mapping from $\mathcal{T}_{\Sigma,\mathcal{V}}$ to $\mathcal{A}$ given by:

$$\begin{aligned} x^\nu &= \nu(x) \\ f(t_1,\ldots,t_n)^\nu &= f^{\mathcal{A}}(t_1^\nu,\ldots,t_n^\nu) \end{aligned}$$

Thus for any $(\mathcal{V},\mathcal{A})$-assignment ν, the denotation function $\_^\nu : \mathcal{T}_{\Sigma,\mathcal{V}} \rightarrow \mathcal{A}$ is a homomorphism. In particular, as a corollary of this result, if $\mathcal{V} = \emptyset$ there is only the trivial $(\emptyset,\mathcal{A})$-assignment. Thus there is a unique (up to isomorphism) homomorphism from T_Σ to $\mathcal{A}$ and thus T_Σ is the *Initial Object* in OSAlg_Σ, the *Initial Algebra.*

From the definition of syntactic sort and the valuation under sorted assignment, we can show the following proposition.

Theorem 4.13. If $t : s$ then for any sorted $(Vars(t),\mathcal{A})$-assignments ν, $t^\nu \in \mathcal{A}^s$.

Proof (4.13). If $t \in \mathcal{X}$, then by definition of sorted $(Vars(t),\mathcal{A})$-assignment $\nu(t) \in \mathcal{A}^s$. If $t = f(t_1,\ldots,t_n)$ then by definition of syntactic sort, there exists a rank for $f : s_1,\ldots s_n \rightarrow s$ and $t_1 : s_1 \ldots t_n : s_n$. By induction hypothesis on the depth of t, $t_i^\nu \in \mathcal{A}^{s_i}$, as $Vars(t_i) \subseteq Vars(t)$. Thus $t^\nu = f^{\mathcal{A}}(t_1^\nu,\ldots,t_i^\nu) \in \mathcal{A}^s$ by definition of $f^{\mathcal{A}}$. □

4.5 Equations and Models

Having constructed a two-tier semantics for order-sorted algebras embedded within unsorted algebras, we introduce the notion of equations.

Definition 4.14. An *unsorted* $\overline{\Sigma}$*-Equation* is a triple (Y,t,t') consisting of a set of variables Y and a pair of $\overline{\Sigma}$-terms t, t', such that $Vars(t) \cup Vars(t') \subseteq Y$. This triple is usually written $\forall Y.t = t'$. If the variable set $Y = Vars(t) \cup Vars(t')$, then we can omit the variable set and write $t = t'$. An *Unsorted Specification* consists of a pair $(\overline{\Sigma},\bar{E})$ where $\overline{\Sigma}$ is an unsorted signature and $\bar{E}$ a set of $\overline{\Sigma}$-equations.

Similarly, an *Order-Sorted* Σ*-Equation* is a triple of a set of variables and a pair of Σ-terms written $\forall Y.t = t'$. An *Order-Sorted Specification* consists of a pair (Σ, E) where Σ is an order-sorted signature (implicitly defined in terms of a unsorted signature $\overline{\Sigma}$) and E a set of order-sorted equations[1].

Thus an order-sorted specification is embedded within an unsorted specification.

We wish to investigate the equational models of order-sorted specifications and develop computational methods which reflect these models. Thus we need to develop a theory of these models. As usual in this development, we begin with a model for the unsorted theory.

4.5.1 Unsorted E-Algebras

We define validity of equations under a specification $\overline{\mathcal{S}} = (\overline{\Sigma}, E)$ in a $\overline{\Sigma}$-algebra $\bar{\mathcal{A}}$. An equation $\forall Y.t = t'$ is *Valid* in an algebra $\bar{\mathcal{A}}$ (or algebra $\bar{\mathcal{A}}$ *Satisfies* equation $\forall Y.t = t'$), written $\bar{\mathcal{A}} \models \forall Y.t = t'$, as defined in the following:

$$\bar{\mathcal{A}} \models \forall Y.t = t' \Leftrightarrow \forall (Y, \bar{\mathcal{A}})\text{-assignments } \bar{\nu} \cdot t^{\bar{\nu}} = t'^{\bar{\nu}}$$

We say that $\bar{\mathcal{A}}$ is a *Model* for $\overline{\mathcal{S}}$ (or $\bar{\mathcal{A}}$ is a $\overline{\mathcal{S}}$*-Algebra*) if every $\forall Y.t = t' \in E$ is valid in $\bar{\mathcal{A}}$. Again, the class of models for $\overline{\mathcal{S}}$ and homomorphisms between them form a category $\mathbf{OSAlg}_{\overline{\mathcal{S}}}$.

Given a specification $\overline{\mathcal{S}}$, the equational theory is the class of equations which are valid in all models of $\overline{\mathcal{S}}$: an equation $\forall Y.t = t'$ is *valid* under a specification $\overline{\mathcal{S}}$, written $\overline{\mathcal{S}} \models \forall Y.t = t'$, if $\forall Y.t = t'$ is valid in every model of $\overline{\mathcal{S}}$.

4.5.2 Order-Sorted E-Algebras

The Example 3.1 demonstrates that it is not sufficient to define a similar model for order-sorted specifications. In such a model, equational consequences of the dynamic method

[1] Here we restrict the equations in the specification to be statically sorted. We can relax this to unsorted equations, as long as the equations prove to be dynamically well-sorted in the sense defined below.

become invalid, even though perfectly satisfactory elements for the unsorted terms exist in the model. We need to modify the concept of the validity of equalities in a model.

The principle behind this change in interpretation is to allow equations which are valid in the unsorted model to be valid in the order-sorted model *provided that they have at least one element equal to some other well-sorted element.* Then they denote the same element in the model as the denotation of a well-sorted term and thus have a valid denotation. That is $t_1 = t_2$ in an order-sorted algebra if and only if for some u, $t_1 = u : S$ or $t_2 = u : S$ and $t_1 = t_2$ in the unsorted model. Not all equalities in the unsorted model can hold as the following example shows.

Example 4.15. Consider the following order-sorted specification, $\mathcal{S}$.

Sorts:	$A\ B$	
Operators:	$a : \rightarrow A$	$f : A \rightarrow A$
	$b : \rightarrow B$	$f : B \rightarrow B$
Equations:	$f(x : A) = a$	

Ignoring the sort information, then any $\overline{\mathcal{S}}$-algebra $\bar{\mathcal{M}}$ will satisfy $f(b) = a$. □

Thus we alter the notion of an unsorted algebra satisfying an equation, and only consider assignments that preserve the sorts of variables.

Definition 4.16. Given an order-sorted specification $\mathcal{S} = (\Sigma, E)$ built on the unsorted specification $\overline{\mathcal{S}} = (\overline{\Sigma}, E)$, then the $\overline{\Sigma}$-algebra $\bar{\mathcal{A}}$, containing sorted Σ-algebra $\mathcal{A}$, *Satisfies Sortedly* $\overline{\mathcal{S}}$ if and only if for all sorted assignments $\nu : \mathcal{X} \rightarrow \mathcal{A}$

$$\forall(\forall Y.t = t') \in E \cdot t^{\nu} = t'^{\nu}$$

If this holds for an equality $\forall Y.t = t'$, we write $\bar{\mathcal{A}} \models_{\Sigma} \forall Y.t = t'$.

This last definition restricts the definition of equality of unsorted terms to consider only *well-sorted* instantiations of variables. This well-behaved equality is used to extend the valid sorts in the algebra. We give a notion of a Sort-Judgement which takes into account the equations in deciding whether a term has a well-sorted denotation.

Definition 4.17. Given an order-sorted specification $\mathcal{S} = (\Sigma, E)$ built on the unsorted specification $\overline{\mathcal{S}} = (\overline{\Sigma}, E)$, then a *Sort Judgement* of sort S of a term $t \in \overline{T}_{\overline{\Sigma}}(\mathcal{X})$ in a $\mathcal{S}$-algebra, $\mathcal{A}$, on unsorted $\overline{\mathcal{S}}$-algebra, $\bar{\mathcal{A}}$, written $\mathcal{A} \models t :: S$, is defined by:

1. $\mathcal{A} \models t :: S$ if $t : S$ or
2. $\mathcal{A} \models t :: S$ if $\exists t'$ such that $\bar{\mathcal{A}} \models_{\Sigma} \forall Y.t = t'$ and $\mathcal{A} \models t' :: S$ where $Y = \mathit{Vars}(t) \cup \mathit{Vars}(t')$[2].

We can then extend Theorem 4.13 to characterise the sort of a term in the algebra.

Theorem 4.18. If $\mathcal{A} \models t :: S$, then for any $(\mathit{Vars}(t), \mathcal{A})$-assignment ν, $t^{\nu} \in \mathcal{A}_s$.

Proof (4.18). Assume $\mathcal{A} \models t :: S$. If $t : S$ then by Theorem 4.13, $t^{\nu} \in \mathcal{A}_s$. Otherwise $\bar{\mathcal{A}} \models_{\Sigma} \forall Y.t = t'$, so $t^{\nu} = t'^{\nu}$, and $\mathcal{A} \models t' :: S$. □

Thus an unsorted term has a well-sorted denotation if under any denotation, it is either a well-sorted term, or equal to a well-sorted term.

We now define the notion of equality: sorted algebras satisfy equations in particular sorts.

Definition 4.19. Equality in a Sort.
Given an order-sorted specification $\mathcal{S} = (\Sigma, E)$ built on the unsorted specification $\overline{\mathcal{S}} = (\overline{\Sigma}, E)$, and an order-sorted $\mathcal{S}$-algebra $\mathcal{A}$ on unsorted $\overline{\mathcal{S}}$-algebra, $\bar{\mathcal{A}}$, then two elements t_1, t_2 of $\mathcal{A}$ are *Equal in Sort* S, where $S \in S_{\Sigma}$, written as $\mathcal{A} \models \forall Y.t_1 =_s t_2$, if and only if:

1. either $\mathcal{A} \models t_1 :: S$ or $\mathcal{A} \models t_2 :: S$
2. $\bar{\mathcal{A}} \models_{\Sigma} \forall Y.t_1 = t_2$

Note that, by the definition of sort judgement, if either disjunct of the first condition holds, then so does the other.

This definition can be used to define the general equality in the order-sorted algebra.

[2]This will prevent sort assignment through empty sorts.

$\mathcal{A} \models \forall Y.t_1 = t_2$ if and only if $\exists S \in S_\Sigma \cdot \mathcal{A} \models \forall Y.t_1 =_S t_2$.

We can now define the notion of validity for an order-sorted specification in the normal fashion. Thus an equation $\forall Y.t = t'$ is *valid* under a specification $\mathcal{S}$, written $\mathcal{S} \models \forall Y.t = t'$, if $\forall Y.t = t'$ is valid in every model of $\mathcal{S}$.

Note that in this definition of a valid order-sorted algebra for a specification $\mathcal{S} = (\Sigma, E)$ we do not build upon an $\bar{\mathcal{S}}$-algebra, but a $\bar{\Sigma}$-algebra. Indeed, Example 4.15 shows that an $\bar{\mathcal{S}}$-algebra can be an unsuitable model upon which to construct the order-sorted model as it will satisfy too many equalities.

4.6 Dynamic Order-Sorted Equational Logic

This section defines Dynamic Order-Sorted Equational Logic and uses it to construct a canonical model for a specification, the quotient term algebra. This gives an inference system which respects the formulation of equality in sorts that we have defined above.

4.6.1 A Presentation of Dynamic Equational Logic

Definition 4.20. A set of equations E *derives* an equation $\forall Y \cdot u = t$ in a sort S, written $E \vdash_\Sigma \forall Y.u =_S t$, if either $\forall Y \cdot u = t \in E$ and either $u : S$ or $t : S$, or else $\forall Y \cdot u =_S t$ is derived using the rules in Figure 4.2. If the subscript S is omitted, then we assume that the equation holds in $\mathcal{LS}(u) \cup \mathcal{LS}(t)$.

We define the notion of a $\mathcal{W}$-substitution, used in the Instantiation rule.

Definition 4.21. Given a specification $\mathcal{S} = (\Sigma, E)$, a *$\mathcal{W}$-substitution* $\sigma : X \to \overline{T}_{\overline{\Sigma}}(\mathcal{X})$, is a mapping on a set of variables X such that if $x \in X$ and $s(x) = s$ then $\exists u \in T_\Sigma(\mathcal{X})$ such that $E \vdash_\Sigma \sigma x =_s u : s$.

Thus the definition of dynamic equational logic is recursive; it uses a definition of substi-

Reflexivity. $E \vdash_\Sigma \forall Y.t =_S t \quad \text{if } t : S$

Symmetry. $\dfrac{E \vdash_\Sigma \forall Y.u =_S t}{E \vdash_\Sigma \forall Y.t =_S u}$

Transitivity. $\dfrac{E \vdash_\Sigma \forall Y.u =_S t,\ E \vdash_\Sigma \forall Y.t =_T v}{E \vdash_\Sigma \forall Y.u =_S v}$

Congruence. $\dfrac{E \vdash_\Sigma \forall Y.u_1 =_{S_1} t_1 \cdots E \vdash_\Sigma, \forall Y.u_n =_{S_n} t_n}{E \vdash_\Sigma \forall Y.f(u_1, \ldots, u_n) =_S f(t_1, \ldots, t_n)}$ if $(W, S) \in ranks(f)$ and $\langle s_1, \ldots s_n \rangle \leq W$

Instantiation. $\dfrac{E \vdash_\Sigma \forall Y.u =_S t}{E \vdash_\Sigma \forall X.\sigma u =_S \sigma t}$ if $\sigma : Y \to T_\Sigma(X)$ is a $\mathcal{W}$-substitution.

Figure 4.2: Rules for Dynamic Order-Sorted Equational Logic

tution which uses the notion of derivation. Nevertheless, this definition is well-defined.

Lemma 4.22. $\vdash_\Sigma$ is well-defined. That is

$$=_E \ =_{def} \ \{(t, t') | \exists S \in S_\Sigma . E \vdash_\Sigma t =_S t'\}$$

is a well-defined relation.

Proof (4.22). For each sort $S \in S_\Sigma$, define the following set of relations

$$=_S^0 \ =_{def} \ \{t =_S t | t : S\} \cup \{t =_S t' | t = u \in E \wedge (t : S \vee u : S)\}$$

Thus $=_S^0$ are those equations immediately valid in S. Let:

$$\begin{aligned} =_S^{i+1} \ =_{def} \ =_S^0 \ &\cup \ \{t =_S u | u =_S^i t\} \\ &\cup \ \{u =_S v | u =_S^i t \wedge t =_T^i v\} \\ &\cup \ \{f(t_1, \ldots, t_n) =_S f(t_1, \ldots, t_n) | u_1 =_{S_1}^i t_1 \wedge \ldots \wedge u_n =_{S_n}^i t_n \wedge \\ &\qquad \exists (W, S) \in ranks(f) \cdot \langle S_1 \ldots S_n \rangle \leq W\} \\ &\cup \ \{\sigma u =_S \sigma t | u =_S^i t \wedge \forall x \in Dom(\sigma) \cdot \exists v \in T_{s(x)}(\mathcal{X}) \cdot \sigma x =_{s(x)}^i v\} \end{aligned}$$

Then let:

$$=_S \ =_{def} \ \bigcup_i =_S^i$$

$$=_E \quad =_{def} \quad \bigcup_{S \in S_\Sigma} =_S$$

As $=_E$ is the set of all finite derivations, then it is a well-defined relation. □

These rules thus define a relation, the *Congruence Generated in sort S by E.*

Definition 4.23. The *Congruence Relation Generated by E*, denoted as $s =_E t$, is the relation generated by taking the union of the congruences in all sorts:

$$=_E = \{(u,t) \mid \exists S \in S_\Sigma \cdot \forall Y \cdot u =_S t\}$$

Using $\mathcal{W}$-substitution in a recursive definition of equational congruence is an awkward way of defining this relation. However, we can replace this definition by a simpler one.

Σ-Instantiation. $\dfrac{E \vdash_\Sigma \forall Y.u =_S t}{E \vdash_\Sigma \forall X.\sigma u =_S \sigma t}$ if $\sigma : Y \to T_\Sigma(X)$ is a Σ-substitution.

Figure 4.3: Alternative Instantiation Rule

Lemma 4.24. The set of rules given by taking the set of rules in Figure 4.2 and replacing the Instantiation rule with the Σ-Instantiation in Figure 4.3 generates the same congruence.

Proof (4.24). If the new set of rules derives an equality in sort S, we denote this by $\forall Y.u =_S^\Sigma t$. Let $=_E^\Sigma = \{(u,t) \mid \exists S \in S_\Sigma \cdot \forall Y \cdot u =_S^\Sigma t\}$. Then clearly $=_E^\Sigma \subseteq =_E$ since every Σ-substitution is also a $\mathcal{W}$-substitution. We need to show that $=_E \subseteq =_E^\Sigma$. We prove this by considering the derivation trees.

If $\dfrac{\forall Y.v =_S u}{\forall X.\sigma v =_S \sigma u}$ where $\sigma : Y \to T(X)$ is a $\mathcal{W}$-substitution, then $\forall x \in Dom(\sigma)$, $\exists t \in T_{s(x)}(X)$, such that $\sigma x =_{s(x)} t$, then construct the substitution

$$\theta = \{x \mapsto t | x \in Dom(\sigma) \wedge \sigma x =_{s(x)} t \wedge t \in T_{s(x)}(X)\}$$

Then θ is a Σ-substitution, and therefore by Σ-Instantiation $\dfrac{\forall Y.v =_S u}{\forall X.\theta v =_S \theta u}$

As $\forall x \in Dom(\sigma), \exists t \in T_{s(x)}(\mathcal{X}) \cdot \sigma x =_{s(x)} t$, we can use the congruence rule multiple times to build the following proof:

$$\frac{\frac{\forall X.\sigma x_1 =_{s(x_1)} t_1, \cdots, \forall Y.\sigma x_n =_{s(x_n)} t_n}{\cdots\cdots}}{\forall X.\sigma v =_S \theta v}$$

and also a similar proof of $\forall X.\sigma u =_S \theta u$. Thus by further applications of the symmetry and transitivity rules, we can derive $\forall X.\sigma v =_S \sigma u$. □

Thus we can use Σ-substitution at the cost of making the derivation steps implicit in the definition of $\mathcal{W}$-substitution explicit. This proof is illustrated via the following example.

Example 4.25. Given a specification $\mathcal{S} = (\Sigma, E)$ as follows:

Sorts:	A, B, C	
Subsorts:	$A \leq C$, $B \leq C$	
Operators:	$a : \rightarrow A$	$b : \rightarrow B$
	$c : \rightarrow C$	$f : A \rightarrow C$
Equations:	$f(x : A) =_C c$	
	$a =_A b$	

then we can derive the equation $f(b) =_C c$ immediately from $f(x : A) =_C c$ using $\mathcal{W}$-substitution $\sigma = \{x \mapsto b\}$. However, it can be derived using only Σ-substitutions as follows:

$$\dfrac{\dfrac{\dfrac{a =_A b}{b =_A a} \qquad \dfrac{f(x : A) =_C c}{f(a) =_C c}\ \Sigma\text{-substitution } \theta = \{x \mapsto a\}}{f(b) =_C f(a)}}{f(b) =_C c}$$

□

The following lemma shows that this congruence has the property that equality of terms

in a sort corresponds to the equality of those terms with a well-sorted Σ-term. Thus this congruence shares this important property with the semantic definition of equality above.

Lemma 4.26. If $E \vdash_\Sigma \forall Y.t =_S u$ then $\exists t' : S$ such that $E \vdash_\Sigma \forall Y.t' =_S t$ and $E \vdash_\Sigma \forall Y.t' =_S u$.

Proof (4.26). By induction on the length of derivation.

Base Case: Reflexivity: by definition $E \vdash_\Sigma \forall Y.t =_S t$ if $t : S$. Thus let $t' = t$.

Base Case: Direct Equality: If $\forall Y.t = u \in E$ then $E \vdash_\Sigma t =_{\mathcal{LS}(t)} u$, so let $t' = t$.

Induction Steps. Symmetry. $E \vdash_\Sigma t =_S u$ and for some $t' : s$, $E \vdash_\Sigma t' =_S t$. Thus $t' : S$, $E \vdash_\Sigma t' =_S t$ for $E \vdash_\Sigma u =_S t$. Similarly for u.

Transitivity. If $E \vdash_\Sigma \forall Y.u =_S t$, $E \vdash_\Sigma \forall Y.t =_T v$ and $E \vdash_\Sigma \forall Y.u =_S t'$ for some $t' : S$, then by application of symmetry and transitivity with this inference $E \vdash_\Sigma \forall Y.v =_S t'$. Thus the proposition holds for $E \vdash_\Sigma \forall Y.u =_S v$.

Congruence. Assume that $\forall i = 1 \ldots n$ $E \vdash_\Sigma \forall Y.t_i =_{S_i} u_i$ and $\exists t_i'.t_i' : S_i \wedge E \vdash_\Sigma \forall Y.t_i =_{S_i} t_i'$, $(W, S) \in \mathit{ranks}(f)$ and $\langle S_1 \ldots S_n \rangle \leq W$. Then by the congruence rule applied twice: $E \vdash_\Sigma \forall Y.f(t_1', \ldots, t_n') =_S f(t_1, \ldots, t_n) =_S f(u_1, \ldots, u_n)$, and by definition of $T_\Sigma(\mathcal{X})$ $f(t_1', \ldots, t_n') : S$.

Instantiation. By the above lemma, we can use a Σ-substitution, $\sigma : Y \rightarrow T_\Sigma(X)$. Hence if $E \vdash_\Sigma \forall Y.t =_S u$ and $E \vdash_\Sigma \forall Y.t =_S t'$ for some $t' : S$, consider $E \vdash_\Sigma \forall X.\sigma t =_S \sigma u$. Also $E \vdash_\Sigma \forall X.\sigma t =_S \sigma t'$ and as $\forall x \in \mathit{Dom}(\sigma).\sigma x : s(x)$, then by definition of $T_\Sigma(\mathcal{X})$, $\sigma t' : S$. The proof is trivially completed via transitivity. □

If an equation can be derived in a sort, then it can be derived in any supersort, so the Sort-Weakening Rule of Figure 4.4 is unnecessary.

Lemma 4.27. The rules of dynamic equational logic imply the rule given in Figure 4.4.

Proof (4.27). By Lemma 4.26, if $E \vdash_\Sigma \forall Y.t =_S t'$ then $\exists u : S$ such that $E \vdash_\Sigma \forall Y.t' =_S u$.

Sort-Weakening. $\dfrac{E \vdash_\Sigma \forall Y.t =_S t'}{E \vdash_\Sigma \forall Y.t =_T t'}$ if $S \leq_\Sigma T$.

Figure 4.4: The Sort-Weakening Rule

If $u : S$ then $u : T$, as $S \leq_\Sigma T$. Hence by transitivity:

$$\dfrac{E \vdash_\Sigma \forall Y.t =_S t' \qquad \dfrac{E \vdash_\Sigma \forall Y.t' =_S u,\ E \vdash_\Sigma \forall Y.u =_T u}{E \vdash_\Sigma \forall Y.t' =_T u}}{E \vdash_\Sigma \forall Y.t =_T t'}$$

□

Under these laws a set of equations E presents a *dynamically sorted equational theory.*

Definition 4.28. For a given term t the *congruence class* of t under an equational theory E is the set of terms which are equal to t in that theory: $[t]_E = \{t' | \forall Y.t =_E t'\}$.

We can then show the following theorem, that the model theory and the proof theory correspond to each other.

Theorem 4.29. The rules for dynamic order-sorted equational logic are sound and complete. That is:

1. **Soundness.** $E \vdash_\Sigma \forall Y.u =_E t \Rightarrow \mathcal{S} \models \forall Y.u = t$
2. **Completeness.** $\mathcal{S} \models \forall Y.u = t \Rightarrow E \vdash_\Sigma \forall Y.u =_E t$

The proofs of these two theorems are given in the following sections.

4.6.2 Soundness

Rather than prove the statement of soundness as it stands, we prove a stronger theorem.

Theorem 4.30. Given an Order-Sorted specification $\mathcal{S} = (\Sigma, E)$:

$$E \vdash_\Sigma \forall Y.u =_S t \Rightarrow \mathcal{S} \models \forall Y.u =_S t$$

Clearly, this proposition implies the soundness of deduction. We prove this by induction on the length of the derivation of $u =_S t$. $\mathcal{A}$ is any $\mathcal{S}$-algebra built on unsorted algebra $\bar{\mathcal{A}}$XS

Proof (4.30).

Base Cases. If $\forall Y.u = t \in E$ then by Instantiation with the empty substitution, $E \vdash_\Sigma \forall Y.u =_S t$ for either $u : S$ or $t : S$. By definition of $\mathcal{S}$-algebra $\mathcal{A} \models \forall Y.u = t$. So $\bar{\mathcal{A}} \models_\Sigma \forall Y.u = t$. As either $u : S$ or $t : S$, $\mathcal{A} \models u :: S$ or $\mathcal{A} \models t :: S$. Hence $\mathcal{A} \models \forall Y.u =_S t$.

Reflexivity. If $E \vdash_\Sigma \forall Y.t =_S t$ for some sort S, then $t : S$ from the order-sorted signature. Obviously, for any sorted assignment of variables to $\bar{\mathcal{A}}$, $\nu : Y \to \bar{\mathcal{A}}$, $t^\nu = t^\nu$ and so $\bar{\mathcal{A}} \models_\Sigma t = t$. And since $t : S$ then $\mathcal{A} \models t :: S$. Hence $\mathcal{A} \models \forall Y.t =_S t$.

Induction Steps.

Symmetry. If $\mathcal{A} \models \forall Y.u =_S t$, then $\bar{\mathcal{A}} \models_\Sigma \forall Y.u = t$ and either $\mathcal{A} \models u :: S$ or $\mathcal{A} \models t :: S$. Thus for any for any sorted assignment of variables to $\bar{\mathcal{A}}$, $\nu : Y \to \bar{\mathcal{A}}$, $t^\nu = u^\nu$ and $\bar{\mathcal{A}} \models_\Sigma \forall Y.t = u$ and so $\mathcal{A} \models \forall Y.t =_S u$.

Transitivity. If $\mathcal{A} \models \forall Y.u =_S t$, and $\mathcal{A} \models \forall Y.t =_T v$, then $\bar{\mathcal{A}} \models_\Sigma \forall Y.u = t$ and $\bar{\mathcal{A}} \models_\Sigma \forall Y.t = v$. So for any sorted assignment $\nu : Y \to \bar{\mathcal{A}}$, $u^\nu = t^\nu = v^\nu$ and so $\bar{\mathcal{A}} \models_\Sigma u = v$. Also $\mathcal{A} \models u :: S$, hence $\mathcal{A} \models \forall Y.u =_S v$.

Congruence. Assume $\mathcal{A} \models \forall Y.u_i =_{S_i} t_i$. Then for all sorted assignments $\nu : Y \to \bar{\mathcal{A}}$, $u_i^\nu = t_i^\nu$. Consider $f(u_1, \ldots, u_n)$.

$$\begin{aligned} f(u_1, \ldots, u_n)^\nu &= f^{\bar{\mathcal{A}}}(u_1^\nu, \ldots, u_n^\nu) \\ &= f^{\bar{\mathcal{A}}}(t_1^\nu, \ldots, t_n^\nu) \\ &= f(t_1, \ldots, t_n)^\nu \end{aligned}$$

Hence $\bar{\mathcal{A}} \models_\Sigma \forall Y.f(u_1, \ldots, u_n) = f(t_1, \ldots, t_n)$.

Further, $\mathcal{A} \models u_i :: S_i$. Hence there exist terms u_i' such that $\bar{\mathcal{A}} \models_\Sigma \forall Y.u_i = u_i'$ and

$u'_i : S_i$. By a similar argument to above $\bar{A} \models_\Sigma \forall Y.f(u_1,\ldots,u_n) = f(u'_1,\ldots,u'_n)$ and since $(W,S) \in \text{ranks}(f)$ and $\langle S_1,\ldots S_n\rangle \leq W$, $f(u'_1,\ldots,u'_n)$ is a well-sorted term of sort S. Hence $\mathcal{A} \models f(u_1,\ldots,u_n) :: S$ and thus $\mathcal{A} \models \forall Y.f(u_1,\ldots,u_n) =_S f(t_1,\ldots,t_n)$.

Instantiation. If $\mathcal{A} \models \forall Y.u =_S t$ for $u : S$, then let X be an arbitrary set of sorted variables. So for any well sorted substitution $\sigma : Y \to \mathcal{T}_\Sigma(X)$ consider a sorted assignment to a $\mathcal{S}$-algebra $\mathcal{A}$, $\mu : X \to \bar{A}$. Thus $\mu \circ \sigma : Y \to \bar{A}$ is a sorted assignment to Y. Since $\mathcal{A} \models \forall Y.u = t$, then $u^{\mu\circ\sigma} = t^{\mu\circ\sigma}$ and thus $(\sigma u)^\mu = (\sigma t)^\mu$ and so $\bar{A} \models_\Sigma \forall Y.u = t$. Clearly, as σ is well-sorted, if $u : S$, then $\sigma u : S$ by construction of sorted terms, and thus $\mathcal{A} \models \sigma u :: S$. So $\mathcal{A} \models \forall X.\sigma u =_S \sigma t$. □

4.6.3 Quotient Algebra

The free term algebra no longer forms a model for the specification. However, we can carry out a construction on the free term algebra to generate the quotient term algebra for a specification, using the notion of congruence as defined above. Intuitively in this construction we identify all those terms which are proven to be equal using the logic. This is regardless of the well-sortedness of terms as terms will appear in the congruence classes if they can be proven equal, even if they are not *syntactically* of that sort.

Definition 4.31. Given an order-sorted specification $\mathcal{S} = (\Sigma, E)$, on unsorted specification $\overline{\mathcal{S}} = (\overline{\Sigma}, E)$ the *Quotient Term Algebra* over a sorted set of variables $\mathcal{X}$, written $\mathcal{T}_{\mathcal{S},\mathcal{X}}$, is given by the following construction.

We first generate the following $\overline{\Sigma}$-algebra, $\bar{T}_{\overline{\mathcal{S}},\mathcal{X}}$ where:

1. $\bar{T}_{\overline{\mathcal{S}},\mathcal{X}} = \{[t]_E \cup \{t\} | t \in \overline{T}_{\overline{\Sigma}}(\mathcal{X})\}$
2. $f^{\bar{T}_{\mathcal{S},\mathcal{X}}}([t_1]_E \cup \{t_1\},\ldots,[t_n]_E \cup \{t_n\}) = [f(t_1,\ldots,t_n)]_E \cup \{f(t_1,\ldots,t_n)\}$

In this algebra, the identity element is used solely if the set $[t]_E$ is empty. Note that according to the proof rules, ill-sorted terms which are not shown to be equal to any well-sorted term have empty equivalence classes. We cannot even show the identity of such

terms. However, for terms $t : S$ which are well-sorted, $[t]_E$ is guaranteed not to be empty. We can then build the following quotient algebra on top of this unsorted algebra:

1. $S^{\mathcal{T}_{\mathcal{S},\mathcal{X}}} = \{[t]_E | t \in \overline{T}_{\overline{\Sigma}}(\mathcal{X}) \text{ and } t : s\}$ for all $S \in S_\Sigma$;
2. $f^{\mathcal{T}_{\mathcal{S},\mathcal{X}}}([t_1]_E, \ldots, [t_n]_E) = [f(t_1, \ldots, t_n)]_E$.

Thus we have denotations for terms which are unsorted but equal in some sort to a well-sorted term and these denotations are in the appropriate sort.

We can now return to the proof of completeness.

Theorem 4.32. Given an Order-Sorted specification $\mathcal{S} = (\Sigma, E)$:

$$\mathcal{S} \models \forall Y.u = t \Rightarrow E \vdash_\Sigma \forall Y.u =_E t$$

Proof (4.32). Completeness.
If for every $\mathcal{S}$-algebra $\mathcal{A}$ we have $\mathcal{A} \models \forall Y.s = t$ then in particular, the quotient term algebra over $\mathcal{X}$, $\mathcal{T}_{\mathcal{S},\mathcal{X}} \models \forall Y.s = t$. However, by the definition of the latter, $\forall Y.s = t$ holds if and only if there is a sort S such that s and t are in the same equivalence class. The equivalence classes are determined by the inference system, and thus the equality holds only if $E \vdash_\Sigma \forall Y.s =_S t$. Hence the equational system is complete. □

Similarly to the free term algebra for the signature (that is a specification with an empty set of equations), the quotient term algebra has the following property.

Theorem 4.33. Given a specification $\mathcal{S} = (\Sigma, E)$, and an $\mathcal{S}$-algebra $\mathcal{A}$, and a $(\mathcal{V}, \mathcal{A})$-assignment ν over a set of variables $\mathcal{V}$, the corresponding denotation function $\_^\nu : \mathcal{T}_{\mathcal{S},\mathcal{V}} \to \mathcal{A}$ is a homomorphism.

As a corollary, the ground quotient term algebra $\mathcal{T}_{\mathcal{S},\emptyset}$ is the initial model of the category $\mathbf{OSAlg}_\mathcal{S}$ (up to isomorphism), and is thus the initial algebra of the specification. The initial algebra can informally be characterised as having 'no junk' and 'no confusion'.

- *No Junk.* Every element in the algebra is a denotation of some ground term. Extra

elements in the algebra which do not denote some ground term are 'junk'.

- *No Confusion.* Two elements are identified in the algebra if and only if they are provably equal using the equational logic. Elements which are identified, but not provably so are 'confused'.

Example 4.34. We return to Example 3.1. The dynamic quotient term algebra is:

$$\begin{aligned} B^{\mathcal{T}_{S,\emptyset}} &= \{[b]\} \\ &= \{\{b,c\}\} \\ C^{\mathcal{T}_{S,\emptyset}} &= \{[c]\} \\ &= \{\{b,c\}\} \\ A^{\mathcal{T}_{S,\emptyset}} &= \{[b],[c],[f(b)]\} \\ &= \{\{b,c\},\{f(b),f(c)\}\} \end{aligned}$$

and we can use the new inference rules to perform the derivation directly.

$$\frac{b =_B c}{f(b) =_A f(c)}$$

So now we have the expected result $f(b) = f(c)$ as a sound inference in our system. □

4.7 Terms and Sorts

In this section, some terminology and results on terms are re-examined in the light of the revised order-sorted semantics. We assume a specification $\mathcal{S} = (\Sigma, E)$ with $\Sigma = (S_\Sigma, \leq_\Sigma, \mathcal{F}_\Sigma)$, and sorted variables $\mathcal{X} = \biguplus_{s \in S_\Sigma} \mathcal{X}_s$. If $x \in \mathcal{X}_s$, then $s(x) = s$. $\overline{\Sigma}$ and $\overline{\mathcal{S}}$ represent the unsorted components of the signature and specification respectively. The definitions of *Unsorted Terms* $\overline{T}_{\overline{\Sigma}}(\mathcal{X})$ and *Sorted Terms*, $T_s(\mathcal{X})$ are as in Definitions 4.2 and 4.4. For any term, we define the *Least Sorts.*

Definition 4.35. Least Sort.
Given a $\overline{T}_{\overline{\Sigma}}(\mathcal{X})$ term, t, then the *least sorts* $\mathcal{LS}(t) \subseteq S_\Sigma$ is the smallest set such that:

1. $\forall s \in \mathcal{LS}(t) \cdot t : s$
2. $\forall s \cdot t : s \Rightarrow \exists s' \in \mathcal{LS}(t) \cdot s' \leq s$

The set of least sorts is thus the smallest set of sorts which can be determined for a term by static sort-checking alone. In any model $\mathcal{A}$ of Σ, the denotation of a term t is an element of every set $\{A_s | s \in \mathcal{LS}(t)\}$ and thus is an element of $\bigcap\{A_s | s \in \mathcal{LS}(t)\}$. If the least sort of a term is unique, we refer to it as the least sort and write $\mathcal{LS}(t) = s$. For some terms the set of least sorts will be empty.

Definition 4.36. Under-sorted Term.
The set of under-sorted terms, or U-Terms, $U_\Sigma(\mathcal{X})$ is given by:

$$U_\Sigma(\mathcal{X}) = \overline{T}_{\overline{\Sigma}}(\mathcal{X}) - T_\Sigma(\mathcal{X})$$

U_Σ is the set of ground under-sorted terms.

Lemma 4.37. If $t \in U_\Sigma(\mathcal{X})$, then $\mathcal{LS}(t) = \emptyset$.

Proof (4.37). Obvious from the definition. □

Thus Σ-Terms are sorted via static sort-checking. However, in the new semantics, an equality holding in a sort was defined for both model and proof theory. Definition 4.23 says that an equality will hold in sort s if its terms are equal to a term in $T_s(\mathcal{X})$. This 'equational' point of view of determining the sorts of equations can be shifted to one based on determining which terms are in a sort. If a term is equal to another in a sort, then it must be denoted by an element of the model of that sort and can be thought of as having that sort. Thus there is more sort information which can be determined about terms by considering the specification. This information cannot be derived statically, but only through a process of equational deduction, that is to say a 'dynamic', or 'runtime' sort-checking process. These are the *semantic sorts* of terms: semantic since in any valid model of the specification denotations of terms will be elements of the denotations of these sorts, as opposed to the syntactic sorts which can be statically determined.

Definition 4.38. Semantic Sort.

Given $t \in \overline{T}_{\overline{\Sigma}}(\mathcal{X})$, the set of semantic sorts for t is the set $S(t) = \{s \mid \exists u \in \overline{T}_{\overline{\Sigma}}(\mathcal{X}).t =_s u\}$. If $s \in S(t)$, then we write $t :: s$.[3]

The set of least semantic sorts of t, written $\mathcal{SS}(t)$, is the minimal set of semantic sorts of t. That is $s \in \mathcal{SS}(t)$ if $t :: s$ and $\forall s' \in S(t) \cdot \exists s \in \mathcal{SS}(t) \cdot s \leq s'$.

It is trivial to prove the following.

Lemma 4.39. $\forall s \in \mathcal{LS}(t) \cdot \exists s' \in \mathcal{SS}(t)$ such that $s' \leq s$.

In any model of the specification the denotation of a term will be an element of the denotation of the least semantic sorts of that term. In any model $\mathcal{A}$ of the signature Σ the denotation of a term t will be an element of all the sets $\{A_s | s \in \mathcal{SS}(t)\}$ and thus will be an element of $\bigcap\{A_s | s \in \mathcal{SS}(t)\}$. Further, we cannot determine from the specification whether $t^{\mathcal{A}}$ will be in the denotation of any subsort of any $s \in \mathcal{SS}(t)$. This is a strong statement, and it is in general undecidable to show $t :: s$ for any arbitrary specification, s and t.

Theorem 4.40. Given an arbitrary specification $\mathcal{S}$, it is undecidable whether $t :: s$ for an arbitrary term t and sort s.

Proof (4.40). We demonstrate that there is a class of specifications for which in general this property is undecidable. A Turing Machine [RS86] is defined as a 6-tuple $(Q, \Sigma, T, P, q_0, F)$ where

1. Q is a finite set of states,
2. Σ is a finite set of symbols including the blank symbol $\wedge$,
3. $T \subseteq \Sigma - \{\wedge\}$ is a set of input symbols,
4. P is a program, a partial function:
$$P : (Q - F) \times \Sigma \rightharpoonup Q \times \Sigma \times \{L, R, 0\}$$
5. $q_0 \in Q$ is the initial state,
6. $F \subseteq Q$ is a set of final states.

[3] In our semantic presentation we use the same notation for a sort judgement: $\mathcal{A} \models t :: s$ if and only if $t^{\mathcal{A}} \in s^{\mathcal{A}}$. As this is the concept captured by semantic sort, there is no confusion in notation.

Any Turing machine can be encoded as an order-sorted specification using the following construction. We take a specification of the structure of the Turing machine as in Figure 4.5. Then we augment this specification with rules as follows.

Sorts:	$\Sigma\ Tape\ Q\ F$
Subsorts:	$\Sigma \leq Tape,\ F \leq Q$
Operators:	$a_1, \ldots, a_k, \wedge : \rightarrow \Sigma$
	$q_1, \ldots, q_n : \rightarrow Q$
	$f_1, \ldots, f_m : \rightarrow F$
	$\epsilon : \rightarrow Tape$
	$\_.\_ : Tape\ Tape \rightarrow Tape$
	$\_[\_]\_ : Tape\ Q\ Tape \rightarrow Tape$
Variables:	$t, t_1, t_2, t_3, l, r : Tape,$
	$s : \Sigma$
	$q : Q$
	$f : F$
Equations:	$\epsilon.t = t$
	$t.\epsilon = t$
	$(t_1.t_2).t_3 = t_1.(t_2.t_3)$
	$l[q]\epsilon = l[q](\wedge.\epsilon)$

Figure 4.5: Specification of a Turing Machine as an Order-Sorted Algebra

For each pair in $(Q - F) \times \Sigma$, q_i, s_j say, if $P(q_i, s_j) = (q_k, s_l, R)$ then add the rule:

$$l[q_i]s_j.r \rightarrow l.s_l[q_k]r$$

if $P(q_i, s_j) = (q_k, s_l, L)$ then add the rules:

$$l.s[q_i]s_j.r \rightarrow l[q_k]s.s_l.r$$

$$\epsilon[q_i]s_j.r \rightarrow \epsilon[q_k] \wedge .s_l.r$$

if $P(q_i, s_j) = (q_k, s_l, 0)$ then add the rule:

$$l[q_i]s_j.r \rightarrow l[q_k]s_l.r$$

In this presentation we also add the output rule:

$$l[f]s.r \rightarrow s$$

Thus the tape is split into two by the current position, marked with the current state, the first symbol on the right being the current symbol. Clearly, for any l, q, r, $l[q]r : Tape$. Assume it is decidable whether $l[q_0]r :: \Sigma$ for any l, r then we can show that $l[q_0]r =_\Sigma a_i$ for some $a_i : \Sigma$. That is, there is some f_i such that $l[q_0]r =_\Sigma l'[f_i]a_i.r'$. Thus, the rules reduce the initial state to a final state. However, this provides a decision procedure for the halting problem for Turing Machines which is well-known to be undecidable, so our assumption that we can decide whether $l[q_0]r :: \Sigma$ is incorrect, and the proposition is shown. □

Using this definition we extend the concept of well-sorted terms beyond those which are sorted through syntax alone.

Definition 4.41. Well-Sorted Terms

The set of well-sorted terms (W-Terms) are defined to be:

$$W_{\mathcal{S}}(\mathcal{X}) = \{t | \mathcal{SS}(t) \neq \emptyset\}$$

The well-sorted terms of sort s are given by:

$$\begin{aligned} W_s(\mathcal{X}) &= \{t | \exists u.t =_s u\} \\ &= \{t | \exists s' \in \mathcal{SS}(t).s' \leq s\} \\ &= \{t | t :: s\} \end{aligned}$$

and thus

$$W_{\mathcal{S}}(\mathcal{X}) = \bigcup_{s \in S_\Sigma} W_s(\mathcal{X})$$

$W_{\mathcal{S}}$ is the set of ground well-sorted terms.

Lemma 4.42. $T_\Sigma(\mathcal{X}) \subseteq W_{\mathcal{S}}(\mathcal{X})$.

Definition 4.43. Ill-sorted Terms (I-Terms).

$$I_{\mathcal{S}}(\mathcal{X}) = \overline{T}_{\overline{\Sigma}}(\mathcal{X}) - W_{\mathcal{S}}(\mathcal{X})$$

$I_{\mathcal{S}}$ is the set of ground ill-sorted terms.

The following properties are obvious.

Lemma 4.44.

1. $\forall t \in I_{\mathcal{S}}(\mathcal{X}) \cdot \mathcal{SS}(t) = \emptyset$
2. $I_{\mathcal{S}}(\mathcal{X}) \subseteq \overline{T}_{\overline{\Sigma}}(\mathcal{X})$.

The relationships between these various sets of terms are succinctly expressed in Figure 4.6. As the semantic sorts are undecidable, whether a $\bar{\Sigma}$-term is an I-term or a W-term is in general undecidable.

$$\begin{array}{rcccc} & & U_{\Sigma}(\mathcal{X}) & \uplus & T_{\Sigma}(\mathcal{X}) \\ \overline{T}_{\overline{\Sigma}}(\mathcal{X}) & = & \cup\mid & & \mid\cap \\ & & I_{\mathcal{S}}(\mathcal{X}) & \uplus & W_{\mathcal{S}}(\mathcal{X}) \end{array}$$

Figure 4.6: Relationships between Sets of Terms.

Example 4.45. Given the following order-sorted specification:

Sorts:	$A\ B\ C$	
Subsorts:	$A \leq B, C \leq B$	
Operators:	$a : \rightarrow A$	$b : \rightarrow B$
	$c : \rightarrow C$	$f : A \rightarrow A$
Equations:	$a = b$	
	$f(x : A) = x$	

Then we can enumerate the following sets of ground terms:

$$\begin{aligned} \overline{T}_{\overline{\Sigma}} &= \{a, b, c, f^n(a), f^n(b), f^n(c) | n \in I\!N\} \\ T_{\Sigma} &= \{a, b, c, f^n(a) | n \in I\!N\} \\ U_{\Sigma} &= \{f^n(b), f^n(c) | n \in I\!N\} \\ W_{\mathcal{S}} &= \{a, b, c, f^n(a), f^n(b) | n \in I\!N\} \\ I_{\mathcal{S}} &= \{f^n(c) | n \in I\!N\} \end{aligned}$$

□

4.8 Alternative Approaches

This approach to order-sorted algebras can be compared to several others given in the literature. An interesting approach to semantics which has yet to be explored fully is given by Poigné [Poi87]. Also Equational Typed Logic [MSS90], Unified Algebra [Mos89a, Mos89b, Mos89c] and Galactic Algebra [Még92] are general approaches to universal algebra which can be specialised to order-sorted algebra in a similar way to that presented here; we only consider Equational Typed Logic in detail, and also a slightly different presentation of the two-tier semantics.

4.8.1 Using Sort Top

An alternative to the two-tier algebra is to augment signatures with sort $\top$ ('Top'), as defined in Section 3.2. Then all terms are well-sorted and the standard (non-overloaded) semantics for the free order-sorted algebra as given in Section 2.3 can be used. However, augmenting with sort $\top$ allows too many 'ill-sorted' terms, without distinguishing those which have valid denotations. Thus the notion of "equality within a sort" is still required, as equations in the sort $\top$ must still be regarded as invalid, so the semantics of specifications would have to be modified, in a similar fashion to the two-tier semantics.

4.8.2 Equational Typed Logic

Equational Typed Logic [MSS89a, MSS89b, MSS89c, MSS90] provides an alternative approach to the semantics of order-sorted specification. Equational Typed Logic (ETL) is an algebraic specification method which is designed to tackle the problems of specifying partiality, type polymorphism, higher-order and dependent types, by regarding sorts or types as first-class objects. An ETL presentation is a single-sorted algebraic specification, with operators for elements, and types, thus in any algebra, there are *elements* rather than sets representing types. Any algebra $\mathcal{A}$ also provides a *typing relation* '$:_{\mathcal{A}}$' between terms, and the presentation includes conditional typing assertions and equations, which can have

typing assertions as conditions. The rules of ETL include rules for type deductions. Thus for example, the specification of a stack is given as follows:

Example 4.46. Stacks can be specified in ETL as follows:

spec STACK
var s, i
in $empty : stack$
$s : stack,\ i : item \rightarrow push(s,i) : stack$
$s : stack,\ i : item \rightarrow pop(push(s,i)) = s$
$s : stack,\ i : item \rightarrow top(push(s,i)) = i$
endspec

The first two axioms are typing rules, the second of which has the types of variables as conditions. The third and fourth axioms give the behaviour of *top* and *pop*, given correctly-typed variables. Although *pop(empty)* is a well-formed term in ETL, there is no term which can be proved to be in the typing relation with it, capturing the partiality of *pop*. □

Clearly, ETL has a close relation with the semantics of dynamic order-sorted algebras: all syntactically well-formed terms are available, the 'meaningful' ones distinguished via a typing assignment, similar to a sort judgement. The similarity of the intentions of the approaches are exposed in the following comment ([MSS89b] p.2):

> ... *not every term has to be meaningful: but for a given presentation, a term can be considered meaningful if and only if it* proves *so in the context of that presentation; more precisely, meaningful terms are all and only those which appear in some formula that is* derivable *from the presentation.*

Indeed, a representation of order-sorted logic in ETL is given in [MSS90], with a systematic translation from an order-sorted specification into an ETL presentation.

Definition 4.47. Given an order-sorted specification $\mathcal{S} = (\Sigma, E)$, its ETL presentation consists of the following ETL formulas:

1. for each $s \leq s' \in \Sigma$, add $y : s \rightarrow y : s'$;

2. for each $f : s_1, \ldots, s_n \rightarrow s \in \Sigma$, add $y_1 : s_1, \ldots, y_n : s_n \rightarrow f(y_1, \ldots, y_n) : s$;
3. for each $\forall X \cdot e \in E$, where $X = \{x_1 : s_1, \ldots, x_n : s_n\}$, add $x_1 : s_1, \ldots, x_n : s_n \rightarrow e$.

This translation is shown to be sound and complete. However, it is not a sound translation of the standard semantics. As pointed out in [MSS90], p.34 (report version):

> ... *it is not clear to us to what extent Smolka's type logic is complete: its completeness is stated in* [e.g. as in [SNGM88]] *for well-typed specifications only, whilst ET logic is complete* tout-court.

Thus equations which are not provable in the standard logic since they contain terms which are not well-formed (such as $f(a) = f(c)$ in Example 3.1) are derivable in ETL. Such equations are derivable in dynamic order-sorted logic, and this translation reflects the semantics presented in this chapter rather than the standard semantics.

ETL is more general and powerful than order-sorted logic, since complex conditions on the types, such as higher-order and dependent types, can be defined. Nevertheless, there is a value in the order-sorted semantics given in this chapter. The two-tier semantics captures exactly the intuition of order-sorted algebra and no more. For instance, there are too many terms available in the ETL presentations since there are terms formed by operators acting on sorts. The dynamic order-sorted logic is also a much more concise description of the logic, providing the useful notion of sorted equations. Further, as we shall see, the computational counterpart of the dynamic logic is a simpler rewriting system than the restricted conditional rewriting supported by ETL [MSS90].

4.9 Conclusions

The standard semantics for order-sorted equational logic prove too restrictive to allow many desired specifications to be written, and natural inferences prove to be unsound. In this chapter, the standard semantics have been reformulated to free the logic from the syntax and to accommodate natural inferences from the equations. This results in a two-tier algebra with an order-sorted algebra embedded within an unsorted one. This

extra layer allows unsorted terms to be identified with well-sorted ones. From this two-tier semantics a simple sound and complete deduction system has been formulated for order-sorted equational logic, and this system captures the intuition of equational reasoning.

Dynamic sorting precludes the use of static type-checking to detect contextual errors by analysing the syntax of terms and checking they are used in the right context, as determined by sort. If the sorts of terms are dependent upon equational consequences, then type-checking becomes a process of theorem-proving and is undecidable in general. This is not a problem as the dynamic sorting method is regarded as primarily a specification technique, and so freedom is preferred over a strict sorting discipline. As specifications are transformed into programs, the type-discipline should become stronger to allow static type-checking. The order-sorted rewriting based-programming language OBJ-3 [GW88] also forgoes static type checking since it allows multiple parses of terms, so each term may have more than one sort, and uses retracts to enable the computation of ill-sorted results. If the final result of a rewriting sequence in OBJ-3 still has retracts, then there must be a type error. This is similar to the use of the unsorted terms in dynamic order-sorted equational logic.

Chapter 5

Dynamic Order-Sorted Terms

5.1 Introducing Dynamic Terms

In previous chapters we have discussed some of the problems associated with the standard presentations of order-sorted algebras and logics, and also explored a semantics and equational logic which overcome these difficulties. This dynamic equational logic introduces a concept of a sorted equation, whereby equations can be shown to hold in sorts. By extension, extra sort information can be generated for terms over and above that which can be deduced from their syntactic structure alone.

However, as with standard presentations of equational logics, this logical system is too undirected to be used practically in an automated theorem-proving system. It is therefore desirable to develop a computational analogue to the dynamic equational logic which can be used for automated proof in the logic, that is a dynamic rewriting relation which corresponds to the dynamic equational logic.

To perform rewriting respecting the dynamic equational logic it is necessary to use the additional sort information of a term operationally to decide whether a rewrite step is valid. This leads to a problem: the semantic sort information deduced about a term needs

to be known, but this sort information is lost. We need a mechanism for recording the sort information deduced concerning a term. Therefore this chapter introduces the concept of *dynamic terms*, which are terms tagged with sorts. As sort information is deduced concerning the sort of the term, the sort tag is updated dynamically[1]. Dynamic terms are given semantics by relating them to the terms in the underlying (unsorted) term algebra.

5.2 Dynamic Terms

This section gives the basic definitions and properties of dynamic terms.

5.2.1 Sorts and Minimal Sets of Sorts

We define an order-sorted signature as in Chapter 4. We extend the notion of ordering on sorts to sets of sorts, which are regarded as the *intersection* set of sorts. That is in any model $\mathcal{A}$, $\{A_{s_1}, A_{s_2}, \ldots, A_{s_n}\} =_{def} A_{s_1} \cap A_{s_2} \cap \ldots \cap A_{s_n}$.

Definition 5.1. If $S, S' \subseteq S_\Sigma$ then $S \leq S'$ if and only if $\forall s' \in S' \; \exists s \in S \cdot s \leq s'$.

It can thus be decided whether a set of sorts is 'less than' another. Note that this means that if $S_1 \subseteq S_2$ then $S_2 \leq S_1$. However, the motivation is when a term has a particular set of sorts, then it is an element of all of these sorts, and thus an element of their intersection, and the ordering relation is defined to reflect this.

The partial order on sorts forms a quasi-order when extended to sets of sorts. This can be seen as if $S_1 \leq S_2$ then $S_1 \cup S_2 \leq S_1$ and $S_1 \leq S_1 \cup S_2$ but it is not the case that $S_1 \cup S_2 = S_1$. Applying the minimisation function $M_\Sigma : \wp S_\Sigma \rightarrow \wp S_\Sigma$ to sets of sorts recovers

[1] This process has an analogy with dynamic binding in programming languages, notably Lisp and Object-Oriented languages, where the operation applied to an object is not determined until runtime as the type of the object is not necessarily known before that point. Unfortunately this makes static type checking incomplete, which is also the case with dynamic terms.

the partial-order.

$$M_\Sigma(S) = \{s | s \in S \wedge \neg\exists s' \in (S - \{s\}) \cdot s' \leq s\}$$

The following lemma is trivial.

Lemma 5.2. If $S_1 \leq S_2$ then $M_\Sigma(S_1 \cup S_2) = M_\Sigma(S_1)$

Thus $\leq$ forms a partial-order on minimised sets. The following lemmas are also useful.

Lemma 5.3.

1. $M_\Sigma(M_\Sigma(S) \cup S_1) = M_\Sigma(S \cup S_1)$
2. $S \leq M_\Sigma(S_1 \cup S_2)$ if and only if $S \leq S_1$ *and* $S \leq S_2$
3. $M_\Sigma(S_1 \cup S_2) \leq S$ if and only if $S_1 \leq S$ or $S_2 \leq S$

When $S \leq \{s\}$, i.e. a singleton set, we write $S \leq s$.

5.2.2 Dynamic Terms

A variable x over a signature Σ is a symbol distinct from $\mathcal{F}_\Sigma$. For each variable there is a *set* of sorts from S_Σ, denoted $s(x)$. We also write $x : s(x)$. The set of all variables is normally denoted by $\mathcal{X}$.

Definition 5.4. Dynamic Terms.
The set of dynamic terms, written $\overline{\mathcal{D}}_\Sigma(\mathcal{X})$, is given by

1. $\mathcal{X} \subseteq \overline{\mathcal{D}}_\Sigma(\mathcal{X})$.
2. if $f \in \mathcal{F}_\Sigma, \alpha(f) = n, S \subseteq S_\Sigma$ and $d_1, \ldots d_n \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ then the pair $f(d_1, \ldots, d_n)\bullet S \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$.

The *Dynamic Sort*, $\mathcal{DS}$, of a dynamic term is defined as $\mathcal{DS}(x) = \{s(x)\}$ if $x \in \mathcal{X}$, $\mathcal{DS}(d\bullet S) = S$ otherwise. $\overline{\mathcal{D}}_\Sigma$ is the set of ground dynamic terms.

A dynamic term is *minimised* if the set of sorts at every node is minimised; it shall be assumed that all terms are minimised.

Notation 5.5. If $d\bullet\{s\}$, then we can omit the brackets, $d\bullet s$. Also we abbreviate the term $f(f(\cdots(f(d)\bullet S)\cdots)\bullet S)\bullet S)$ to $f^n(d)\bullet^n S$, and denote by $t \downarrow S$, where $t \in \overline{T}_{\overline{\Sigma}}(\mathcal{X})$ the dynamic term formed by inserting S at each node in the term, that is $x \downarrow S = x$ if $x \in \mathcal{X}$, $f(t_1,\ldots,t_n) \downarrow S = f(t_1 \downarrow S,\ldots,t_n \downarrow S)\bullet S$

A canonical way in which to relate the dynamic terms to a term in $\overline{T}_{\overline{\Sigma}}(\mathcal{X})$, is given by the following forgetful mapping

Definition 5.6. $\phi : \overline{\mathcal{D}}_{\Sigma}(\mathcal{X}) \rightarrow \overline{T}_{\overline{\Sigma}}(\mathcal{X})$ is defined as

$$\begin{aligned} \phi(x) &= x \text{ if } x \in \mathcal{X}, \\ \phi(f(d_1,\ldots,d_n)\bullet S) &= f(\phi(d_1),\ldots,\phi(d_n)) \text{ otherwise.} \end{aligned}$$

$\phi(d)$ is known as the *resolvent* of d.

Thus a term $t \in \overline{T}_{\overline{\Sigma}}(\mathcal{X})$ is represented as a set of dynamic terms, each with the same resolvent, but with differing sets of sorts tagged at each node. This can be expressed by an isomorphism between terms $t \in \overline{T}_{\overline{\Sigma}}(\mathcal{X})$ and the sets formed by $\{d|\phi(d) = t, d \in \overline{\mathcal{D}}_{\Sigma}(\mathcal{X})\}$. The following important relation exists between dynamic terms.

Definition 5.7. Two dynamic terms d_1, d_2 are *ϕ-equivalent*, written $d_1 \simeq d_2$, if their resolvents are equal, $\phi(d_1) = \phi(d_2)$. Note that the quotient of $\overline{\mathcal{D}}_{\Sigma}(\mathcal{X})$ by $\simeq$, $\overline{\mathcal{D}}_{\Sigma}(\mathcal{X})/\simeq$ is isomorphic to $\overline{T}_{\overline{\Sigma}}(\mathcal{X})$.

A canonical way to transform a $\bar{\Sigma}$-Term into a dynamic term is by tagging each subterm with its least sort.

Definition 5.8. Given $t \in \overline{T}_{\overline{\Sigma}}(\mathcal{X})$ then the *least syntactic dynamic term* $L(t)$ is given by $L(t) = t$ if $t \in \mathcal{X}$, or $L(t) = f(L(t_1),\ldots,L(t_n))\bullet\mathcal{LS}(t)$ if $t = f(t_1,\ldots,t_n)$.

We extend this to dynamic terms. The least syntactic dynamic term of $d \in \overline{\mathcal{D}}_{\Sigma}(\mathcal{X})$, $L(d)$

is given by $L(\phi(d))$. If $d = L(d)$ then it is in *L-normal form.* $\mathcal{L}_\Sigma(\mathcal{X})$ is the set of L-normal forms by .

Thus the least syntactic dynamic term of a dynamic term is the dynamic term with the same syntactic form, but tagged with the least sort at all paths.

We then define those dynamic terms which carry valid sets of sorts for their corresponding term in $\overline{T}_{\overline{\Sigma}}(\mathcal{X})$.

Definition 5.9. Well-sorted Dynamic Terms.
A dynamic term d is *well-sorted* if and only if $d = f(d_1, \ldots, d_n)\bullet S$, $S \geq \mathcal{SS}(\phi(d))$ and if $d_i, i = 1 \ldots n$ are well-sorted, or $d \in \mathcal{X}$. Then S is a *valid* set of sorts for d.

Further, a term $d = f(d_1, \ldots, d_n)\bullet S$ is *minimally well-sorted* if and only if it is well-sorted and $\forall s \in S, \forall s' \in (S - \{s\}).s' \not\leq s$, and all $d_i, i = 1 \ldots n$ are minimally well-sorted. All $x \in \mathcal{X}$ are trivially minimally well-sorted.

The set of well-sorted dynamic terms is denoted as $\mathcal{D}_\mathcal{S}(\mathcal{X})$, and the set of well-sorted ground dynamic terms is denoted $\mathcal{D}_\mathcal{S}$.

The set of sorts carried with a well-sorted dynamic term represents a set of sorts such that in any model of the specification, the denotation of that term's resolvent is a member of the denotations of all these sorts. The set of sorts at the nodes of a minimally well-sorted dynamic term is a minimal representation of such a set of sorts. Thus the denotation of the resolvent will be in the *intersection* of these sorts.

We define the dynamic term which carries the maximal amount of valid sort information for each subterm.

Definition 5.10. Given $t \in \overline{T}_{\overline{\Sigma}}(\mathcal{X})$ then the *least semantic dynamic term* $\mathcal{SD}(t)$ is given by $\mathcal{SD}(t) = t$ if $t \in \mathcal{X}$, or if $t = f(t_1, \ldots, t_n)$ then $\mathcal{SD}(t) = f(\mathcal{SD}(t_1), \ldots, \mathcal{SD}(t_n))\bullet\mathcal{SS}(t)$.

Again, we extend this to dynamic terms. The least semantic dynamic term of $d \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, $\mathcal{SD}(d)$ is given by $\mathcal{SD}(\phi(d))$. A dynamic term such that $d = \mathcal{SD}(d)$ is in *S-normal form.*

A trivial lemma is that these canonical conversions of terms to dynamic terms are into the set of well-sorted dynamic terms.

Lemma 5.11. $\forall t \in \overline{T}_{\overline{\Sigma}}(\mathcal{X}) \cdot L(t) \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$ and $\forall t \in \overline{T}_{\overline{\Sigma}}(\mathcal{X}) \cdot \mathcal{SD}(t) \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$.

Two other trivial dynamic terms can be defined for each $t \in \overline{T}_{\overline{\Sigma}}(\mathcal{X})$.

Definition 5.12. The top term $\top(t) = t \downarrow \{\}$, and the bottom term $\bot(t) = t \downarrow S_{\Sigma}$. Thus $\mathcal{SD}(t) = \top(t)$ if and only if $t \in I_{\mathcal{S}}(\mathcal{X})$.

We shall also need to refer to the set of ill-sorted dynamic terms.

Definition 5.13. The set of *ill-sorted dynamic terms* $\mathcal{ID}_{\mathcal{S}}(\mathcal{X})$ is given by

$$\mathcal{ID}_{\mathcal{S}}(\mathcal{X}) = \overline{\mathcal{D}}_{\Sigma}(\mathcal{X}) - \mathcal{D}_{\mathcal{S}}(\mathcal{X}).$$

By $\mathcal{ID}_{\mathcal{S}}$ we denote the ill-sorted ground dynamic terms.

Example 5.14. Given the Order-Sorted Specification given in Example 4.45 we can enumerate the following sets of minimised ground dynamic terms:

$$\begin{aligned}
\overline{\mathcal{D}}_{\Sigma} &= \{a\bullet\{\}, a\bullet A, a\bullet B, a\bullet C, a\bullet\{A,C\}, b\bullet\{\}, b\bullet A, b\bullet B, b\bullet C, b\bullet\{A,C\},\\
&\quad c\bullet\{\}, c\bullet A, c\bullet B, c\bullet C, c\bullet\{A,C\}, f(a\bullet A)\bullet\{\}, f(a\bullet A)\bullet A, f(a\bullet A)\bullet B,\\
&\quad f(a\bullet A)\bullet C, f(a\bullet A)\bullet\{A,C\}, f^n(a\bullet A)\bullet^n A, f(f(a\bullet A)\bullet A)\bullet B \ldots\}\\
\mathcal{D}_{\mathcal{S}} &= \{a\bullet\{\}, a\bullet A, a\bullet B, b\bullet\{\}, b\bullet A, b\bullet B, c\bullet C, f^n(a\bullet A)\bullet^n A, f^n(b\bullet A)\bullet^n A, \ldots\}\\
\mathcal{ID}_{\mathcal{S}} &= \{a\bullet C, a\bullet\{C,A\}, b\bullet C, b\bullet\{C,A\}, c\bullet A, c\bullet\{C,A\}, f(c\bullet C)\bullet C, f(a\bullet C)\bullet C,\\
&\quad f(a\bullet C)\bullet C, \ldots\}
\end{aligned}$$

□

Analogously to standard terms, paths are introduced to access subterms within dynamic terms.

Definition 5.15. A *Path* is a possibly empty (written ϵ) finite sequence of integers, written

$n_1.n_2.\cdots.n_k$. The set of paths for a dynamic term d, $O(d)$ is defined to be:

$O(d) = \{\epsilon\}$ if d is a variable or a constant

$O(f(d_1,\ldots,d_n)\bullet S) = \{\epsilon\} \cup \{i.p | 1 \leq i \leq n, p \in O(d_i)\}$

We extend the notation to joining paths: if p, q are paths, then so is $p.q$ (conventionally dropping the intervening ϵ). Path p is a subpath of path q, written $p \leq q$, if p is an initial subsequence of q. If neither $p \leq q$, nor $q \leq p$ then the paths are independent, written $p \bowtie q$.

Given a term d and a path $p \in O(d)$, the subterm occurring at p written $t|_p$ is $d|_\epsilon = d$ if $p = \epsilon$ and $f(d_1,\ldots,d_n)\bullet S|_{i.u} = d_i|_u$ if $p = i.u$. The result of replacing a subterm of d occurring at path $p \in O(d)$ with a term e is written as $d[p \leftarrow e]$. The height of a dynamic term $ht(d)$ is the length of the longest path in d, defined as $ht(x) = 1$, $ht(f(d_1,\ldots,d_n)\bullet S) = 1 + max\{ht(d_1),\ldots,ht(d_n)\}$.

5.2.3 Dynamic Approximation

One characterisation of ordinary terms using dynamic terms is by considering the set of dynamic terms which have the same resolvent. Another characterisation which has more structure and which is useful later, is to use an approximation relation on dynamic terms.

Definition 5.16. $d_1 \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ is a *dynamic approximation* of $d_2 \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, written $d_1 \trianglerighteq d_2$, if and only if

1. $d_1 \simeq d_2$
2. $\forall p \in O(d_1).DS(d_1|_p) \geq DS(d_2|_p)$

Lemma 5.17. $\trianglerighteq$ forms a partial ordering on minimised terms.

Proof (5.17). Trivial from $\leq$ being a partial order on minimised sets of sorts. □

The approximation ordering partitions the set of dynamic terms. The strongly connected

components of the graph of this relation are isomorphic to the set of ordinary terms.

Definition 5.18. Let the relation $\underline{\trianglelefteq\trianglerighteq}$ be the symmetric transitive closure of $\trianglerighteq$.

Then we have the isomorphism

$$\forall t \in \overline{T}_{\overline{\Sigma}}(\mathcal{X}) \cdot t \cong \{d | d \underline{\trianglelefteq\trianglerighteq} L(t), d \in \overline{\mathcal{D}}_{\Sigma}(\mathcal{X})\}$$

This isomorphism however, is to an object with structure. For each term there is a lattice, with $\top(t)$ at the top and $\bot(t)$ at the bottom. The lattice for a term is shown in Figure 5.1.

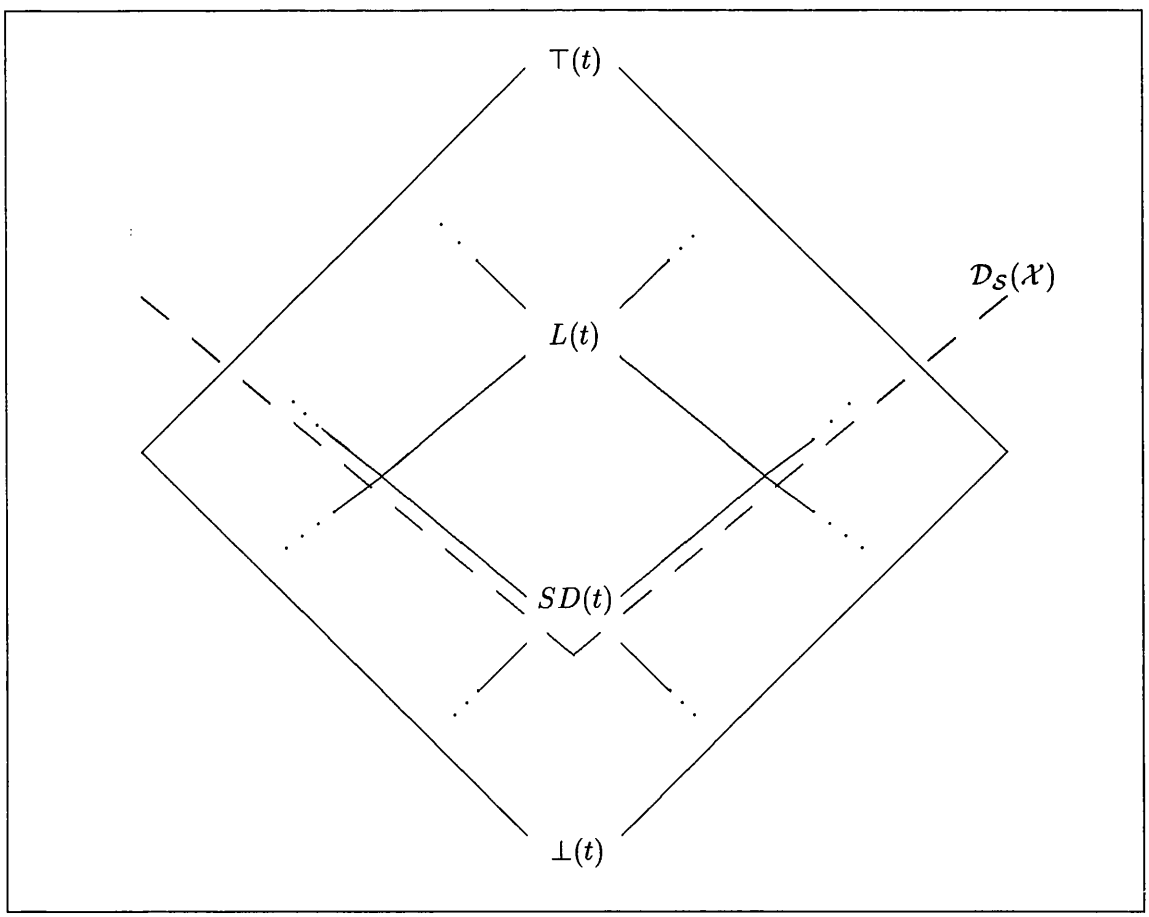


Figure 5.1: The Approximation Relation Lattice for $t \in \overline{T}_{\overline{\Sigma}}(\mathcal{X})$.

The lattice has meet and join terms.

Definition 5.19. Given $d_1, d_2 \in \overline{\mathcal{D}}_{\Sigma}(\mathcal{X})$ such that $d_1 \simeq d_2$, then $d_1 \wedge d_2$ is the dynamic term

such that $d_1 \simeq (d_1 \wedge d_2) \simeq d_2$ and $\forall p \in O(t_1).\mathcal{DS}((d_1 \wedge d_2)|_p) = M_\Sigma(\mathcal{DS}(d_1|_p) \cup \mathcal{DS}(d_2|_p))$. If $d_1 \trianglelefteq d_2$, then $d_1 \wedge d_2 = d_1$.

Similarly, $d_1 \vee d_2$ is the term such that $d_1 \simeq (d_1 \vee d_2) \simeq d_2$ and $\forall p \in O(t_1).\mathcal{DS}((d_1 \vee d_2)|_p) = M_\Sigma(\mathcal{DS}(d_1|_p) \cap \mathcal{DS}(d_2|_p))$. If $d_1 \trianglelefteq d_2$, then $d_1 \vee d_2 = d_2$.

We give some basic properties of the approximation ordering in the next two lemmas.

Lemma 5.20. If $d_1 \trianglerighteq d_2$, then $L(d_1) = L(d_2)$ and $\mathcal{SD}(d_1) = \mathcal{SD}(d_2)$.

This is obvious as they both have the same resolvent. More interesting are the following:

Lemma 5.21. $\forall d, d_1, d_2 \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ such that $d_1 \simeq d \simeq d_2$,

1. $\top(\phi(d)) \trianglerighteq d$.
2. $d \trianglerighteq \bot(\phi(d))$.
3. $d \in \mathcal{D}_\mathcal{S}(\mathcal{X})$ if and only if $d \trianglerighteq \mathcal{SD}(d)$.
4. $d_1, d_2 \trianglerighteq (d_1 \wedge d_2)$.
5. If $d_1, d_2 \trianglerighteq d$ then $(d_1 \wedge d_2) \trianglerighteq d$.
6. If $d_1, d_2 \in \mathcal{D}_\mathcal{S}(\mathcal{X})$ then $(d_1 \wedge d_2) \in \mathcal{D}_\mathcal{S}(\mathcal{X})$.
7. If $d_1 \trianglerighteq d_2$ and $d_2 \in \mathcal{D}_\mathcal{S}(\mathcal{X})$ then $d_1 \in \mathcal{D}_\mathcal{S}(\mathcal{X})$.
8. If $d_1 \trianglerighteq d_2$ and $d_1 \in \mathcal{ID}_\mathcal{S}(\mathcal{X})$ then $d_2 \in \mathcal{ID}_\mathcal{S}(\mathcal{X})$.

Proof (5.21). 1, 2 and 4 are immediate from the definitions.

For 3, note that for all $p \in O(d)$, $\mathcal{DS}(d|_p) \geq S(\phi(d)|_p) \geq \mathcal{SS}(\phi(\mathcal{SD}(d))|_p) = \mathcal{DS}(\mathcal{SD}(d)|_p)$.

For 5, let $d_1, d_2 \trianglerighteq d'$. Then $\forall p \in O(d_1).DS(d_1|_p) \geq DS(d'|_p) \wedge DS(d_2|_p) \geq DS(d'|_p)$. Hence $\forall p \in O(d_1).DS(d_1|_p) \cup DS(d_2|_p) \geq DS(d'|_p)$.

6 is immediate from 5 noting that $d_1, d_2 \trianglerighteq \mathcal{SD}(d_1)$, and 7 is a corollary of 3.

For 8 if $d_2 \in \mathcal{D}_\mathcal{S}(\mathcal{X})$ then $d_2 \trianglerighteq \mathcal{SD}(d)$ and so by transitivity $d_1 \trianglerighteq \mathcal{SD}(d)$, which contradicts

the assumption. □

The approximation ordering clarifies the relationship between dynamic terms and the sorts of terms. Given a dynamic term d, the least well-sorted dynamic term according to this approximation ordering is the least semantic dynamic term, that is the minimum of the set $\{d' | d \simeq d' \text{ and } d' \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})\}$ (assuming that this set is not empty). Clearly this is not decidable, but with each sort deduction the dynamic term moves down this lattice to make a better approximation to the least semantic term.

Example 5.22. Using the specification in Example 5.14:

$$\begin{aligned}
\top(f(a)) = f(a\bullet\{\})\bullet\{\} &\trianglerighteq L(f(a)) = f(a\bullet A)\bullet A = \mathcal{SD}(f(a)) \\
&\trianglerighteq \bot(f(a)) = f(a\bullet\{A,C\})\bullet\{A,C\} \\
\top(f(b)) = f(b\bullet\{\})\bullet\{\} &\trianglerighteq f(b\bullet B)\bullet\{\} = L(f(b)) \trianglerighteq f(b\bullet A)\bullet A = \mathcal{SD}(f(b)) \\
&\trianglerighteq \bot(f(b)) = f(b\bullet\{A,C\})\bullet\{A,C\}
\end{aligned}$$

□

5.2.4 Dynamic Substitution

Dynamic substitutions are almost everywhere identity mappings between $\mathcal{X}$ and $\overline{\mathcal{D}}_{\Sigma}(\mathcal{X})$.

Definition 5.23. A *dynamic substitution* σ is an almost everywhere identity mapping from $\mathcal{X}$ to $\overline{\mathcal{D}}_{\Sigma}(\mathcal{X})$ and for all $x \in Dom(\sigma)$, $\mathcal{DS}(\sigma x) \leq_{\Sigma} \mathcal{DS}(x)$. If for all $x \in Dom(\sigma), \sigma x \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$, then the substitution is well-sorted. The set of dynamic substitutions over a signature is given by $\overline{DSubst}_{\Sigma}$. The set of well-sorted dynamic substitutions over a signature is given by $DSubst_{\mathcal{S}}$.

To extend this to $\overline{\mathcal{D}}_{\Sigma}(\mathcal{X})$, σd is defined as: if $d \in \mathcal{X}$ then σd. Otherwise, $d = f(d_1, \ldots, d_n)\bullet S$, and $\sigma d = f(\sigma d_1, \ldots, \sigma d_n)\bullet S$.

We denote the composition of substitutions by $\theta \circ \sigma$ or just $\theta\sigma$ where $(\theta \circ \sigma)s = \theta(\sigma(s))$. We also define the forgetful function on dynamic substitutions into $\mathcal{W}$-substitutions. If

$\sigma = \{x_1 \mapsto d_1, \ldots, x_n \mapsto d_n\}$ then:

$$\phi(\sigma) = \{x_1 \mapsto \phi(d_1), \ldots, x_n \mapsto \phi(d_n)\}$$

Note that $\phi(\sigma)\phi(d) = \phi(\sigma d)$ and therefore we abuse notation by writing $\sigma\phi(d)$.

Note that these substitutions are *dynamic-sort preserving.* The intuition can clearly be seen by converting back to $\mathcal{W}$-substitutions by applying the function ϕ to the range of σ. Thus the substitution maps variables to terms of sorts less than or equal to its own.

Lemma 5.24. For any $\sigma \in \overline{DSubst_\Sigma}$ if $d \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ then $\forall p \in O(d)\ \mathcal{DS}(d|_p) \geq \mathcal{DS}(\sigma d|_p)$. Further, if $d|_p \notin \mathcal{X}$ then $\mathcal{DS}(d|_p) = \mathcal{DS}(\sigma d|_p)$.

Proof (5.24). If $p \in O(d)$ is a leaf, then if $d|_p$ is a constant, $\sigma d|_p = d|_p$ and we are done. Otherwise $d|_p$ is a variable, and by definition $\mathcal{DS}(\sigma d|_p) \leq \mathcal{DS}(d|_p)$. If p is not a leaf, then the dynamic sort is not changed by substitution, $\mathcal{DS}(\sigma d|_p) = \mathcal{DS}(d|_p)$. □

We extend the notions of $\simeq$, $\trianglerighteq$, meets and joins to substitutions and define a subsumption ordering on terms and substitutions.

Definition 5.25. Dynamic substitutions σ, ρ are *ϕ-equivalent*, written $\sigma \simeq \rho$, if and only if $\forall x \in \mathcal{X} \cdot \sigma x \simeq \rho x$.

A substitution σ_2 *approximates* substitution σ_1 written $\sigma_1 \trianglelefteq \sigma_2$ if $Dom(\sigma_1) = Dom(\sigma_2)$ and $\forall x \in Dom(\sigma_1)\ \sigma_1 x \trianglelefteq \sigma_2 x$.

If $\sigma \simeq \rho$, then

1. $\sigma \vee \rho = \{x \mapsto \sigma x \vee \rho x\}, \forall x \in \mathcal{X}$.
2. $\sigma \wedge \rho = \{x \mapsto \sigma x \wedge \rho x\}, \forall x \in \mathcal{X}$.

Lemma 5.26. If $\sigma, \rho \in \overline{DSubst_\Sigma}$, then $\sigma \vee \rho \in \overline{DSubst_\Sigma}$ and $\sigma \wedge \rho \in \overline{DSubst_\Sigma}$. If $\sigma, \rho \in DSubst_\mathcal{S}$, then $\sigma \vee \rho \in DSubst_\mathcal{S}$ and $\sigma \wedge \rho \in DSubst_\mathcal{S}$.

Proof (5.26). Since $\mathcal{DS}(\sigma x) \leq s_x$, and $\mathcal{DS}(\sigma x) \leq s_x$, then $\mathcal{DS}(\sigma x \vee \rho x) \leq s_x$, therefore $\sigma \vee \rho \in \overline{DSubst_\Sigma}$; $\sigma \wedge \rho \in \overline{DSubst_\Sigma}$ is obvious, Also note that if $\sigma x, \rho x \in \mathcal{D}_\mathcal{S}(\mathcal{X})$ then $\sigma x \wedge \rho x \in \mathcal{D}_\mathcal{S}(\mathcal{X})$ and $\sigma x \vee \rho x \in \mathcal{D}_\mathcal{S}(\mathcal{X})$, so the second part holds. □

Definition 5.27. Subsumption Preorder. A dynamic term t *subsumes* a dynamic term s written $t \preceq s$ if and only if there exists a dynamic substitution σ such that $s \simeq \sigma t$. Such a substitution is called a *matching* substitution[2].

Definition 5.28. A dynamic substitutions σ is *more general* than a dynamic substitution σ' with respect to a set of variables $\mathcal{W}$, written $\sigma \preceq^\mathcal{W} \sigma'$, if there is a dynamic substitution π such that $\forall d \in \mathcal{D}_\mathcal{S}(\mathcal{W}) \cdot \sigma'(d) \simeq \pi \circ \sigma(d)$. Further $\forall x \in \mathcal{W} \cdot \sigma'(x) \trianglelefteq \pi\sigma(x)$. If $\sigma \preceq^\mathcal{W} \sigma'$ and also $\sigma' \preceq^\mathcal{W} \sigma$ then they are *equivalent*, $\sigma \equiv^\mathcal{W} \sigma'$. If $\mathcal{W} = \mathcal{X}$, then we may drop the superscript.

Note the approximation ordering condition used in this definition of more general substitutions. Special cases of substitution are specialisation and renaming.

Definition 5.29. A *specialisation* ρ is a dynamic substitution such that $\forall x \in Dom(\rho) \cdot \rho x \in X$ and $s(x) \geq s(\rho x)$. A *renaming* ρ is a specialisation such that $\forall x \in Dom(\rho) \cdot s(x) = s(\rho x)$. Dynamic terms d_1, d_2 are *alpha-equivalent*, $d_1 =_\alpha d_2$, if there is a renaming ρ such that $\rho d_1 = d_2$.

A subclass of dynamic substitutions is those with range terms in L-normal form.

Definition 5.30. A dynamic substitution σ is a Σ-substitution if $\forall x \in Dom(\sigma)$, σx is in L-normal form.

Σ-substitutions thus correspond to Σ-substitutions in standard order-sorted theory. It is trivial to see that a Σ-substitution is a well-sorted dynamic substitution.

We give some properties of substitutions with respect to the approximation ordering.

[2] We shall consider well-sorted matching in more detail later.

Lemma 5.31.

1. If $d_2 \trianglerighteq d_1$ then $d[p \leftarrow d_2] \trianglerighteq d[p \leftarrow d_1]$ (monotonicity) and $\sigma d_2 \trianglerighteq \sigma d_1$ (stability).

2. If $\sigma d_1 \trianglerighteq d$ and $d_2 \trianglerighteq d_1$ then $\sigma d_2 \trianglerighteq d$.

3. If $\theta s \trianglerighteq t$ and $\sigma t \trianglerighteq u$ then $\sigma\theta s \trianglerighteq u$.

Proof (5.31). 1 is obvious from their definitions, and 2 follows as a consequence of stability, and transitivity of $\trianglerighteq$.

For 3, note that $\phi(\theta s) = \phi(t)$ and so $\phi(\sigma\theta s) = \phi(\sigma t)$. Consider $p \in O(t)$. If $t|_p = x \in Dom(\sigma)$ then $\theta s = x$ and so $\sigma\theta s = \sigma t|_p \trianglerighteq u|_p$. If $t|_p \notin Dom(\sigma)$, then $\mathcal{DS}(u|_p) \leq \mathcal{DS}(\sigma t|_p) = \mathcal{DS}(t|_p) \leq \mathcal{DS}(\sigma\theta s|_p) = \mathcal{DS}(\theta s|_p)$. □

Important special cases arise when we wish to consider substitutions which preserve the well sorted properties.

Lemma 5.32. If $d \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$ and σ is a well-sorted substitution then $\sigma d \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$.

Proof (5.32). If $\sigma \in DSubst_{\mathcal{S}}$, then $\forall x_i : s \in Dom(\sigma)$, $\forall s' \in \mathcal{DS}(\sigma x_i)$ $\exists t_i \in T_\Sigma(\mathcal{X})$ such that $\phi(\sigma x_i) =_{s'} t_i$. Thus $\xi = \{x_i \mapsto \phi(\sigma x_1), \ldots, x_n \mapsto \phi(\sigma x_n)\}$ is a $\mathcal{W}$-substitution. Then as $d \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$, $\forall s \in \mathcal{DS}(d)$ there is a term $t \in T_\Sigma(\mathcal{X})$ such that $\phi d =_s t$. As ξ is a $\mathcal{W}$-substitution, then $\xi(\phi d) =_s \xi t$. Hence $\phi(\sigma d) =_s \xi t$. This holds for all $s \in \mathcal{DS}(d)$ and thus $\sigma d \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$. □

Another important lemma is the following.

Lemma 5.33. If $d \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, $d' \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$ and for some $\sigma \in \overline{DSubst}_\Sigma$, $\sigma d \trianglerighteq d'$ then $\sigma d \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$.

Proof (5.33). Immediate from Lemma 5.21 part 3. □

5.3 Sort Propagation

We call the logic described in Chapter 4 *dynamic* order-sorted equational logic as the sorts of equations and terms can change through the action of the equational theory. However, it may be possible to infer statically more sort information about the dynamic term than the current dynamic sorts. This inference is known as *Sort Propagation* and has two forms: Sort Balancing and Sort Percolation.

5.3.1 Sort Balancing

Sort balancing ensures that subterms with the same underlying form have the same sorts.

Definition 5.34. Balanced Term $d \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ is sort-balanced if:

$$\forall p, q \in O(d) \cdot d|_p \simeq d|_q \Rightarrow \mathcal{DS}(d|_p) = \mathcal{DS}(d|_q)$$

Sort Balancing thus propagates sort information across a term: if two subterms are the same up to their operator structure then it is reasonable to give both the same dynamic sorts. This observation gives rise to the rule of inference in Figure 5.2.

(1) Balance.

$$\frac{d[p \leftarrow d_1 \bullet S_1][q \leftarrow d_2 \bullet S_2]}{d[p \leftarrow d_1 \wedge d_2][q \leftarrow d_1 \wedge d_2]} \quad \text{if } p \bowtie q,\ \phi(d_1 \bullet S_1) = \phi(d_2 \bullet S_2) \text{ and } S_1 \neq S_2.$$

Figure 5.2: The Balancing Rule

We denote by $B_{p,q}(d)$ the result of applying Balance at p, q and $B_\Sigma(d)$ the result of repeatedly applying the rule to a term until it cannot be applied again.

Lemma 5.35. For any $d \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, balancing terminates with $B_\Sigma(d)$ which is uniquely defined.

Proof (5.35). For $d \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ and path $p \in O(d)$, form the multiset $M_p(d)$ such that

$$M_p(d) = \{\mathcal{DS}(d|_q) | d|_q = d|_p\}$$

Let $\sqcup M_p(d) = \{\bigcup M_p(d)\}$. Then, if $\mathcal{DS}(d|_p) \neq \mathcal{DS}(d|_q)$:

$$M_p(B_{p,q}(d)) = M_p(d) - \{\mathcal{DS}(d|_p), \mathcal{DS}(d|_q)\} \cup \{\mathcal{DS}(d|_p) \cup \mathcal{DS}(d|_q)\}$$

(using multiset '$-$' and '$\cup$'). Let $\prec\!\!\prec$ be the multiset extension of $\geq_\Sigma$. Clearly:

$$M_p(d) \prec\!\!\prec M_p(B_{p,q}(d)) \stackrel{\preccurlyeq}{=} \sqcup M_p(d)$$

So each $M_p(d)$ is bounded below and balancing strictly decreases in this relation. We take as a measure the multiset $M(d) = \{M_p(d) | p \in O(d)\}$, then using the multiset extension of $\prec\!\!\prec$, this relation is strictly decreasing and bounded below, and as balancing respects this ordering, balancing is terminating.

For unique definition we show that for any $p \in O(d)$, $\mathcal{DS}(B_\Sigma(d)|_p) = \sqcup M_p(d)$. Firstly, note that $\forall q \in O(d)$ such that $d|_q = d|_p$, $\mathcal{DS}(B_\Sigma(d)|_p) = \mathcal{DS}(B_\Sigma(d)|_q)$, otherwise $B_\Sigma(d)$ will not have terminated. From the above:

$$\{\mathcal{DS}(B_\Sigma(d)|_p)\} = M_p(B_\Sigma(d)) \stackrel{\preccurlyeq}{=} \sqcup M_p(d)$$

But $M_p(d) \prec\!\!\prec M_p(B_\Sigma(d))$, and so by definition of lower bound $\sqcup M_p(d) \stackrel{\preccurlyeq}{=} M_p(B_\Sigma(d))$. □

Lemma 5.36. The following facts about Balanced terms hold.

1. $B_\Sigma(d)$ is sort balanced.
2. $d \trianglerighteq B_\Sigma(d)$.
3. If $d \in \mathcal{D}_\mathcal{S}(\mathcal{X})$ if and only if $B_\Sigma(d) \in \mathcal{D}_\mathcal{S}(\mathcal{X})$.
4. If $\sigma d \in \mathcal{D}_\mathcal{S}(\mathcal{X})$ then $\sigma(B_\Sigma(d)) \in \mathcal{D}_\mathcal{S}(\mathcal{X})$.
5. If $\sigma d \notin \mathcal{D}_\mathcal{S}(\mathcal{X})$ then $\sigma(B_\Sigma(d)) \notin \mathcal{D}_\mathcal{S}(\mathcal{X})$.
6. If $d \trianglerighteq d'$ then $B_\Sigma(d) \trianglerighteq B_\Sigma(d')$.

Proof (5.36). 1 is immediate from definition.

For 2, given d such that $d[p \leftarrow d_1 \bullet S_1][q \leftarrow d_2 \bullet S_2]$, $p \bowtie q$, $\phi(d_1 \bullet S_1) = \phi(d_1 \bullet S_1)$ and $S_1 \neq S_2$ then $d \trianglerighteq B_{p,q}(d)$ since at paths $DS(d|_p) \geq DS(B_{p,q}(d)|_p)$ and $DS(d|_q) \geq DS(B_{p,q}(d)|_q)$ and all other subterms remain the same. By transitivity of $\trianglerighteq$, $d \trianglerighteq B_\Sigma(d)$.

Again we show that each step preserves the property to show 3. By assumption $S_1 \geq \mathcal{DS}(\mathcal{SD}(d)|_p)$ and $S_2 \geq \mathcal{DS}(\mathcal{SD}(d)|_q)$. Hence $S_1 \cup S_2 \geq \mathcal{DS}(\mathcal{SD}(d)|_p)$ and $S_1 \cup S_2 \geq \mathcal{DS}(\mathcal{SD}(d)|_q)$, and so $B_{p,q}(d) \in \mathcal{D}_\mathcal{S}(\mathcal{X})$. The opposite direction is trivial from part 2.

For part 4, we show that one balance step preserves this property. By assumption $\mathcal{SS}(\phi(\sigma(d|_p))) = \mathcal{SS}(\phi(\sigma(d|_q))) \leq \mathcal{DS}(\sigma(d|_p))$, $\mathcal{DS}(\sigma(d|_q))$. So $\mathcal{SS}(\phi(\sigma(d|_q))) \leq \mathcal{DS}(\sigma(B_{p,q}(d|_p)))$, $\mathcal{DS}(\sigma(B_{p,q}(d|_q)))$, and we are done.

For part 5, by part 2, $d \trianglerighteq B_\Sigma(d)$, so by Lemma 5.31 part 1 $\sigma d \trianglerighteq \sigma(B_\Sigma(d))$ and thus we are done by Lemma 5.21 part 8.

For part 6, we consider one step of balancing at positions p, q. Thus $d = d[p \leftarrow d_1 \bullet S_1][q \leftarrow d_2 \bullet S_2]$ and $d' = d'[p \leftarrow d'_1 \bullet S'_1][q \leftarrow d'_2 \bullet S'_2]$, where $S_1 \geq S'_1$ and $S_2 \geq S'_2$. Thus $S_1 \cup S_2 \geq S'_1 \cup S'_2$, so $d[p \leftarrow d_1 \bullet S_1 \cup S_2][q \leftarrow d_2 \bullet S_2 \cup S_1] \trianglerighteq d'[p \leftarrow d'_1 \bullet S'_1 \cup S'_2][q \leftarrow d'_2 \bullet S'_2 \cup S'_1]$, as all other subterms are unchanged. □

Sort balance is not preserved under substitution as the following example shows.

Example 5.37. We augment the specification in Example 5.14 with the ranks:

Operators: $g : B\ B \rightarrow B \quad f : B \rightarrow B$
$g : A\ A \rightarrow A$

The term $d = g(f(b \bullet B) \bullet B, g(x : B, y : B) \bullet B) \bullet B$ is well-sorted and balanced. If $\sigma = \{x : B \mapsto f(b \bullet A) \bullet A, y : B \mapsto f(b \bullet A) \bullet A\}$ which is a well-sorted and balanced substitution (that is all range terms are balanced with respect to each other), then $\sigma d = g(f(b \bullet B) \bullet B, g(f(b \bullet A) \bullet A, f(b \bullet A) \bullet A) \bullet B) \bullet B$. Balancing this term results in $B_\Sigma(\sigma d) = g(f(b \bullet A) \bullet A, g(f(b \bullet A) \bullet A, f(b \bullet A) \bullet A) \bullet B) \bullet B$. □

Balancing 'distributes' through substitution and subterm replacement.

Lemma 5.38. Given $d \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ then:

1. $\forall \sigma \in \overline{DSubst}_\Sigma$, $B_\Sigma(\sigma(B_\Sigma(d))) = B_\Sigma(\sigma d)$.
2. $\forall d' \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ and position $p \in O(d)$ $B_\Sigma(d[p \leftarrow B_\Sigma(d')]) = B_\Sigma(d[p \leftarrow d'])$.

Proof (5.38). Part 1. Consider a step in balancing σd, at paths p, q. Then if both p and q are non-variable paths in d, then there is a balancing step at those paths in d. Doing all such balances first results in $\sigma(B_\Sigma(d))$. Clearly, $\sigma(B_\Sigma(d)) = B_\Sigma(\sigma d)$, and thus the lemma holds.

Part 2. Consider a balancing step at paths q, q'. Then if both q and q' are paths below p, such steps are balancing steps in d'. Performing all such balances first results in $d[p \leftarrow B_\Sigma(d')]$, and the lemma holds. □

A extension to balanced terms is the concept of balanced sets of terms.

Definition 5.39. A set of dynamic terms $\{d_i\}_{i \in I}$ is said to be *balanced* if $\forall i, j \in I$ and $p \in O(d_i)$ and $q \in O(d_j)$, if $d_i|_p \simeq d_j|_q$ then $d_i|_p = d_j|_q$.

5.3.2 Sort Percolation

Sort Percolation is similar to the least-sort function. It uses the ranks of operators in the signature to infer further sort information using the dynamic sorts of subterms.

Definition 5.40. A $d \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ is in *Signature Least Form (ΣL-form)* if $\forall p \in O(d)$ and $d|_p = f(d_1, \ldots, d_n) \bullet S$ and $\omega = \langle s_1 \ldots s_n \rangle = \langle \mathcal{DS}(d_1) \ldots \mathcal{DS}(d_n) \rangle$ then the set $\{s' | \omega \leq \omega', (\omega', s') \in ranks(f)\} \geq S$.

Sort percolation propagates sort information towards the root of a term. If a subterm changes sort, the argument of the parent operator may fall into the range of a different

rank for that operator, and thus the parent operator has a new sort. This is captured in the rule of inference in Figure 5.3.

(2) Percolate.

$$\frac{d[p{\leftarrow}f(d_1,\ldots,d_n){\bullet}S]}{d[p{\leftarrow}f(d_1,\ldots,d_n){\bullet}S \cup \{s\}]} \quad \text{if } (\langle s_1 \ldots s_n\rangle, s) \in \mathit{ranks}(f), \mathcal{DS}(d_1) \leq s_1, \ldots, \mathcal{DS}(d_n) \leq s_n \text{ and } S \not\leq s$$

Figure 5.3: The Percolate Rule

$P_{p,(\omega,s)}(d)$ is the result of applying the percolate rule to a term at path p and rank (ω, s). $P_\Sigma(d)$ is the result of repeatedly applying the percolate rule until it cannot be applied again.

Lemma 5.41. For every $d \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, $P_\Sigma(d)$ always exists.

Proof (5.41). For each $d \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ form the multiset of paths $MP(d)$, with one entry for each rank where the above percolation rule applies. Let $\prec\!\!\prec$ be the multiset extension of the strict subpath ordering on paths, which is noetherian. We show that the percolate rule reduces in this ordering. Let q be the parent node of p, for some k $p = q.k$, and let g be the operator at q. Then either after percolation there is a rank $(\omega', s') \in \mathit{ranks}(g)$ such that now $\mathcal{DS}(P_{p,(\omega,s)}(d)|_p) \leq s'_k$ where it was not before and also $\forall i \neq k \cdot \mathcal{DS}(d|_{q.i}) \leq s'_i$. So $MP(P_{p,(\omega,s)}(d)) = MP(d) - p \cup \{q\}$, As $q < p$, $MP(d) \prec\!\!\prec MP(P_{p,(\omega,s)}(d))$. Alternatively, if this is not the case, $\#MP(P_{p,(\omega,s)}(d)) = \#MP(d) - 1$, so $MP(d) \prec\!\!\prec MP(P_{p,(\omega,s)}(d))$. □

Lemma 5.42. For every $d \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, $P_\Sigma(d)$ is uniquely defined.

Proof (5.42). Consider two paths $p, q \in O(d)$, such that $d|_p = f(d_1,\ldots,d_n){\bullet}S$, $(\langle s_1 \ldots s_n\rangle, s) \in \mathit{ranks}(f)$, $\mathcal{DS}(d_1) \leq s_1, \ldots, \mathcal{DS}(d_n) \leq s_n$ and $S \not\leq s$, and also $d|_q = g(e_1,\ldots,e_m){\bullet}R$, $(\langle r_1 \ldots r_m\rangle, r) \in \mathit{ranks}(g)$, $\mathcal{DS}(e_1) \leq r_1, \ldots, \mathcal{DS}(e_m) \leq r_n$ and $R \not\leq r$.

If $p \bowtie q$, or if without loss of generality $p \leq q$ and there is a non-empty path p' and integer i such that $q = p.i.p'$ then clearly $P_{q,(\langle r_1 \ldots r_m\rangle,r)}(P_{p,(\langle s_1 \ldots s_n\rangle,s)}(d)) = P_{p,(\langle s_1 \ldots s_n\rangle,s)}(P_{q,(\langle r_1 \ldots r_m\rangle,r)}(d))$.

If, without loss of generality, $q = p.i$, for some i, then $\mathcal{DS}(P_{q,(\langle r_1 \ldots r_m\rangle,r)}(d)|_q) < \mathcal{DS}(d_1) \leq s_1$. So we can still percolate at p resulting in:

$$d[p \leftarrow f(d_1, \ldots, g(e_1, \ldots, e_m) \bullet R \cup \{r\}, \ldots, d_n) \bullet S \cup \{s\}]$$

which is also the result of doing percolation at p before percolation at q. □

We give some general properties of Percolation.

Lemma 5.43.

1. $P_\Sigma(d)$ is in ΣL-form.
2. $d \trianglerighteq P_\Sigma(d)$.
3. $d \in \mathcal{DS}(\mathcal{X})$ if and only if $P_\Sigma(d) \in \mathcal{DS}(\mathcal{X})$.
4. If $d \trianglerighteq d'$ then $P_\Sigma(d) \trianglerighteq P_\Sigma(d')$.
5. $L(t) = P_\Sigma(\top(t))$.

Proof (5.43). 1 is immediate from definition.

For 2, suppose that for some p, $d|_p = f(d_1, \ldots, d_n) \bullet S$ and $(\omega, s) = (\langle s_1 \ldots s_n \rangle, s) \in \mathit{ranks}(f)$, $\mathcal{DS}(d_1) \leq s_1, \ldots, \mathcal{DS}(d_n) \leq s_n$, and $S \not\leq s$. Then $\mathcal{DS}(d|_p) = S \geq S \cup \{s\} = \mathcal{DS}(P_{p,(\omega,s)}(d)|_p)$, and all other subterms remain the same, so by Lemma 5.31 part 1, $d \trianglerighteq P_{p,(\omega,s)}(d)$ and by transitivity of $\trianglerighteq$, $d \trianglerighteq P_\Sigma(d)$.

For 3, the 'if' direction is immediate from part 2 and Lemma 5.21 part 7. For the 'only if' we show that applications of percolation preserve the property. Suppose that for some $p \in O(d)$, $d|_p = f(d_1, \ldots, d_n) \bullet S$ and $(\omega, s) = (\langle s_1 \ldots s_n \rangle, s) \in \mathit{ranks}(f)$, $\mathcal{DS}(d_1) \leq s_1, \ldots, \mathcal{DS}(d_n) \leq s_n$, and $S \not\leq s$. As $d \in \mathcal{DS}(\mathcal{X})$, $\phi(d_1) :: s_1, \ldots, \phi(d_n) :: s_n$, and so by the congruence rule $f(\phi(d_1), \ldots, \phi(d_n)) :: s$. So $f(d_1, \ldots, d_n) \bullet s \in \mathcal{DS}(\mathcal{X})$, and by Lemma 5.21 part 6, $f(d_1, \ldots, d_n) \bullet S \cup \{s\} \in \mathcal{DS}(\mathcal{X})$, and so $P_{p,(\omega,s)}(d) = d[p \leftarrow f(d_1, \ldots, d_n) \bullet S \cup \{s\}] \in \mathcal{DS}(\mathcal{X})$.

For 4, consider when a percolation step can be applied to either d or d'. If a percolation step applies to d', then from the proof of part 2 $d'' \trianglelefteq d' \trianglelefteq d$. If the step is applied to d, then for some p, $d|_p = f(d_1, \ldots, d_n) \bullet S$ and $(\omega, s) = (\langle s_1 \ldots s_n \rangle, s) \in \mathit{ranks}(f)$, $\mathcal{DS}(d_1) \leq s_1, \ldots, \mathcal{DS}(d_n) \leq s_n$, and $S \not\leq s$. However, $\mathcal{DS}(d'_1) \leq \mathcal{DS}(d_1) \leq s_1, \ldots, \mathcal{DS}(d'_n) \leq$

$\mathcal{DS}(d_n) \leq s_n$ for the equivalent subterm $d'|_p$, and thus, if $\mathcal{DS}(d'|_p) \not\leq s$, there is also a percolation step available at p in d'. If $\mathcal{DS}(d'|_p) \leq s$, then we are done. Hence $P_\Sigma(d) \trianglerighteq P_\Sigma(d')$.

For 5, note that percolating from the leaves to the root performs all the sort calculations necessary to form the least-sort of each subterm. Note also that this term is balanced. □

Again percolation "distributes" through substitution and subterms.

Lemma 5.44.

1. $\forall d \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ and for any dynamic substitution σ, $P_\Sigma(\sigma(P_\Sigma(d))) = P_\Sigma(\sigma d)$.
2. $\forall d, d' \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ and path p in $O(d)$, $P_\Sigma(d[p \leftarrow P_\Sigma(d')]) = P_\Sigma(d[p \leftarrow d'])$.

Proof (5.44). Part 1. Given σd, then since σ is sort-preserving, any percolation step in this term at a non-variable path $p \in O(d)$ using a rank $f : s_1 \ldots s_2 \rightarrow s$, where $f = root(d|_p)$, and $\mathcal{DS}((\sigma d)|_{p.1}) \leq \mathcal{DS}(d|_{p.1}) \leq s_1 \ldots \mathcal{DS}((\sigma d)|_{p.n}) \leq \mathcal{DS}(d|_{p.n}) \leq s_n$ and $\mathcal{DS}((\sigma d)|_p) = \mathcal{DS}(d|_p) \not\leq s$ is also a percolation step for d. Recursively performing all such steps results in $\sigma(P_\Sigma(d))$ and we are done.

Part 2. Given $d[p \leftarrow d']$, then percolation step below p will be also percolation step in d'. Recursively performing all such steps results in $d[p \leftarrow P_\Sigma(d')]$ and we are done. □

Example 5.45. Continuing Example 5.37, we note that

$$B_\Sigma(\sigma d) = g(f(b \bullet A) \bullet A, g(f(b \bullet A) \bullet A, f(b \bullet A) \bullet A) \bullet B) \bullet B$$

is not in ΣL-form. Applying percolation to it results in

$$P_\Sigma(B_\Sigma(\sigma d)) = g(f(b \bullet A) \bullet A, g(f(b \bullet A) \bullet A, f(b \bullet A) \bullet A) \bullet A) \bullet A.$$

□

5.4 Dynamic Equations

In this section, we briefly define dynamic equations which are used in the next chapter. We return to consider dynamic equations and rewrite rules in more detail in Chapter 7.

Equations are constructed from dynamic terms in a similar manner to equations between ordinary terms described in chapter 4 above.

Definition 5.46. A dynamic equation is a quadruple consisting of a set of variables Y, a pair of dynamic terms $d_1, d_2 \in \overline{\mathcal{D}}_{\mathcal{S}}(Y)$, and a set of sorts S such that $\mathcal{DS}(d_1) \cup \mathcal{DS}(d_2) \subseteq S$. This quadruple is usually written $\forall Y.d_1 =_S d_2$. A dynamic equation is well-sorted if $d_1, d_2 \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$, and $S \geq \mathcal{SS}(d_1)$.

If the set of sorts is a singleton, say $S = \{A\}$, then for notational convenience we write $l =_A r$. If $\mathcal{DS}(l) = \mathcal{DS}(r) = S$, then we can omit the subscripted sort set.

The semantics of dynamic equations is given by a translation into equations over $\overline{T}_{\overline{\Sigma}}(\mathcal{X})$. In a dynamic equation, the two dynamic terms are the representation of the terms to be made equal, and the set of sorts represents the sorts in which the equality holds. Thus we can convert a dynamic equation into an equivalent set of sorted equations. We use Φ to represent the extension of ϕ to sets of dynamic equations.

$$\Phi(\forall Y.l =_S r) = \{\forall Y.\phi(l) =_s \phi(r) | s \in S\}$$

This notion is extended in the obvious fashion to sets of dynamic equations, and to relations over dynamic terms to give sets of standard equations and relations over $\overline{T}_{\overline{\Sigma}}(\mathcal{X})$ respectively.

We can also make a canonical conversion of a Σ-equation into a dynamic equation.

Definition 5.47. The least dynamic equation for a given Σ-equation $\forall Y.l =_s r$, denoted $\Psi(\forall Y.l =_s r)$ is given by $\forall Y\, L(l) =_S L(r)$ where $S = M_\Sigma(\mathcal{LS}(l) \cup \mathcal{LS}(r) \cup \{s\})$.

This definition, extended to sets of equations, allows the conversion of specifications over

$\overline{T}_{\overline{\Sigma}}(\mathcal{X})$ into systems of dynamic equations.

Definition 5.48. Given an order-sorted specification $\mathcal{S} = (\Sigma, E)$, the *canonical set* of well-sorted dynamic equations is given by $\Psi(E) = \{\Psi(e) | e \in E\}$.

Thus $(\Sigma, \Psi(E))$ is the canonical interpretation of a specification (Σ, E) using dynamic equations.

Chapter 6

Dynamic Matching and Unification

In this chapter we discuss the concepts of Matching and Unification within the context of dynamic sorts. Different definitions of these two operations can be given which have different properties. We discuss some of these definitions, giving their strengths and weaknesses and the relations between them. In order to keep this presentation of matching and unification as simple as possible, we do not consider the case of matching and unification modulo an equational theory.

6.1 Dynamic Matching

Matching determines whether one term is an instance of another. In this section we give several definitions of a dynamic match, and discuss their relative merits and properties.

6.1.1 Semantic Matching

Semantic matching is the most general match on well-sorted terms. This definition is similar to (Σ, R)-matching given in [Wer93], couched in a dynamic framework.

Definition 6.1. Semantic Matching

A dynamic term p (the *pattern*) *semantically matches* dynamic term t (the *target*) if and only if there exists a well-sorted dynamic substitution σ such that $\sigma p \simeq t$.

In general it is *not* the case that $\sigma p = t$ since the dynamic sorts of the two terms may differ, but the semantic match occurs on the underlying terms and the matching substitution is well-sorted. However, in general it is not decidable whether such a match exists. If it was, the least semantic sort of every term would be decidable.

Theorem 6.2. Semantic matching is not decidable.

Proof (6.2). Assume semantic matching is decidable, then given an arbitrary dynamic term d, for each sort $s \in S_\Sigma$ match $x : s$ to d. If it matches, the substitution $\{x \mapsto d \bullet s\}$ is well-sorted and hence $d \bullet s \in \mathcal{D}_\mathcal{S}(\mathcal{X})$. Hence this method decides which sorts are valid for $\phi(d)$. This contradicts the undecidability of sorting in Theorem 4.40. □

It is thus desirable to use a purely syntactic definition of matching, which exploits the dynamic sort. There are several approaches to this.

6.1.2 Weak Matching

We retain the property that underlying forms remain the same, but loosen the restriction that the matching substitution must be well-sorted.

Definition 6.3. Weak Dynamic Matching

A dynamic term $p \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ *weak matches* $t \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ if there exists a dynamic substitution σ such that $\sigma p \simeq t$. If σ is a weak matching substitution for p onto t we write $p \sqsubseteq^W_\sigma t$.

In general, there is more than one weak match for each pattern and target. However, we can give a definition of a 'canonical' match. This match represents the 'best' match in that it preserves the maximum amount of sort information available in the target term alone, without using external deductions as used in semantic matching.

Definition 6.4. A match κ of p on t is *canonical* if for all paths $q \in O(p)$ such that $p|_q \in \mathcal{X}$ then $\kappa p|_q = B_\Sigma(t)|_q$.

Intuitively, the distinguishing feature of the canonical weak match is that it is the minimal match (in the $\trianglerighteq$ ordering) such that the maximum amount of known sort information of the pattern is preserved in the match. By balancing the term first, we maximise the sort information used in the term, and also make $\simeq$-equivalent subterms identical, allowing the canonical match to be well defined.

Lemma 6.5. If $d, d' \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ are dynamic terms, $d \in \mathcal{D}_\mathcal{S}(\mathcal{X})$ is well-sorted and κ is the canonical weak match of d' on d, then κ is a well-sorted substitution.

Proof (6.5). If $x = d'|_q$, then $\mathcal{DS}(B_\Sigma(d|_q)) = \mathcal{DS}(\kappa d'|_q) \leq s(x)$ and $B_\Sigma(d|_q) \in \mathcal{D}_\mathcal{S}(\mathcal{X})$. □

Lemma 6.6. If σ is a weak match of p onto t, and if $\forall q \in O(p)$ such that $p|_q \in \mathcal{X}$, $\sigma p|_q \trianglerighteq t|_q$ then $\sigma \trianglerighteq \kappa$, where κ is the canonical weak match of p on t.

Proof (6.6). $\sigma p|_q \trianglerighteq t|_q \trianglerighteq B_\Sigma(t)|_q = \kappa p|_q$. □

Lemma 6.7. Given a canonical match κ of p on t and t is balanced, then for all paths $q \in O(p)$ such that $p|_q \in \mathcal{X}$, $\kappa p|_q = t|_q$.

Proof (6.7). $\kappa p|_q = B_\Sigma(t)|_q = t|_q$. □

Example 6.8. Given the specification:

Sorts:	$A\ B$
Subsorts:	$A \leq B$
Operators:	$a : \rightarrow A \qquad b : \rightarrow B$
	$f : B\ B \rightarrow B$
Equations:	$a = b$

If the pattern $f(x : B, x : B)\bullet B$ is matched onto $f(b\bullet A, b\bullet B)\bullet B$, $\{x \mapsto b\bullet B\}$ consti-

tutes a weak match, and $\{x \mapsto b \bullet A\}$ is a canonical weak-matching substitution since $B_\Sigma(f(b \bullet A, b \bullet B) \bullet B) = f(b \bullet A, b \bullet A) \bullet B$. □

However, there is not necessarily a canonical match, even in cases where there is a weak match and with balanced terms.

Example 6.9. Continuing with Example 6.8, there is a weak match between the dynamic terms $x : A$ and $b \bullet B$, $\{x \mapsto b \bullet A\}$, but no canonical weak match. However, the same substitution does form a canonical weak dynamic match between $x : A$ and $b \bullet A$, which is also a well-sorted dynamic term. □

Although $\{x \mapsto b \bullet A\}$ is well-sorted, it is not a canonical match since it cannot be derived from the target term alone.

Canonical weak matching is decidable and the set of rules in Figure 6.1 gives an algorithm for generating a canonical match on balanced terms. To determine the canonical match, the target term is first balanced and then the matching algorithm is applied. The symbol □ is used to represent failure. These rules are applied to sets of equations between dynamic terms of the form $d_1 \stackrel{?}{=} d_2$.

Before proving these rules correct, we give some auxiliary definitions.

Definition 6.10. A weak match for a set of equations $\{d_1 \stackrel{?}{=} e_1, \ldots, d_n \stackrel{?}{=} e_n\}$, is a dynamic substitution σ such that $\forall i \cdot \sigma d_i \simeq e_i$, and is canonical if for all $x : s \in \mathcal{X}$ and paths p such that $d_i|_p = x : s$ then $B_\Sigma(e_i)|_p = \sigma d_i|_p$.

A set of equations $E = \{d_1 \stackrel{?}{=} e_1, \ldots, d_n \stackrel{?}{=} e_n\}$ is in *normal form* if $\forall i \cdot d_i \in \mathcal{X}, \mathcal{DS}(e_i) \leq \mathcal{DS}(d_i)$ and $i \neq j \Rightarrow d_i \neq d_j$.

If $E = \{x_1 \stackrel{?}{=} e_1, \ldots, x_n \stackrel{?}{=} e_n\}$ is in normal form then $\sigma = \{x_1 \mapsto B_\Sigma(e_1), \ldots, x_n \mapsto B_\Sigma(e_n)\}$ forms a canonical weak match for E.

Lemma 6.11. The rules in Figure 6.1, applied to a balanced target term, terminate with either failure or in a normal form.

(1) Weak Decomposition.

$$\frac{\{f(d_1,\ldots,d_n)\bullet S \stackrel{?}{=} f(d'_1,\ldots,d'_n)\bullet S'\} \cup E}{\{d_1 \stackrel{?}{=} d'_1,\ldots,d_n \stackrel{?}{=} d'_n\} \cup E} \text{ if } f \in \mathcal{F}_\Sigma$$

(2) Conflict.

$$\frac{\{f(d_1,\ldots,d_n)\bullet S \stackrel{?}{=} g(d'_1,\ldots,d'_m)\bullet S'\} \cup E}{\square} \text{ if } f,g \in \mathcal{F}_\Sigma \text{ and } f \neq g$$

(3) Coalesce.

$$\frac{\{x : s \stackrel{?}{=} d, x : s \stackrel{?}{=} d\} \cup E}{\{x \stackrel{?}{=} d\} \cup E} \; \mathcal{DS}(d) \leq s$$

(4) Failure.

$$\frac{\{d \stackrel{?}{=} x : s\} \cup E}{\square} \text{ if } d \notin \mathcal{X}$$

(5) Sort Clash.

$$\frac{\{x : s \stackrel{?}{=} d\} \cup E}{\square} \text{ if } \mathcal{DS}(d) \not\leq s$$

(6) Clash.

$$\frac{\{x : s \stackrel{?}{=} d_1, x : s \stackrel{?}{=} d_2\} \cup E}{\square} \text{ if } d_1 \neq d_2$$

Figure 6.1: Rules for Canonical Weak Dynamic Matching on Balanced terms.

Proof (6.11). Clearly rules 2, 4, 5 and 6 terminate with failure. Given a finite set of equations E, let $M(E)$ be the multiset of the heights of the terms in E. Let $\succ$ be the well-founded multiset extension of the standard ordering on the natural numbers. The non-failure rules strictly reduce $M(E)$ in this ordering. The weak decomposition rule removes terms of heights N and M respectively and inserts terms of heights at most $N-1$ and $M-1$ respectively, reducing $M(E)$. The Coalesce rule removes terms of heights 1 and N. Consequently, the rules must terminate.

To show the second part, assume the rules terminate with a set of equations E which are not in normal form. Thus there exists an equation $d_i \stackrel{?}{=} e_i \in E$ such that either $d_i \notin \mathcal{X}$, in which case one of rules 1, 2, or 4 will apply, or $\mathcal{DS}(e_i) \not\leq \mathcal{DS}(d_1)$ and rule 5 applies or $d_i \in \{d_j\}_{j \in 1..i-1,i+1..n}$, in which case rule 3, or 6 applies. This contradicts the assumption of termination, and so E must be in normal form. $\square$

Lemma 6.12. The rules of Figure 6.1, applied to a balanced target term, either preserve weak dynamic matches, or if they fail, then no weak match exists.

Proof (6.12). Consider each rule in turn.

(1) Weak Decomposition. If σ is a weak match for $\{f(d_1,\ldots,d_n)\bullet S \stackrel{?}{=} f(d'_1,\ldots,d'_n)\bullet S'\} \cup E$, then $\sigma(f(d_1,\ldots,d_n)\bullet S) \simeq f(d'_1,\ldots,d'_n)\bullet S'$, hence $f(\sigma d_1,\ldots,\sigma d_n)\bullet S \simeq f(d'_1,\ldots,d'_n)\bullet S'$, so σ is a weak match for $\{d_1 \stackrel{?}{=} d'_1,\ldots,d_n \stackrel{?}{=} d'_n\} \cup E$. Similarly, if σ is a weak match for $\{d_1 \stackrel{?}{=} d'_1,\ldots,d_n \stackrel{?}{=} d'_n\} \cup E$, then it is a weak match for $\{f(d_1,\ldots,d_n)\bullet S \stackrel{?}{=} f(d'_1,\ldots,d'_n)\bullet S'\} \cup E$.

(2) Conflict. If $\{f(d_1,\ldots,d_n)\bullet S \stackrel{?}{=} g(d'_1,\ldots,d'_m)\bullet S'\} \cup E$ if $f,g \in \mathcal{F}_\Sigma$ and $f \neq g$ then there can be no substitution σ such that $\sigma(f(d_1,\ldots,d_n)\bullet S)) \simeq g(d'_1,\ldots,d'_m)$.

(3) Coalesce. Clearly σ is a match for $\{x : s \stackrel{?}{=} d, x : s \stackrel{?}{=} d\} \cup E$, if and only if it is also a match for $\{x : s \stackrel{?}{=} d\} \cup E$.

(4) Failure. If $\{d \stackrel{?}{=} x : s\} \cup E$, for some $d \notin \mathcal{X}$, then there is no σ such that $\sigma(\phi(d)) = x : s$.

(5) Sort Clash. Given $\{x : s \stackrel{?}{=} d\} \cup E$ and $\mathcal{DS}(d) \not\leq s$, if σ is a weak matching substitution, then $d = \sigma x$, by definition of weak match, and $\mathcal{DS}(\sigma x) \leq s$, by well formed substitution. Hence $\mathcal{DS}(d) \leq s$, which contradicts our assumption, and thus no such σ can exist.

(6) Clash Given $\{x : s \stackrel{?}{=} d_1, x : s \stackrel{?}{=} d_2\} \cup E$ where $d_1 \neq d_2$, there is no substitution σ such that $d_1 = \sigma(x) = d_2$. □

Theorem 6.13. Applying the rules in Figure 6.1 to an initial set $\{d \stackrel{?}{=} B_\Sigma(d')\}$, where d' is a target term, either fails or terminates with a set of normal forms which have a canonical weak dynamic match σ for d on d'.

Proof (6.13). Immediate from Lemmas 6.11 and 6.12. □

6.1.3 Strong Matching

The idea encapsulated in canonical weak matching is to check from the dynamic sorts of the pattern term, whether there already exists a dynamic term of the right sort at the same path in the target. However, while the above lemma tells us that a dynamic term d' weakly matching onto a well-sorted dynamic term results in a well-sorted *substitution*, we cannot guarantee that the resulting *instance* of d' will be well-sorted. Consider the following example.

Example 6.14. Extend Example 6.8 with the operator g with rank $g : B{\rightarrow}B$, then $g(x : A){\bullet}A \in \mathcal{ID}_{\mathcal{S}}(\mathcal{X})$. This weak matches with the well-sorted term $g(a{\bullet}A){\bullet}B$, with well-sorted substitution $\{x \mapsto a{\bullet}A\}$, but $\sigma(g(x : A){\bullet}A) = g(a{\bullet}A){\bullet}A \in \mathcal{ID}_{\mathcal{S}}(\mathcal{X})$. □

To exclude this possibility, we extend the idea of checking the dynamic sorts of the pattern to apply to *all paths*, rather than just those at variables, giving us a *strong dynamic match.*

Definition 6.15. Upward Strong Dynamic Matching
A dynamic term p *upward strong matches* a dynamic term t, written $p \sqsubseteq^{U} t$, if and only if there exists a dynamic substitution σ such that $\sigma p \simeq t$, and for all paths $q \in O(p)$ $\mathcal{DS}(B_{\Sigma}(t)|_q) \leq \mathcal{DS}(\sigma p|_q)$.

The upward strong match can be succinctly expressed using the approximation relation.

Lemma 6.16. If $p \in \overline{\mathcal{D}}_{\Sigma}(\mathcal{X})$ upward strong matches a $t \in \overline{\mathcal{D}}_{\Sigma}(\mathcal{X})$ via a matching dynamic substitution σ then $\sigma p \trianglerighteq B_{\Sigma}(t)$. If t is balanced, then $\sigma p \trianglerighteq t$.

Proof (6.16). Immediate from the definitions of upward strong matching and approximation relation. □

Again, there can be many upward strong matches, and the definition of canonical weak match given in Definition 6.4, is used for canonical upward strong match. In upward strong matching, the canonical match has the following minimal property.

Lemma 6.17. If σ upward strong matches p onto t, then $\sigma \trianglerighteq \kappa$.

Proof (6.17). If $q \in O(p)$ and $p|_q \in \mathcal{X}$, then $\sigma p|_q \trianglerighteq B_\Sigma(t)|_q = \kappa p|_q$. □

For completeness, we also consider the dual matching algorithm which checks that the pattern is less than the target at all paths.

Definition 6.18. Downward Strong Dynamic Matching
A dynamic term p *downward strong matches* a balanced dynamic term t written $p \sqsubseteq^V t$, if and only if there exists a dynamic substitution σ such that $\sigma p \simeq t$ and for all paths $q \in O(p)$ $\mathcal{DS}(B_\Sigma(t)|_q) \geq \mathcal{DS}(\sigma p|_q)$.

Downward strong matching can also be expressed using the approximation relation.

Lemma 6.19. A dynamic term $p \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ downward strong matches a d-term $t \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ via a matching dynamic substitution σ if and only if $B_\Sigma(t) \trianglerighteq \sigma p$. If t is balanced, then $\sigma p \trianglelefteq t$.

Proof (6.19). Immediate from the definitions of downward strong matching and the approximation relation. □

Again the definition of canonical downward strong match can be given, which has the following maximal property.

Lemma 6.20. If σ downward strong matches p onto t, then $\sigma \trianglelefteq \kappa$.

Proof (6.20). If $q \in O(p)$ and $p|_q \in \mathcal{X}$, then $\sigma p|_q \trianglelefteq B_\Sigma(t)|_q = \kappa p|_q$. □

The strong matches form a proper subclass of weak matches.

Example 6.21. Extend Example 6.14 with the additional rank $g : A \rightarrow A$, then the dynamic term $g(x_A) \bullet A$ will weak match onto $g(b \bullet A) \bullet B$, but not upward strong match. □

Example 6.22. In Example 6.8, $\{x \mapsto b \bullet B\}$ also forms an Upward Strong match of $f(x : B, x : B) \bullet B$ onto $f(b \bullet A, b \bullet B) \bullet B$, and $\{x \mapsto b \bullet A\}$ is the canonical upward strong matching substitution, as balancing the target term results in $f(b \bullet A, b \bullet A) \bullet B$. □

The weak matching algorithm for canonical matching on balanced terms needs little modification to perform these checks, modifying the decomposition rule, and adding a new failure mode. The rules which replace the weak decomposition rule are given in Figures 6.2 and 6.3.

(1a) Upward Strong Decomposition.

$$\frac{\{f(d_1, \ldots, d_n) \bullet S \stackrel{?}{=} f(d'_1, \ldots, d'_n) \bullet S'\} \cup E\}}{\{d_1 \stackrel{?}{=} d'_1, \ldots, d_n \stackrel{?}{=} d'_n\} \cup E} \quad \begin{array}{l} \text{if } f \in \mathcal{F}_\Sigma \\ \text{and } S' \leq S \end{array}$$

(1b) Upward Strong Decomposition Failure.

$$\frac{\{f(d_1, \ldots, d_n) \bullet S \stackrel{?}{=} f(d'_1, \ldots, d'_n) \bullet S'\} \cup E\}}{\square} \quad \begin{array}{l} \text{if } f \in \mathcal{F}_\Sigma \\ \text{and } S' \not\leq S \end{array}$$

Figure 6.2: Additional Rules for Upward Strong Dynamic Matching.

We give details of upward strong matching, using some auxiliary definitions.

Definition 6.23. An *upward strong match* (downward strong match) for a set of equations $\{d_1 \stackrel{?}{=} e_1, \ldots, d_n \stackrel{?}{=} e_n\}$, is a dynamic substitution σ such that $\forall i \cdot \sigma d_i \simeq e_i$, and $\forall i$, $\forall p \in O(d_i)$ then $\mathcal{DS}(B_\Sigma(e_i)|_p) \leq \mathcal{DS}(\sigma d_i|_p)$ (respectively $\mathcal{DS}(\sigma d_i|_p) \leq \mathcal{DS}(e_i|_p)$).

If $E = \{x_1 \stackrel{?}{=} e_1, \ldots, x_n \stackrel{?}{=} e_n\}$ is in normal form then $\sigma = \{x_1 \mapsto B_\Sigma(e_1), \ldots, x_n \mapsto B_\Sigma(e_n)\}$ forms a canonical (upward/downward) strong match for E.

Lemma 6.24. The rules in Figure 6.1, excluding the weak decomposition rule including the rules in Figure 6.2, applied to a set of equations E terminate with either failure or with a normal form.

Proof (6.24). Given the same measure $M(E)$ as the termination proof for weak matching above, note that the strong decomposition rule removes terms of heights N and M respectively and replaces them with terms at most $N - 1$ and $M - 1$. The rest of the proof follows the proof of termination for weak matching. □

Lemma 6.25. The rules in Figure 6.1, excluding the weak decomposition rule including the rules in Figure 6.2, applied to a set of equations E either preserve upward strong matches, or if they fail, then no upward strong match exists.

Proof (6.25). Consider the new rules in turn.

(1a) Upward Strong Decomposition. If σ is a upward strong match for $\{f(d_1,\ldots,d_n)\bullet S = f(d'_1,\ldots,d'_n)\bullet S'\}\cup E\}$ where $f \in \mathcal{F}_\Sigma$ and $S' \leq S$ then $\sigma(f(d_1,\ldots,d_n)\bullet S) \simeq f(d'_1,\ldots,d'_n)\bullet S'$, hence $f(\sigma d_1,\ldots,\sigma d_n)\bullet S \simeq f(d'_1,\ldots,d'_n)\bullet S'$. Hence σ is an upward strong match for $\{d_1 \stackrel{?}{=} d'_1,\ldots,d_n \stackrel{?}{=} d'_n\}\cup E$. Similarly, if σ is an upward strong match for $\{d_1 \stackrel{?}{=} d'_1,\ldots,d_n \stackrel{?}{=} d'_n\}\cup E$ then it will be an upward strong match for $\{f(d_1,\ldots,d_n)\bullet S \stackrel{?}{=} f(d'_1,\ldots,d'_n)\bullet S'\} \cup E$, as $S' \leq S$.

(1b) Upward Strong Decomposition Failure. If σ was an upward strong match for $\{f(d_1,\ldots,d_n)\bullet S = f(d'_1,\ldots,d'_n)\bullet S'\} \cup E\}$ where $f \in \mathcal{F}_\Sigma$, and $S' \not\leq S$, then for the path $\epsilon \in O(\sigma(f(d_1,\ldots,d_n)\bullet S))$ then $S' \not\leq S$, which contradicts the definition of upward strong match.

The rest of the proof follows that of the soundness of weak dynamic matching. □

Theorem 6.26. The rules in Figure 6.1, excluding the weak decomposition rule but including the rules in Figure 6.2, applied to a initial set $\{d \stackrel{?}{=} d'\}$, where d' is a balanced target term, either fails or terminates with a set of normal forms which have a canonical upward strong dynamic match σ for d on d'.

Proof (6.26). Immediate from Lemmas 6.24 and 6.25, noting that the canonical strong match of a set of terms in normal form is extracted. □

Similarly, the rules for downward strong dynamic matching are given in Figure 6.3.

The proofs of termination and correctness of this form of matching are similar to those for upward strong dynamic matching and are not repeated. Both forms of strong matching are compositional.

(1a) Downward Strong Decomposition.

$$\frac{\{f(d_1,\ldots,d_n)\bullet S \stackrel{?}{=} f(d'_1,\ldots,d'_n)\bullet S'\} \cup E\}}{\{d_1 \stackrel{?}{=} d'_1,\ldots,d_n \stackrel{?}{=} d'_n\} \cup E} \quad \begin{array}{l}\text{if } f \in \mathcal{F}_\Sigma \\ \text{and } S \leq S'\end{array}$$

(1b) Downward Strong Decomposition Failure.

$$\frac{\{f(d_1,\ldots,d_n)\bullet S \stackrel{?}{=} f(d'_1,\ldots,d'_n)\bullet S'\} \cup E\}}{\square} \quad \begin{array}{l}\text{if } f \in \mathcal{F}_\Sigma \\ \text{and } S \not\leq S'\end{array}$$

Figure 6.3: Additional Rules for Downward Strong Dynamic Matching.

Lemma 6.27. Let d, d_1, $d_2 \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ and d_1, d_2 are balanced. Then d upward (downward) strong matches d_1 via θ and d_1 upward (downward) strong matches d_2 via σ if and only if d upward (downward) strong matches d_2 via $\sigma\theta$.

Proof (6.27). For upward, as $\theta d \trianglerighteq d_1$ and $\sigma d_1 \trianglerighteq d_2$ this is immediate from Lemma 5.31. For downward, as $d \trianglerighteq \theta d_1$ and $d_1 \trianglerighteq \sigma d_2$ this is immediate from Lemma 5.31. □

Upward strong dynamic matching has one important property: an (ill-sorted) pattern term will not upward strong match a well-sorted target and result in an ill-formed instance.

Lemma 6.28. Let $d \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ and $d' \in \mathcal{D}_\mathcal{S}(\mathcal{X})$. If $d \sqsubseteq^U_\sigma d'$, then $\sigma d \in \mathcal{D}_\mathcal{S}(\mathcal{X})$.

Proof (6.28). Immediate from Lemma 5.33. □

The definition of upward strong matching is not however, powerful enough to exclude well-sorted terms matching onto ill-sorted ones; if it were the case we could decide semantic matching.

Example 6.29. Continuing Example 6.21 with the extra constant $c : \rightarrow B$ we see that the ill-sorted term $f(c\bullet B)\bullet A$ is upward strong matched by the well-sorted term $f(x : B)\bullet B$ with matching substitution $\{x \mapsto c\bullet B\}$. □

This may mean in the definition of rewriting we are able to rewrite ill-sorted terms as well as well-sorted ones using well-sorted rewrite rules.

Downward strong matching has a different property: well-sorted terms only match onto well-sorted terms, that is it is not possible to downward strong match a well-sorted pattern with an ill-sorted target.

Lemma 6.30. Let $d \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$ and $d' \in \overline{\mathcal{D}}_{\Sigma}(\mathcal{X})$. If $d \sqsubseteq^{V}_{\sigma} d'$, and σ is a well-sorted dynamic substitution, then $d' \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$.

Proof (6.30). Immediate from Lemmas 6.19 and 5.21. □

6.1.4 Some Comments on Matching

It is interesting to contrast the approach taken here with that of Hintermeier, Kirchner and Kirchner [HKK93]. They define a variety of matching which they call *strict* matching. In the notation given in this thesis, this becomes:

Definition 6.31. Strict Dynamic Matching
A dynamic term p *strict matches* a dynamic term t written $p \sqsubseteq^{St} t$, if and only if there exists a dynamic substitution σ such that $p \simeq t$ and for all paths $q \in O(p)$ $\mathcal{DS}(t|_q) = \mathcal{DS}(\sigma(p)|_q)$.

This variety of matching is syntactic on dynamic sort sets as well as operators, since the dynamic sorts at all nodes must be equivalent (that is $S \equiv S'$ if and only if $S \leq S'$ and $S' \leq S$). Strict matching thus ensures that $\sigma p = t$, while strong matching only ensures $\sigma p \trianglerighteq t$, and weak matching only ensures that $\phi(\sigma p) = \phi(t)$. It might seem that strict matching is simpler and more intuitive than strong or weak. However, the use of strict matching leads to a less powerful rewriting system. If only strict matches can be made, then extra rules are needed to add sorts explicitly to the dynamic sorts of terms, whilst strong matches 'code up' classes of strict matches within one match. In [HKK93], this question is partly answered by the use of sort variables in dynamic sorts.

This difference in approaches is also influenced by a difference in motivation. In [HKK93], the semantic framework is Galactic algebras and the computation is designed to answer questions on the algebra, which might include type formulae. In this thesis, order-sorted

specifications are defined in a conventional way and only equalities between conventional terms are tested, not in which sort that they are valid. The dynamic framework is a means to this end rather than an end in itself.

Thus we have four candidates for matching between dynamic terms; weak, upward strong, downward strong, and strict dynamic matching. If we represent the matching definitions by the pairs of well-sorted terms which they match:

$Subst_X =_{def} \{(d, d') | d, d' \in \mathcal{D}_{\mathcal{S}}(\mathcal{X}),\ d, d' \text{ in SL-form and } \sigma \text{ is a } X\text{-match such that } d \sqsubseteq^X d'\}$,

then we have the relationship between matches as shown in Figure 6.4.

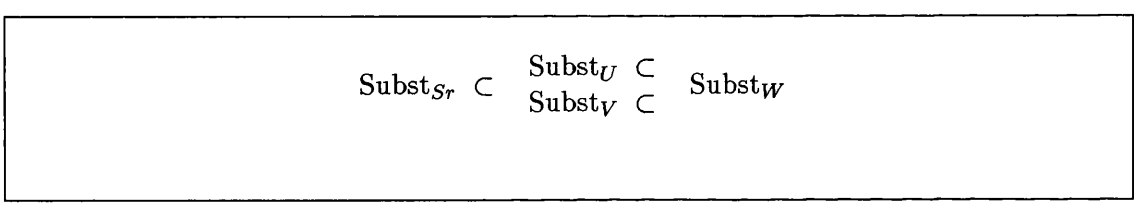


Figure 6.4: The relationship between well-sorted matches

The appropriate match depends on the properties of the match required. The weak match has fewer restrictions. However, it will only be sound in cases where both the pattern and target terms are well-sorted. If the pattern term (which may be one side of an equality) is not well-sorted then under weak matching the resulting instantiation may not be well-sorted and so replacing equals by equals will not be sound. The upward strong match leads to a less powerful rewrite relation, but allows us to use ill-sorted terms in our equalities; there may occur circumstances where this is appropriate. Thus the choice of matching rewriting relation generated by that match depends on whether we wish to allow ill-sorted terms.

6.2 Dynamic Unification

Semantic dynamic unification has a similar definition to that of semantic matching.

Definition 6.32. Semantic Dynamic Unification

Given a specification $\mathcal{S} = (\Sigma, E)$ and dynamic terms $d_1, d_2 \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$, then the terms are *unifiable*, if there exists a *well-sorted* dynamic substitution σ such that $\sigma d_1 \simeq \sigma d_2$. σ is then known as the semantic unifier of d_1 and d_2.

As Werner [Wer93] points out in a different context, the semantic dynamic unification problem is closely related to E-Unification. It is not in general possible to decide whether a dynamic term is well-sorted, as it requires the consideration of the entire equational theory. Thus it is not possible, in general, to decide whether a dynamic substitution is well-sorted, and similarly to semantic matching, semantic dynamic unification is not decidable. In cases where it is, it is in general infinitary as the following example [Wer93] demonstrates.

Example 6.33. [Wer93] Given the specification:

Sorts:	$Pos\ Nat$
Subsorts:	$Pos \leq Nat$
Operators:	$0 : \rightarrow Nat \qquad s : Nat \rightarrow Pos$
	$mod_2 : Nat \rightarrow Nat$
Equations:	$mod_2(0 \bullet Nat) \bullet Nat =_{Nat} 0 \bullet Nat$
	$mod_2(s(0 \bullet Nat) \bullet Pos) \bullet Nat =_{Pos} s(0 \bullet Nat) \bullet Pos$
	$mod_2(s(s(x_{Nat}) \bullet Pos) \bullet Pos) \bullet Nat =_{Nat} mod_2(x_{Nat}) \bullet Nat$

Consider unifying $y_{Pos} \stackrel{?}{=} mod_2(z_{Nat}) \bullet Nat$. Then each of:

$$\begin{aligned} \{y_{Pos} &\mapsto mod_2(s^{2n+1}(0 \bullet Nat) \bullet^{2n+1} Pos) \bullet Pos \\ z_{Nat} &\mapsto s^{2n+1}(0 \bullet Nat) \bullet^{2n+1} Pos\} \end{aligned}$$

are unifiers since for each n, $mod_2(s^{2n+1}(0 \bullet Nat) \bullet^{2n+1} Pos) \bullet Pos \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$, [1] as it is equal to $s(0 \bullet Nat) \bullet Pos$, and none of them is subsumed by any other well-sorted unifiers. $\{y_{Pos} \mapsto mod_2(z_{Nat}) \bullet Nat\}$ is not a dynamic substitution, and thus cannot subsume these unifiers. Another candidate substitution might be $\{y_{Pos} \mapsto mod_2(z_{Nat}) \bullet Pos\}$. However, this is not a well-sorted substitution as $mod_2(z_{Nat}) \bullet Pos \notin \mathcal{D}_{\mathcal{S}}(\mathcal{X})$. □

Nevertheless, dynamic unification can be decided and is finitary if we relax the restriction

[1] These unifiers are similar to recurrence terms [CH91a]. It is conjectured that for a suitable class of problems, such infinite sets of unifiers can be characterised by recurrence terms.

to well-sorted substitutions. Similarly to matching, there are several forms which dynamic unification can take. Two general definitions of the unification of dynamic terms are given.

Definition 6.34. Loose Dynamic Unification
Given a specification $S = (\Sigma, E)$, $d_1, d_2 \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ are *loosely unifiable* if there exists a dynamic substitution σ such that $\sigma d_1 \simeq \sigma d_2$. σ is the loose unifier of d_1 and d_2.

Definition 6.35. Strict Dynamic Unification
Given a specification $S = (\Sigma, E)$, $d_1, d_2 \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ are *strictly unifiable* if there exists a dynamic substitution σ such that $\sigma d_1 = \sigma d_2$. σ is the strict unifier of d_1 and d_2.

The loose unification of dynamic terms requires the underlying terms to be the same under the unifying substitution, whilst strict unification requires that the dynamic terms to be the same, including their sort components, which is analogous to strict matching. Loose unification is analogous to weak and both forms of strong matching. However, the strong matching algorithms are distinguished by the existence of particular *Unified Terms.*

Definition 6.36. Given matching algorithm X and (loosely) unifiable d_1, d_2 with substitution σ an *X-Unified Term* for d_1 and d_2 is a term $u \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ such that $d_1 \sqsubseteq^X_\sigma u$ and $d_2 \sqsubseteq^X_\sigma u$. u is *X-maximal* if for all v such that v is an X-unified term for d_1 and d_2, then $v \trianglelefteq u$, and *X-minimal* if for all v such that v is an X-unified term for d_1 and d_2, then $v \trianglerighteq u$.

It is these unified terms that characterise dynamic unification in contrast to standard unification theory. In standard theory, the unified term is trivial; it is just the query terms with the unifying substitution applied. However, in dynamic unification, this unified term is not trivial. Indeed, in general there is a set of such terms for each unifier, and particular unified terms are of special interest.

We characterise the unifications which are associated with strong matches by the existence of particular unified terms.

Lemma 6.37. If $d_1, d_2 \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ are loosely unifiable with substitution σ, then:

1. $\sigma d_1 \wedge \sigma d_2$ is the unique *Upward Strong maximal Unified Term* ($\mathrm{USMUT}_{d_1 \stackrel{?}{=} d_2}(\sigma)$) for d_1 and d_2.
2. $\sigma d_1 \vee \sigma d_2$ is the unique *Downward Strong minimal Unified Term* ($\mathrm{DSMUT}_{d_1 \stackrel{?}{=} d_2}(\sigma)$) for d_1 and d_2.

Proof (6.37). Proof of part 1. As, if d_1 and d_2 loosely unify, $\sigma d_1 \trianglerighteq \sigma d_1 \wedge \sigma d_2$ and $\sigma d_2 \trianglerighteq \sigma d_1 \wedge \sigma d_2$, and by definition of upward strong match $d_1 \sqsubseteq^U_\sigma \sigma d_1 \wedge \sigma d_2$ and $d_2 \sqsubseteq^U_\sigma \sigma d_1 \wedge \sigma d_2$, which thus forms an upward strong unified term. If for some u, $d_1 \sqsubseteq^U_\sigma u$ and $d_2 \sqsubseteq^U_\sigma u$, then $\sigma d_1 \trianglerighteq u$ and $\sigma d_2 \trianglerighteq u$. Then, by Lemma 5.21, part 5, $\sigma d_1 \wedge \sigma d_2 \trianglerighteq u$.

Proof of part 2 is similar. □

Example 6.38. Given the following Order-Sorted Specification.

Sorts:	$S\ A\ B\ C\ D\ T$	
Subsorts:	$T \leq C \leq A \leq S, T \leq D \leq B \leq S$	
Operators:	$a : \rightarrow A$	$b : \rightarrow B$
	$f : A\,B \rightarrow S$	$g : A \rightarrow C$
	$h : B \rightarrow D$	

The unifiers of:

$$f(g(a \bullet A) \bullet C, y : B) \bullet S \stackrel{?}{=} f(x : A, h(b \bullet B) \bullet D) \bullet S$$

include the following substitutions with their upward strong maximal unified terms:

$\{x \mapsto g(a \bullet \{\}) \bullet A, y \mapsto h(b \bullet \{\}) \bullet B\}$	$f(g(a \bullet A) \bullet C, h(b \bullet B) \bullet D) \bullet S$
$\{x \mapsto g(a \bullet \{\}) \bullet C, y \mapsto h(b \bullet \{\}) \bullet D\}$	$f(g(a \bullet A) \bullet C, h(b \bullet B) \bullet D) \bullet S$
$\{x \mapsto g(a \bullet S) \bullet C, y \mapsto h(b \bullet S) \bullet D\}$	$f(g(a \bullet A) \bullet C, h(b \bullet B) \bullet D) \bullet S$
$\{x \mapsto g(a \bullet A) \bullet C, y \mapsto h(b \bullet B) \bullet D\}$	$f(g(a \bullet A) \bullet C, h(b \bullet B) \bullet D) \bullet S$
$\{x \mapsto g(a \bullet C) \bullet C, y \mapsto h(b \bullet D) \bullet D\}$	$f(g(a \bullet C) \bullet C, h(b \bullet D) \bullet D) \bullet S$
$\{x \mapsto g(a \bullet T) \bullet C, y \mapsto h(b \bullet T) \bullet D\}$	$f(g(a \bullet T) \bullet C, h(b \bullet T) \bullet D) \bullet S$

□

In both strict and loose cases, we define a minimal complete set of unifiers.

Definition 6.39. The set of (loose/strict) unifiers of a pair of dynamic terms d_1, d_2, written $U_Y(d_1 \stackrel{?}{=} d_2)$, where Y is loose or strict, is the set of dynamic substitutions which are (loose/strict) unifiers of d_1, d_2. A *Complete* set of unifiers for a pair of dynamic terms d_1, d_2 on $Vars(d_1) \cup Vars(d_2) \subseteq \mathcal{W} \subseteq \mathcal{X}$, written $CSU_Y^{\mathcal{W}}(d_1 \stackrel{?}{=} d_2)$, is a set of dynamic substitutions such that:

1. $\forall \sigma \in CSU_Y^{\mathcal{W}} \cdot Im(\sigma) \cap \mathcal{W} = \emptyset$
2. $CSU_Y^{\mathcal{W}}(d_1 \stackrel{?}{=} d_2) \subseteq U(d_1 \stackrel{?}{=} d_2)$
3. $\forall \tau \in U(d_1 \stackrel{?}{=} d_2) \cdot \exists \sigma \in CSU_Y^{\mathcal{W}}(d_1 \stackrel{?}{=} d_2) \cdot \sigma \preceq^{\mathcal{W}} \tau$.

A complete set of unifiers is *minimal*, denoted $\mu CSU_Y^{\mathcal{W}}(d_1 \stackrel{?}{=} d_2)$, if $\forall \sigma, \rho \in \mu CSU_Y^{\mathcal{W}}(d_1 \stackrel{?}{=} d_2) \cdot \sigma \preceq^{\mathcal{W}} \rho \Rightarrow \sigma \equiv^{\mathcal{W}} \rho$. Such unifiers are known as *most-general.*

Lemma 6.40. If σ is a most general unifier, and ρ is a unifier of the same terms, then there is a π such that $\rho \trianglelefteq \pi\sigma$.

Proof (6.40). Immediate from the definition of $\preceq^{\mathcal{W}}$. □

The problem of unifying s and t becomes that of finding a minimal complete set of unifiers for a singleton set of equations, conventionally written $\{s \stackrel{?}{=} t\}$.

Example 6.41. Given the following order-sorted specification.

Sorts:	$A\ B$	
Subsorts:	$A \leq B$	
Operators:	$f : B \to B$	$g : B \to B$
	$a : \to A$	
Equations:	$g(x : A)\bullet B =_A x : A$	

A most general loose unifier of $f(g(x : A)\bullet B)\bullet B \stackrel{?}{=} f(y : B)\bullet B$ is $\sigma = \{y : B \mapsto g(x : A)\bullet B\}$.

Another unifier is $\sigma' = \{y : B \mapsto g(x : A)\bullet A\}$, and $\sigma \trianglerighteq \sigma'$. □

All most general loose unifiers are equivalent.

Lemma 6.42. If $\sigma, \rho \in \mu CSU^{\mathcal{W}}_{loose}(d_1 \stackrel{?}{=} d_2)$, then $\sigma \simeq \rho$.

Proof (6.42). By definition of loose unification, $\phi(\sigma d_1) = \phi(\sigma d_2)$ and $\phi(\rho d_1) = \phi(\rho d_2)$. Therefore, $\phi(\sigma)$ and $\phi(\rho)$ are unifiers of standard unsorted terms $\phi(d_1), \phi(d_2) \in \overline{T}_{\overline{\Sigma}}(\mathcal{X})$. By a well known theorem (for example [JD90]), unsorted unifiers are unique up to renaming, hence $\phi(\sigma) =_\alpha \phi(\rho)$, and $\sigma \simeq \rho$ up to renaming. □

Loose dynamic unification is unitary.

Theorem 6.43. If $\sigma, \rho \in \mu CSU^{\mathcal{W}}_{loose}(d_1 \stackrel{?}{=} d_2)$, then $\sigma = \rho$ up to renaming of variables.

Proof (6.43). Assume that there are $\sigma, \rho \in \mu CSU^{\mathcal{W}}_{loose}(d_1 \stackrel{?}{=} d_2)$ such that $\sigma \neq_\alpha \rho$. Then, by the previous lemma $\sigma \simeq \rho$. Consider the substitution $\tau = \sigma \vee \rho$; clearly, $\tau d_1 \simeq \sigma d_1 \simeq \sigma d_2 \simeq \tau d_2$ and thus $\tau \in U^{\mathcal{W}}_{loose}(d_1 \stackrel{?}{=} d_2)$. Since $\sigma \trianglelefteq \tau$ and $\rho \trianglelefteq \tau$, by definition σ and ρ are not most general, which contradicts the assumption. □

The unique member of $\mu CSU^{\mathcal{W}}_{loose}(d_1 \stackrel{?}{=} d_2)$, if it exists, is known as the *most general loose unifier* of d_1 and d_2.

Example 6.44. In Example 6.38, $\{x \mapsto g(a\bullet\{\})\bullet A, y \mapsto h(b\bullet\{\})\bullet B\}$ is the most general unifier. We also observe that $\{x \mapsto g(a\bullet A)\bullet C, y \mapsto h(b\bullet B)\bullet D\}$ is the *minimal* unifier (under the $\trianglerighteq$ ordering) with the *maximal* USMUT. □

When used in conjunction with upward strong matching, we have the following property on the upward strong maximal Unified Terms.

Lemma 6.45. If σ is the most general loose unifier for $d_1, d_2 \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, with u the upward strong maximal Unified Term for σ, and if $\rho \in U_{loose}(d_1 \stackrel{?}{=} d_2)$, with v the upward strong maximal Unified Term for ρ, then $\exists \pi \in \overline{DSubst}_\Sigma$ such that $v \trianglelefteq \pi u$.

Proof (6.45). σ is most general so there is a substitution π such that $\rho \trianglelefteq \pi\sigma$. $u = \sigma d_1 \wedge \sigma d_2$, therefore $\pi u = \pi(\sigma d_1 \wedge \sigma d_2) = (\pi\sigma d_1 \wedge \pi\sigma d_2) \trianglerighteq (\rho d_1 \wedge \rho d_2) = v$. □

Thus it can be seen that all unifiers of terms can be 'factored through' the most general loose unifier.

6.2.1 A Dynamic Unification Algorithm

A unification problem is described as a set of equations Γ and the aim is to transform this into a solved form, from which a unifying substitution can be trivially derived, whilst maintaining the set of minimal unifiers of the equations.

Definition 6.46. A dynamic equation is in *Solved Form* if it is of the form $x \stackrel{?}{=} d$ where x is a variable of sort s, and d is a dynamic term such that $s \in \mathcal{DS}(d)$ and $x \notin Vars(d)$. A set of equations $\Gamma = \{x_1 \stackrel{?}{=} d_1, \ldots, x_n \stackrel{?}{=} d_n\}$ is in solved form if each $x_i \stackrel{?}{=} d_i \in \Gamma$ is in solved form, and if $\forall i \neq j$, $x_i \neq x_j$.

Once a set of equations are all in solved form, then a loose unifying substitution can easily be derived.

Theorem 6.47. Given a set of equations $\Gamma = \{x_1 : s_1 \stackrel{?}{=} d_1, \ldots, x_n : s_n \stackrel{?}{=} d_n\}$ in solved form, the most general loose unifier $\mu CSU^{\mathcal{W}}(\Gamma)$ is given by $\sigma = \{x_1 \mapsto d_1 \downarrow \{\}\bullet\{s_1\}, \ldots, x_1 \mapsto d_1 \downarrow \{\}\bullet\{s_n\}\}$ where $\mathcal{W} = \{x_1, \ldots, x_n\}$.

Proof (6.47). Clearly σ is a loose unifier for Γ. Let τ be another unifier of Γ. Thus for all $x_i \stackrel{?}{=} d_i \in \Gamma$, $\tau x_i \simeq \tau d_i$.

If $\tau x_i \not\trianglelefteq d_i$, define λ to be $\lambda x = x$ if $x \in \mathcal{W}$, $\lambda x = \tau x$ if $x \notin \mathcal{W}$. Thus for all $x_i = d_i \in \Gamma$, $\tau x_i \trianglelefteq \lambda(\sigma x_i)$, and thus $\sigma \preceq^{\mathcal{W}} \tau$.

If $\tau x_i \simeq d_i$, then consider $p \in O(d_i)$. By definition of substitution, $\mathcal{DS}(\tau x_i|_\epsilon) \leq \mathcal{DS}(\sigma x_i|_\epsilon) = s_i$. If $\epsilon < p$ then, $\mathcal{DS}(\tau x_i|_\epsilon) \leq \mathcal{DS}(\sigma x_i|_\epsilon) = \{\}$, thus $\tau \trianglelefteq \sigma$.

So σ is the most general loose unifier for Γ. □

A similar substitution can be derived for strict unification, which is omitted here.

The Loose Dynamic Unification algorithm is similar to the standard order-sorted unification, modified to allow for the dynamic sort-information to be exploited. This algorithm is given in Figure 6.5. The symbol □ is used to represent failure.

(1) Loose Decomposition.

$$\frac{\{f(d_1,\ldots,d_n)\bullet S \stackrel{?}{=} f(d'_1,\ldots,d'_n)\bullet S'\} \cup U}{\{d_1 \stackrel{?}{=} d'_1,\ldots,d_n \stackrel{?}{=} d'_n\} \cup U} \quad \text{if } f \in \mathcal{F}_\Sigma$$

(2) Conflict.

$$\frac{\{f(d_1,\ldots,d_n)\bullet S \stackrel{?}{=} g(d'_1,\ldots,d'_m)\bullet S'\} \cup U}{\square} \quad \text{if } f,g \in \mathcal{F}_\Sigma \text{ and } f \neq g$$

(3) Trivial Equation.

$$\frac{\{d \stackrel{?}{=} d\} \cup U}{U}$$

(4) Occurs Check.

$$\frac{\{x \stackrel{?}{=} d\} \cup U}{\square} \quad \text{if } x \in \mathcal{X},\ d \notin \mathcal{X} \text{ and } x \in Vars(d).$$

(5) Eliminate.

$$\frac{\{x : s \stackrel{?}{=} d\} \cup U}{\{x \stackrel{?}{=} d\} \cup \{x \rightarrow d\}(U)} \quad \text{if } x \notin Vars(d),\ x \in Vars(U) \text{ and } \mathcal{DS}(d) \leq_\Sigma s$$

(6) Intersect.

$$\frac{\{x : s \stackrel{?}{=} y : s'\} \cup U}{\{x \stackrel{?}{=} z, y \stackrel{?}{=} z\} \cup U} \quad \text{if } x,y \in Vars(U) \text{ and } s' \bowtie s \text{ and } z : M_\Sigma(s,s') \text{ is a new variable.}$$

(7) Abstract.

$$\frac{\{x : s \stackrel{?}{=} f(d_1,\ldots,d_n)\bullet S\} \cup U}{\{x : s \stackrel{?}{=} f(d_1 \downarrow \{\},\ldots,d_n \downarrow \{\})\bullet\{s\}\} \cup U} \quad \text{if } x \in \mathcal{X}\ x \notin Vars(f(d_1,\ldots,d_n)\bullet S) \text{ and } S \neq \{s\}.$$

Figure 6.5: Rules for Loose Dynamic Unification.

The Decomposition, Conflict, Trivial Equation and Occurs Check rules are similar to ordinary statically sorted unification. However, the variable elimination rule, applied when a new solved form is found, is modified to cater for the possibility of a right-hand side having a dynamic sort less than the sort of the variable.

The remaining rules carry out the conversion of equations of the form $x \stackrel{?}{=} d$, where $x \in \mathcal{X}$, into solved forms. This ensures that the right-hand term has a dynamic sort less than or equal to x. This process is called *weakening* and analyses the sorts of the right-hand terms and weakens them by equating them to a variable of a lower sort if necessary. The Intersect rule handles the case where two variables are equated but are of incomparable sorts. They are both equated in alternative unifiers to a variable of a maximal common subsort. The Abstract rule converts a variable and non-variable pair which are not in solved form into a solved form by changing the dynamic sort of the non-variable term, to the variable's sort, and setting all other dynamic sorts to the empty set[2].

6.2.2 Correctness of the Unification Algorithm

If the above rules are applied to a unification problem, $\{d_1 \stackrel{?}{=} d_2\}$ say, using a suitable fair control which nondeterministically applies rules to the set of equations until no rule is applicable to any equation, this will terminate resulting in a set of equations in solved form, from which the most general unifier can be derived in a straightforward manner. This is formalised in the following theorem.

Definition 6.48. Given a control strategy, and a sequence of sets of equations $\{\Gamma_i\}_{i \in \{0 \ldots N\}}$ for N application steps of the strategy, set Γ_j is *Descended* from set Γ_i, written $\Gamma_i \stackrel{*}{\rightsquigarrow} \Gamma_j$, if there is a sequence of rule applications under the control strategy that converts Γ_i to Γ_j.

A control strategy is *Fair* if for all sequences $\Gamma_0, \Gamma_1, \ldots$ generated by the rules, if a rule M can be applied to some candidate Γ_m then for some $n \geq m$, such that $\Gamma_m \stackrel{*}{\rightsquigarrow} \Gamma_n$, Γ_n is transformed by the strategy by the rule M, $\Gamma_n \stackrel{M}{\rightsquigarrow} \Gamma_{n+1}$.

Theorem 6.49. Given a suitable fair nondeterministic control, the set of rules in Figure 6.5 forms a complete algorithm for loose dynamic unification. That is, given a unification problem $\Gamma_0 = \{d_1 \stackrel{?}{=} d_2\}$ it generates the most general loose unifier σ of d_1 and d_2.

[2]Strictly, this Abstract rule is unnecessary, since the most general unifier of the solved forms will perform this transformation. Nevertheless, for clarity we retain the rule here.

Proof (6.49). The proof is in two parts. First we demonstrate that the rules preserve unifiers: if $\Gamma_i \rightsquigarrow \Gamma_{i+1}$ then $\mu CSU^{\mathcal{W}}(\Gamma_i) = \mu CSU^{\mathcal{W}}(\Gamma_{i+1})$. The second part demonstrates that this transformation system terminates with a set of equations Γ_N, for some finite integer N, in solved form and, by Theorem 6.47, we can generate the most general loose unifier for Γ_N.

To demonstrate that the rules preserve unifiers, consider the rules in turn.

1. **Decomposition.** If $f(d_1,\ldots,d_n)\bullet S \stackrel{?}{=} f(d'_1,\ldots,d'_n)\bullet S' \in \Gamma_i$ for $f \in \mathcal{F}_{\Sigma d}$ and if for some substitution $\sigma, \sigma d_1 \simeq \sigma d'_1,\ldots,\sigma d_n \simeq \sigma d'_n$ then $\sigma(f(d_1,\ldots,d_n)) \simeq \sigma(f(d'_1,\ldots,d'_n))$. Thus $\mu CSU^{\mathcal{W}}(\Gamma_i) = \mu CSU^{\mathcal{W}}(\Gamma_{i+1})$, and the decomposition rule holds.
2. **Conflict.** If $f(d_1,\ldots,d_n)\bullet S \stackrel{?}{=} g(d'_1,\ldots,d'_m)\bullet S' \in \Gamma$ for $f \neq g \in \mathcal{F}_{\Sigma d}$, then no unifier exists which will make the terms equal.
3. **Trivial Equation.** If $d \stackrel{?}{=} d \in \Gamma$, any substitution is a unifier of this equation. Hence the unifiers of Γ are the unifiers of $\Gamma - \{d \stackrel{?}{=} d\}$.
4. **Occurs Check.** If $x \stackrel{?}{=} d \in \Gamma$, $x \in \mathcal{X}$ and $x \in Vars(t)$ then there is no substitution σ such that σx and σd are identified as σx will always be a subterm of σt, and no unifier exists.
5. **Eliminate.** If $x \stackrel{?}{=} d \in \Gamma_i$, $x \in \mathcal{X}$ and $x \notin Vars(t)$ then using Eliminate results in Γ_{i+1}, where $x = d \in \Gamma_{i+1}$ and hence a unifier, σ, of Γ_i and Γ_{i+1} must both identify x and d. Consider another pair $d_1 \stackrel{?}{=} d_2 \in \Gamma_i$. This is transformed into $d'_1 \stackrel{?}{=} d'_2 \in \Gamma_{i+1}$. Now every instance of x in $d_1 \stackrel{?}{=} d_2$ has been replaced by an instance of d. Thus $\sigma d_1 = \sigma d'_1$ and $\sigma d_2 = \sigma d'_2$ and so any unifier of Γ_i is a unifier of Γ_{i+1}.
6. **Intersect.** If $x : s \stackrel{?}{=} y : s' \in \Gamma$ where $s \bowtie s'$, then $\{x \mapsto y\}$ is not a well-formed Σ-substitution, but clearly $\{x \mapsto z : s'', y \mapsto z : s''\}$ where $s'' = M_\Sigma(s, s')$ are unifiers. This set is complete as for any other unifier $\{x \mapsto d\bullet S_1, y \mapsto d\bullet S_1\}$ will have some $s_1 \in S_1$ such that $s_1 \leq_\Sigma s''$, and the given unifier will subsume it.
7. **Abstract.** If $x : s \stackrel{?}{=} f(d_1,\ldots,d_n)\bullet S\} \in \Gamma$ and $\nexists s'' \in S$ such that $s'' \leq s$, then $\{x \mapsto f(d_1,\ldots,d_n)\bullet S\}$ is not a substitution. However, $\sigma = \{x \mapsto f(d_1 \downarrow \{\},\ldots,d_n \downarrow \{\})\bullet\{s\}\}$ forms a loose unifier.

We now prove termination. Let the measure M on equations be defined as a triple:

$$M(d_1 \bullet s_1 \stackrel{?}{=} d_2 \bullet s_2) = (\mathit{MinSort}(s_1, s_2), |\mathit{Vars}(d_1) \cup \mathit{Vars}(d_2)|, \{ht(d_1), ht(d_2)\})$$

where

$$\begin{aligned} \mathit{MinSort}(s_1, s_2) &= \{\} && \text{if } s_1 = s_2 \\ &= \{s_1\} && \text{if } s_1 \leq_\Sigma s_2 \\ &= \{s_2\} && \text{if } s_2 \leq_\Sigma s_1 \\ &= \{s_1, s_2\} && \text{if } s_2 \bowtie s_1 \end{aligned}$$

Define the ordering on M to be the lexicographic combination of the multiset ordering over $\leq_\Sigma$, the normal natural number ordering and the multiset ordering over the naturals. Clearly this is well-founded. Then let $M(\Gamma)$ be the multiset of these triples on a set of equations Γ, with the well-founded multiset extension of the above ordering.

Clearly Conflict and Occurs Check terminate; Loose Decomposition reduces the multiset of depths of terms; Trivial Equation reduces the number of triples in the multiset; Eliminate reduces the number of variables occurring equations, while Intersect and Abstract reduce the value of *MinSort*, and thus the rules terminate.

If $x = d \in \Gamma$ is in solved form and if for all other $y = d' \in \Gamma$, if $y = x$ then $d = d'$ and if $y \neq x$, then $x \notin d'$ then none of the above rules apply to $x = d$. Thus the rules terminate with Γ in solved form. Then from Theorem 6.47, the most general loose unifier can be given. □

Example 6.50. Using the signature given in Example 2.61, the dynamic unification algorithm applied to $\{x : \mathit{NeList} \stackrel{?}{=} (y : \mathit{List} \mathbin{@} z : \mathit{List}) \bullet \mathit{List}\}$ results in the most general unifier $\{x : \mathit{NeList} \mapsto (y : \mathit{List} \mathbin{@} z : \mathit{List}) \bullet \mathit{NeList}\}$, which is ill-sorted, but subsumes the well-sorted unifiers:

$$\{x \mapsto ((y_1 : \mathit{NeList}) @ z) \bullet \mathit{NeList},\ y \mapsto y_1\}$$

$$\{x \mapsto (y @ (z_1 : \mathit{NeList})) \bullet \mathit{NeList},\ z \mapsto z_1\}$$

□

Example 6.51. If we now reconsider Werner's Example 6.33, we can see that the

unification algorithm generates the most general dynamic unifier:

$$\sigma = \{y \mapsto mod_2(z : Nat) \bullet Pos\}$$

This term is not well-sorted, but note that *every* well-sorted solution can be factored through this ill-sorted solution. That is to say, given a well sorted solution:

$$\theta = \{y : Pos \mapsto mod_2(s^{2n+1}(0 \bullet Nat)\bullet^{2n+1}) \bullet Pos, z : Pos \mapsto s^{2n+1}(0 \bullet Nat)\bullet^{2n+1}\}$$

for some n, we can decompose this solution so that $\theta = \xi \circ \sigma$ where $\xi = \{z : Pos \mapsto s^{2n+1}(0 \bullet Nat)\bullet^{2n+1}\}$ is a well-sorted substitution. □

6.2.3 Generating Well-sorted Unifiers

The unification algorithm generates the unique most general loose unifier; however, this unifier is not in general well-sorted. Indeed, as we have discussed already, there is in general no decision procedure for deciding the existence of well-sorted unifiers. However, in practice it will be necessary to give sufficient conditions to decide the well-sortedness of unifiers.

Definition 6.52. The set of *most general well-sorted dynamic unifiers* is similar to Definition 6.39 A *Complete* set of well-sorted dynamic unifiers for a pair of dynamic terms d_1, d_2 on $Vars(d_1) \cup Vars(d_2) \subseteq \mathcal{W} \subseteq \mathcal{X}$, written $CWSU_Y^{\mathcal{W}}(d_1 \stackrel{?}{=} d_2)$, is a set of dynamic well-sorted substitutions such that:

1. $\forall \sigma \in CWSU_Y^{\mathcal{W}} \cdot Im(\sigma) \cap \mathcal{W} = \emptyset$
2. $CWSU_Y^{\mathcal{W}}(d_1 \stackrel{?}{=} d_2) \subseteq U(d_1 \stackrel{?}{=} d_2)$
3. $\forall \tau \in U(d_1 \stackrel{?}{=} d_2)$ such that $\tau \in DSubst_{\mathcal{S}}$ $\exists \sigma \in CWSU_Y^{\mathcal{W}}(d_1 \stackrel{?}{=} d_2) \cdot \sigma \preceq^{\mathcal{W}} \tau$.

A complete set of unifiers is *minimal*, denoted $\mu CWSU_Y^{\mathcal{W}}(d_1 \stackrel{?}{=} d_2)$, if $\forall \sigma, \rho \in \mu CWSU_Y^{\mathcal{W}}(d_1 \stackrel{?}{=} d_2) \cdot \sigma \preceq^{\mathcal{W}} \rho \Rightarrow \sigma \equiv^{\mathcal{W}} \rho$.

The well-sortedness or otherwise of unifiers can be isolated to the application of two rules in the above loose unification algorithm: the Intersect and the Abstract rules, which determine

the sorts of solved forms. These rules may introduce ill-sorted substitutions. In this section, we give some guidelines to the areas where sufficient criteria may be produced to determine well-sorted unifiers in special cases.

If sorts A and B are unrelated then unifying $x : A \stackrel{?}{=} y : B$ results in the unifier $\{x : A \mapsto z : \{A, B\}, y : B \mapsto z : \{A, B\}\}$, which may be ill-sorted if this intersection is empty. The Intersect rule produces well-sorted substitutions when the intersection sorts of the introduced new variable satisfies the *Semantic Inhabitedness* property.

Definition 6.53. Given a specification $\mathcal{S}$, a set of sorts S is *semantically inhabited* if there is a term $t \in W_{\mathcal{S}}(\mathcal{X})$ such that $S \subseteq \mathcal{SS}(t)$. Clearly, a set of sorts S is inhabited if there is a well-sorted dynamic term $d \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$ such that $S \subseteq \mathcal{DS}(d)$. The set of semantically inhabited sorts of $\mathcal{S}$ is denoted $Inhab_{\mathcal{S}}$.

This is more general than the syntactic inhabitedness of sorts discussed earlier (Definition 2.31), and clearly it is undecidable. However, it reduces to syntactic inhabitedness if the following criteria is met.

Lemma 6.54. Given a specification $\mathcal{S} = (\Sigma, E)$, if $\forall d \in \mathcal{D}_{\mathcal{S}}(\mathcal{X}) \cdot d \trianglerighteq L(d)$, then $Inhab_{\mathcal{S}} = Inhab_{\Sigma}$

Proof (6.54). For all $d \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$, $\mathcal{SS}(d) = \mathcal{DS}(L(d))$. Hence if S is semantically inhabited, then there is $t \in T_{\Sigma}(\mathcal{X})$ such that the $LS(t) \leq S$, and thus the sort is syntactically inhabited. □

The Abstract rule is even more intractable. However, we do know that all well-sorted unifiers can be 'factored through' a loose unifier; we can at least determine that a well-sorted unifier does not exist when there is no loose one.

Nevertheless, in special cases it may be possible to provide criteria to decide the well-sorted unifiers. If no equational theory is provided with the specification, the unification problem reverts to that of the standard order-sorted theory, with the use of the standard abstract rule which analyses the signature to determine unifiers.

Standard Abstract.

$$\frac{\{\{x : s \stackrel{?}{=} f(d_1, \ldots, d_n) \bullet S\} \cup U\} \cup C}{\{\{z_1 : s_1 \stackrel{?}{=} d_1, \ldots, z_n : s_n \stackrel{?}{=} d_n, x \stackrel{?}{=} f(z_1, \ldots, z_n) \bullet \{s\}\} \cup U \quad |s_1, \ldots, s_n \rightarrow s' \in \Sigma_f^s\} \cup C}$$

if $S \not\leq s$ and
$\Sigma_f^s = \{w \rightarrow s' | f : w \rightarrow s' \in \Sigma \wedge s' \leq s \wedge w \; maximal\}$
$z_1, \ldots, z_n$ new variables.

Figure 6.6: Abstract Rule for Standard Well-sorted Loose Dynamic Unification.

It can be shown that these rule can be applied to any specification for which least syntactic dynamic terms are also least semantic terms.

Lemma 6.55. Given a specification $\mathcal{S} = (\Sigma, E)$, if $\forall d \in \mathcal{D}_{\mathcal{S}}(\mathcal{X}) \cdot d \trianglerighteq L(d)$, then the most general well-sorted unifiers are generated by the algorithm in Figure 6.5, with the abstract rule replaced by the rule in Figure 6.6. Note that this generates sets of unifiers.

Proof (6.55). If $x : s \stackrel{?}{=} f(d_1, \ldots, d_n) \bullet S$ and $S \not\leq s$, a most-general well-sorted unifier σ is such that $\sigma x \simeq f(\sigma d_1, \ldots \sigma d_n) \bullet S$ and $\mathcal{DS}(\sigma x) = s \geq \mathcal{SS}(\sigma x)$, as σ is well-sorted. However, as $\forall d \in \mathcal{D}_{\mathcal{S}}(\mathcal{X}) \cdot d \trianglerighteq L(d)$, then $\mathcal{SS}(\sigma x) = \mathcal{LS}(L(\sigma x))$, so $\exists s_1, \ldots, s_n \rightarrow s' \in \Sigma_f^s$ such that $SS(\sigma x) = s'$ and so $\mathcal{DS}(\sigma d_1) = s_1, \ldots, \mathcal{DS}(\sigma d_n) = s_n$. So σ is also a well-sorted most-general unifier of $\{z_1 : s_1 \stackrel{?}{=} d_1, \ldots, z_n : s_n \stackrel{?}{=} d_n, x \stackrel{?}{=} f(z_1, \ldots, z_n) \bullet \{s\}\}$, where $z_1, \ldots, z_n$ are new variables. □

This rule can generate several alternative unifiers, each with its own candidate sets.

It still remains to be seen what is to be done in cases which fall outside these criteria. In [HKK93] the unification process is aborted when a term is encountered which does not already have a dynamic sort of at least that of the variable. That is the Abstract rule is replaced by that in Figure 6.7.

If the terms being unified are well-sorted, this rule will guarantee that the unifiers and unified terms are well-sorted also, but it may result in well-sorted unifiers being missed. Sufficient criteria to guarantee the completeness of this method are verified during the completion process.

Abstract for [HKK93]

$$\frac{\{x : s \stackrel{?}{=} f(d_1, \ldots, d_n) \bullet S\} \cup U}{\square} \quad \text{if } x \in \mathcal{X},\ x \notin \mathit{Vars}(f(d_1, \ldots, d_n) \bullet S) \text{ and } S \not\leq \{s\}.$$

Figure 6.7: Abstract for [HKK93]

Alternatively, it may be useful to generate the most general loose unifier as here, and the upward strong maximal unified term, and later analyse them to determine well-sorted instances.

Chapter 7

Dynamic Order-Sorted Rewriting

In this chapter we define the concept of rewriting using dynamic terms. For simplicity, we do not consider rewriting modulo equations.

7.1 Dynamic Rewriting

The concept of rewriting using dynamic rules can be sketched as follows. Dynamic terms record the known minimal sorts of terms. Rewriting changes this sort information dynamically; each time a rewrite occurs extra sort information can be gathered from this application of an equational inference step. Thus rewriting not only changes the terms, but also manipulates the sorts of terms.

Definition 7.1. Dynamic Rewrite Rule.
A *dynamic rewrite rule* is a directed dynamic equation, written $\forall Y.l \rightarrow_S r$ such that $\mathcal{DS}(l) \cup \mathcal{DS}(r) \subseteq S$. When the set of variables $Y = \mathit{Vars}(l) \cup \mathit{Vars}(r)$, then we may omit this set. If $l, r \in \mathcal{D}_S(\mathcal{X})$and $S \subseteq \mathcal{SS}(l)$ then the rewrite rule is well-sorted.

As dynamic rewrite rules are dynamic equations, the translation into $\overline{T}_{\overline{\Sigma}}(\mathcal{X})$-equations is

as Definition 5.47. The rewriting relation is dependent on the definition of matching. The most general form using well-sorted terms is to use semantic matching.

Definition 7.2. Semantic Rewriting.
Given a set of dynamic rewrite rules R, a dynamic term d rewrites to d' with rule $l \rightarrow_S r \in R$, if there is a well-sorted dynamic substitution σ, and path $p \in O(d)$, written $d \rightarrow^{(\Sigma,R)}_{l \rightarrow_S r, \sigma, p} d'$, such that $d|_p \simeq \sigma l$ and $d' = d[p \leftarrow \sigma r]$.

This definition is analogous to the $\rightarrow^{(\Sigma,R)}$ defined in [Wer93]. If a subterm is of the same semantic sort as an instance of the left-hand side of a rule, the rule can be applied. However, semantic matching is undecidable, and thus so is this relation. Consequently, we replace this definition with one using weaker matchings. The following defines a rewrite relation for each form of matching.

Definition 7.3. Dynamic Term Rewriting.
Given a set of dynamic rewrite rules R, a dynamic term d *X-term-rewrites* to d' by rule $\forall Y. l \rightarrow_S r \in R$, where $\phi(l) \neq \phi(r)$, written $d \rightarrow^{X,T}_{l \rightarrow_S r, \sigma, p} d'$, if there is a dynamic substitution $\sigma : Y \rightarrow \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, and path $p \in O(d)$, such that there is an X-match $l \sqsubseteq^X_\sigma d|_p$ and $d' = d[p \leftarrow \sigma r] \downarrow_p S \cup \mathcal{DS}(d|_p)$.

Note that we exclude rewriting via a rule which has terms with the same resolvents. Such a rule is valid in the set of rules, and can be non-trivial since the dynamic sorts of the terms may be different, but its use for term rewriting would lead to non-termination. Subscripts are omitted when the context is clear.

Example 7.4. Given the specification $\mathcal{S}$ as follows:

Sorts:	$A\ B$		
Subsorts:	$A \leq B$		
Operators:	$a : \rightarrow A,$	$b : \rightarrow B,$	$g : B \rightarrow B$
	$f : B \rightarrow B$		$g : A \rightarrow A$
Equations:	$b = a$		
	$f(x : A) = x$		

The equations are converted to dynamic rules, by taking the canonical set of well-sorted dynamic equations, as in Definition 5.48. These equations are then oriented left to right, to give:

$$\begin{aligned} b\bullet B &\rightarrow_A a\bullet A \\ f(x:A)\bullet B &\rightarrow_A x:A \end{aligned}$$

Thus we have a rewriting proof under semantic rewriting that

$$f(g(b\bullet B)\bullet B)\bullet B \rightarrow^{(\Sigma,R)} f(g(a\bullet A)\bullet B)\bullet B \rightarrow^{(\Sigma,R)} g(a\bullet A)\bullet A$$

using the *semantic match* $\{x \mapsto g(a\bullet A)\bullet A\}$ with the second rule. However, there is no dynamic term-rewriting proof, since the substitution for the second rule to apply would be $\{x : A \mapsto g(a\bullet A)\bullet B\}$ which is not sort-preserving. However, the term $g(a\bullet A)\bullet A$ is well-sorted via sort-propagation. □

In standard rewriting theory the use of the least sort is implicit when checking the well-sortedness of a substitution. In the dynamic rewriting theory, as sorts are explicitly carried as part of the terms, the recalculation of sorts needs to be carried out explicitly. This motivates the following definition.

Definition 7.5. Static Sort Rewriting (Sort Propagation).
Given a signature Σ, a dynamic term d *statically sort-rewrites (sort propagates)* to d' written $d \rightarrow^{\Sigma}_{p,(\langle s_1 \ldots s_n\rangle,s)} d'$ if $\exists p \in O(d)$ such that $d|_p = f(d_1,\ldots,d_n)\bullet S$, $(\langle s_1 \ldots s_n\rangle, s) \in ranks(f), \mathcal{DS}(d_1) \leq s_1,\ldots,\mathcal{DS}(d_n) \leq s_n$ and $S \not\leq s$. Then $d' = P_{p,(\langle s_1 \ldots s_n\rangle,s)}(d)$.

In general we shall refer to this relation as sort propagation, to prevent confusion with dynamic sort-rewriting below.

Example 7.6. Example 7.4 revisited. Now we can perform the rewriting steps:

$$f(g(b\bullet B)\bullet B)\bullet B \rightarrow^{T} f(g(a\bullet A)\bullet B)\bullet B \rightarrow^{\Sigma} f(g(a\bullet A)\bullet A)\bullet B \rightarrow^{T} g(a\bullet A)\bullet A$$

□

However, rewriting is still not complete.

Example 7.7. Given the specification $\mathcal{S}$ as follows:

Sorts:	$A\ B$
Subsorts:	$A \leq B$
Operators:	$a : \rightarrow A, b : \rightarrow B$
	$f : B \rightarrow B$
Equations:	$a = b$
	$f(x : A) = x$

In standard order-sorted rewriting theory it is not possible to orient the equation, $a \rightarrow b$ and then give a rewriting proof of $f(b) = b$. However, converting it to a dynamic rewriting system gives the following dynamic rules.

$$a \bullet A \rightarrow_A b \bullet B$$
$$f(x : A) \bullet B \rightarrow_A x : A$$

and there is a direct rewriting proof using semantic rewriting.

$$f(b \bullet B) \bullet B \rightarrow^{(\Sigma,R)} b \bullet A$$

with the semantic match $\{x \mapsto b \bullet A\}$ with the second rule. However, there is no dynamic term-rewriting proof, since there is no (weak/strong) match of the left-hand side of either rule on any subterm of $f(b \bullet B) \bullet B$. Indeed, there is no proof of $f(b \bullet B) \bullet B \overset{*}{\longleftrightarrow}^{T} b \bullet A$, as there is no $d \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$ such that $d \rightarrow^{T} f(b \bullet B) \bullet B$. However, we do have the dynamic rewrite proof: $f(b \bullet A) \bullet B \rightarrow^{T} b \bullet A$. □

This example demonstrates that rewriting is incomplete, even combined with sort-propagation. The term $b \bullet B$ is a normal form with respect to the $\rightarrow^{T}$ relation, but this term is not in S-normal form; there is a well-sorted term, $b \bullet A \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$, such that $b \bullet A \trianglelefteq b \bullet B$. However, there is no proof of $b \bullet B \rightarrow^{T} b \bullet A$. This rewrite step can be achieved by rewriting using a *right-hand side* of a rule. This motivates the following definition.

Definition 7.8. Dynamic Sort Rewriting.

Given a set of dynamic rewrite rules R, a dynamic term d *X-sort-rewrites* to d' written $d \rightarrow^{X,S}_{l \rightarrow_S r,\sigma,p} d'$ where $d' = d \downarrow_p S$, if there is a rule $\forall Y.l \rightarrow_S r \in R$, a dynamic substitution

$\sigma : Y \rightarrow \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, $p \in O(d)$ such that there is an X-match $r \sqsubseteq^X_\sigma d|_p$ and $\mathcal{DS}(d) \not\leq S$.

Thus extra sort information is derived by matching using right-hand sides. Note that we can sort-rewrite by a rule $l \rightarrow_S r$ where $l \simeq r$. Sort rewriting does not change the operators of a term, only its dynamic sort, replacing the dynamic sort by a lower one. In general this relation is called *Sort Rewriting*, without confusion with sort propagation. *Sort normalisation* is the reflexive-transitive closure of the sort-rewriting relation, $\xrightarrow{*}{}^S$. For any finite signature, and finite set of dynamic rewrite rules, sort-normalisation terminates, since when a rule sort rewrites a subterm, it cannot rewrite at that path again.

Lemma 7.9. If $d \bullet A \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ is sort-rewritten at root, $d \bullet A \rightarrow^S_{\rightarrow_{C r,\sigma,\epsilon}} d \bullet B$ then $B < C$.

Proof (7.9). Direct from the definition of sort rewriting. □

Rewriting thus has three components, combined into one dynamic rewriting relation.

Definition 7.10. Dynamic Rewriting
Given a set of dynamic rewrite rules, the *dynamic rewriting* relation with respect to a matching algorithm X, $\rightarrow^X$, is given by $\rightarrow^{X,T} \cup \rightarrow^{X,S} \cup \rightarrow^\Sigma$. When the (Term, Sort, or Propagation) rewriting takes place at position p we write $\rightarrow^X_p$.

Depending on the variety of matching used, this relation is weak, or upward strong, downward strong or strict, as denoted in the following table.

$\rightarrow^W$	Weak Matching
$\rightarrow^U$	Upward Strong Matching
$\rightarrow^{V,}$	Downward Strong Matching
$\rightarrow^{St}$	Strict Matching

Due to the inclusions of the matching algorithms described above, we have the following relationship between the rewriting relations on well-sorted terms.

$$\longrightarrow^{St} \subset \begin{array}{l} \longrightarrow^{U} \subset \\ \longrightarrow^{V} \subset \end{array} \longrightarrow^{W} \subset \longrightarrow^{\Sigma,R}$$

If the matching used is unimportant or obvious in context, we write $\rightarrow^{D}$.

The dynamic rewriting relation is strictly more general than dynamic term rewriting as we can see by reconsidering the previous example.

Example 7.11. Example 7.7 revisited. We now have the dynamic rewrite proof:

$$f(b\bullet B)\bullet B \rightarrow^{S} f(b\bullet A)\bullet B \rightarrow^{T} b\bullet A \,{}^{S}\!\leftarrow\, b\bullet B$$

$f(b\bullet B)\bullet B$ does not match with any left-hand side, but will sort normalise by the first rule. This reduces the term to one which will term rewrite, producing the dynamic term $b\bullet A$ recording that b has semantic sort A. Thus we now have a rewrite proof. □

Thus we do not need the compatibility condition (see Definition 2.67) on the rewrite relation to ensure its completeness; this is proven in the next section.

For a set of dynamic rewrite rules R we write its dynamic rewriting relation as $\rightarrow^{D}_{R}$, or when it is clear from context that it is the full rewriting relation that is used, then the superscript may be dropped. The following relations are also defined.

- Inverse relation: $d \,{}^{D}\!\leftarrow d'$ if $d' \rightarrow^{D} d$.
- nth Iteration: $d \xrightarrow{n}{}^{D} d'$ if $\exists d_0, \ldots, d_n$ such that $d = d_0 \rightarrow^{D} d_1 \cdots d_{n-1} \rightarrow^{D} d_n = d'$.
- nth Inverse Iteration: $d \,{}^{D}\!\overset{n}{\leftarrow} d'$ if $\exists d_0, \ldots, d_n$ s.t. $d = d_0 \,{}^{D}\!\leftarrow d_1 \cdots d_{n-1} \,{}^{D}\!\leftarrow d_n = d'$.
- Transitive Closure: $d \xrightarrow{+}{}^{D} d'$ if $\exists n > 0$ such that $d \xrightarrow{n}{}^{D} d'$.
- Transitive-Reflexive Closure: $d \xrightarrow{*}{}^{D} d'$ if $d \xrightarrow{+}{}^{D} d'$ or $d = d'$.

Definition 7.12. Given a set of Dynamic Rewrite Rules R, then $d \in \overline{\mathcal{D}}_{\Sigma}(\mathcal{X})$ is in *normal form* if there is no rule $l \rightarrow_{s} r$, path p and substitution σ, such that $d \rightarrow^{D}_{l \rightarrow_{s} r, p, \sigma} d'$. A *normal*

form of a term d, $d\downarrow_R$, is a term such that $d \xrightarrow{*}{}^{D} d\downarrow_R$ and is in normal form. R is *normalising* if every term has a normal form.

We also define dynamic rewriting with respect to a specified set of dynamic terms.

Definition 7.13. Given a set of dynamic rewrite rules R, and a set of terms $E \subseteq \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, *$R$ rewrites in E* is the relation $\{(d,d') \mid d,d' \in E \text{ and } d \rightarrow^D d'\}$.

The results which follow are with respect to the set $\overline{\mathcal{D}}_\Sigma(\mathcal{X})$ unless otherwise specified.

The dynamic rewriting relation is a well-defined reduction relation on dynamic terms which is stable with respect to substitutions and monotonic with respect to subterms.

Lemma 7.14. Stability of Dynamic Rewriting.
If $d \rightarrow^D d'$ then for any dynamic substitution λ, $\lambda d \rightarrow^D \lambda d'$.

Proof (7.14). Consider a term-rewriting step with respect to X-matching. If $d \rightarrow^T_{l \rightarrow_{S^r,\sigma,p}} d'$ then $l \sqsubseteq^X_\sigma d|_p$ and $d' = d[p \leftarrow \sigma r]\downarrow_p S \cup \mathcal{DS}(d|_p)$. Then as $\sigma l \simeq d|_p$, $(\lambda \circ \sigma)l \simeq \lambda(d|_p)$, so $l \sqsubseteq^X_{\lambda\circ\sigma} d|_p$ and $\lambda d \rightarrow^T_{l \rightarrow_{S^r,\lambda\circ\sigma,p}} d_1$ where $d_1 = \lambda d[p \leftarrow (\lambda \circ \sigma)r]\downarrow_p S \cup \mathcal{DS}(\lambda d|_p)$. By lemma 5.24 $\mathcal{DS}(\lambda d|_p) = \mathcal{DS}(d|_p)$ if $p \in O(d)$ and $d|_p \notin \mathcal{X}$. For any e, $\lambda d[p \leftarrow \lambda e] = \lambda d[p \leftarrow e]$, so $d_1 = \lambda d[p \leftarrow \sigma r]\downarrow_p S \cup \mathcal{DS}(\lambda d|_p) = \lambda d'$. The proof for a dynamic sort-rewriting step is similar, and sort-propagation is trivial. □

Lemma 7.15. Monotonicity of Dynamic Rewriting.
If $d_1 \rightarrow^D_{l \rightarrow_{S^r,\sigma,p}} d_2$ then $\forall d \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, and $\forall q \in O(d)$, $d[q \leftarrow d_1] \rightarrow^D_{l \rightarrow_{S^r,\sigma,q.p}} d[q \leftarrow d_2]$.

Proof (7.15). Consider a term-rewriting step with respect to X-matching. If $d_1 \rightarrow^T_{l \rightarrow_{S^r,\sigma,p}} d_2$ then $l \sqsubseteq^X_\sigma d_1|_p$ and $d_2 = d_1[p \leftarrow \sigma r]\downarrow_p S \cup \mathcal{DS}(d_1|_p)$. Then as $\sigma l \simeq d_1|_p$, $\sigma l \simeq d[q \leftarrow d_1]|_{q.p}$ and thus $d \rightarrow^T_{l \rightarrow_{S^r,\sigma,q.p}} d'$ where $d' = d[q \leftarrow d_1][q.p \leftarrow \sigma r]\downarrow_{q.p} S \cup \mathcal{DS}(d[q \leftarrow d_1]|_{q.p})$. Since $q \leq q.p$:

$$d' = d[q \leftarrow d_1[p \leftarrow \sigma r]\downarrow_p S \cup \mathcal{DS}(d_1|_p)] = d[q \leftarrow d_2]$$

A similar proof follows for dynamic sort-rewriting, and sort-propagation is trivial. □

Dynamic rewriting either reduces or leaves unchanged the top sort of a term.

Lemma 7.16. If $\rightarrow_R$ is a dynamic rewriting relation then if $d_1 \rightarrow_R d_2$, $\mathcal{DS}(d_1) \geq_\Sigma \mathcal{DS}(d_2)$.

Proof (7.16). Trivial from the definition of term and sort rewriting. □

Both term and sort rewriting preserve the well-sortedness of terms.

Lemma 7.17. If $d_1, d_2 \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, and $\mathcal{S} = (\Sigma, \Phi(R))$ and R is a set of well-sorted dynamic rewrite rules then if $d_1 \in \mathcal{D}_\mathcal{S}(\mathcal{X})$ and if $d_1 \xrightarrow{*}_R d_2$ then $d_2 \in \mathcal{D}_\mathcal{S}(\mathcal{X})$.

Proof (7.17). For this proof, we need only show that the lemmas hold for the one step relations $\rightarrow^T_R$, $\rightarrow^S_R$ and $\rightarrow^\Sigma_R$, with respect to X-matching.

Assume that $d_1 \rightarrow^T_{\sigma,l \rightarrow_{S^r,\epsilon}} d_2$, that is term rewrite at root. Then $d_2 = \sigma r \downarrow_\epsilon S \cup \mathcal{DS}(d_1)$ and $l \sqsubseteq^X_\sigma d_1$. If $d_1 \in \mathcal{D}_\mathcal{S}(\mathcal{X})$, and by definition $l, r \in \mathcal{D}_\mathcal{S}(\mathcal{X})$, so σ must be well sorted via Lemma 6.5 and so by Lemma 5.32, $\sigma r \in \mathcal{D}_\mathcal{S}(\mathcal{X})$, and since S is a valid set of sorts for r, and $\mathcal{DS}(d_1)$ is a valid set of sorts for d_1, then $\sigma r \downarrow_\epsilon S \cup \mathcal{DS}(d_1) \in \mathcal{D}_\mathcal{S}(\mathcal{X})$.

If $d_1 \rightarrow^T_{\sigma,l \rightarrow_{S^r,p}} d_2$, where p is not root, then $d_2 = d_1[p \leftarrow \sigma r] \downarrow_p S \cup \mathcal{DS}(d_1|_p)$. $d_1|_p \in \mathcal{D}_\mathcal{S}(\mathcal{X})$ and $l, r \in \mathcal{D}_\mathcal{S}(\mathcal{X})$, and thus $\sigma r \in \mathcal{D}_\mathcal{S}(\mathcal{X})$ such that $\phi(\sigma r) =_R \phi(d_1|_p)$. Hence, $d_1[p \leftarrow \sigma r] \in \mathcal{D}_\mathcal{S}(\mathcal{X})$. Also as S are valid sorts for r, and $\mathcal{DS}(d_1)$ is a valid set of sorts for $d_1|_p$, then $d_1[p \leftarrow \sigma r] \downarrow_p S \cup \mathcal{DS}(d_1|_p) \in \mathcal{D}_\mathcal{S}(\mathcal{X})$.

Similarly for sort rewriting. If $d_1 \rightarrow^S_{\sigma,l \rightarrow_{S^r,p}} d_2$, then $d_2 = d_1 \downarrow_p S$. As S is a valid sort for σr then $d_1 \downarrow_p S \in \mathcal{D}_\mathcal{S}(\mathcal{X})$.

From previous lemmas sort propagation preserves well-sortedness. □

Example 7.18. If we have the specification:

Subsorts:	$A \leq B$
Operators:	$a : \rightarrow A \quad f : A\ A \rightarrow A$
	$b : \rightarrow B \quad f : B\ B \rightarrow B$
Equations:	$b = a$

Then $f(x : A, b \bullet B) \bullet A \in \mathcal{D}_\mathcal{S}(\mathcal{X})$, and $f(x : A, b \bullet B) \bullet A \rightarrow^T f(x : A, a \bullet A) \bullet A$ using the rewrite

rule $b\bullet B \rightarrow_A a\bullet A$, and $f(x : A, a\bullet A)\bullet A \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$. Note that a strict match is used here, so this follows for all varieties of matching. □

However, the converse does not hold for all types of matching: it is not the case that if $d_2 \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$ and if $d_1 \xrightarrow[R]{*\ U} d_2$ then $d_1 \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$.

Example 7.19. If we have the specification:

Sorts:	$A\ B$	
Subsorts:	$A \leq B$	
Operators:	$b : \rightarrow B$	$g : B \rightarrow B$
Equations:	$g(g(x : B)) = x : B$	

Then $g(g(b\bullet B)\bullet A)\bullet B \notin \mathcal{D}_{\mathcal{S}}(\mathcal{X})$. However:

$$g(g(b\bullet B)\bullet A)\bullet B \rightarrow b\bullet B$$

using the rule $g(g(x : B)\bullet B)\bullet B \rightarrow_B x : B$ with an upward strong match. □

Thus problems can be caused if there are invalid sorts occurring in the redex of a reduction. However, we can show the following property.

Lemma 7.20. Given a set of well-sorted dynamic rewrite rules R, if $d_1 \rightarrow_p^D d_2$ and $d_2 \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$, then there is a dynamic term $d_1' \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$ such that $d_1' \rightarrow_p^D d_2$ and $\phi(d_1) = \phi(d_1')$. Further, if strict or downward strong rewriting are used, then $d_1 \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$.

Proof (7.20). If $d_1 \rightarrow^T_{l \rightarrow_{S^r,\sigma,p}} d_2$ then $d_2 = d_1[p \leftarrow \sigma r] \downarrow_p S \cup \mathcal{DS}(d_1|_p)$. So $d_2 \rightarrow^T_{r \rightarrow_{Sl,\sigma,p}} d_1'$ where $d_1' = d_2[p \leftarrow \sigma l] \downarrow_p S \cup \mathcal{DS}(d_2|_p)$ and by previous lemma d_1' must be well-sorted. Clearly, $d_1' \rightarrow^T_{l \rightarrow_{S^r,\sigma,p}} d_2$, and $\phi(d_1) = \phi(d_1')$.

Clearly, in the case of the sort-rewriting and sort-propagation, as the sort of the term is reduced in either case, if $d_2 \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$ then $d_1 \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$.

For strict term-rewriting $d_1|_p = \sigma l$, or for downward strong rewriting $d_1|_p \trianglerighteq \sigma l$. In either case, $\sigma l \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$, then $d_1|_p \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$, and since $d_2 \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$, then so is d_1. □

However, a property of upward strong dynamic rewriting not shared by downward strong or strict is that it respects the approximation relation.

Lemma 7.21. Term Rewriting. If $d_1, d_2 \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, $d_1 \trianglerighteq d_2$, and $d_1 \rightarrow^T_{l \longrightarrow_{sr},\sigma,p} d_1'$ using weak or upward strong rewriting, then there is some substitution σ', and dynamic term d_2' such that $d_2 \rightarrow^T_{l \longrightarrow_{sr},\sigma',p} d_2'$ and $d_1' \trianglerighteq d_2'$.

Sort Rewriting. If $d_1, d_2 \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, $d_1 \trianglerighteq d_2$, and $d_1 \rightarrow^S_{l \longrightarrow_{sr},\sigma,p} d_1'$ using weak or upward strong rewriting, then if $\mathcal{DS}(d_2|_p) \not\leq S$, there is some substitution σ', and dynamic term d_2' such that $d_2 \rightarrow^S_{l \longrightarrow_{sr},\sigma',p} d_2'$ and $d_1' \trianglerighteq d_2'$, and if $\mathcal{DS}(d_2|_p) \leq S$, then $d_1' \trianglerighteq d_2$.

Proof (7.21). Consider the case of weak term-rewriting. $l \sqsubseteq^W_\sigma d_1|_p$ then $d_1|_p \simeq \sigma l$, and for any q such that $l|_q \in \mathcal{X}$, x_q say, then $\sigma x_q = d_1|_{p.q}$ and $s(x_q) \geq \mathcal{DS}(d_1|_{p.q})$. However, as $d_1 \trianglerighteq d_2$ then $s(x_q) \geq \mathcal{DS}(d_1|_{p.q}) \geq \mathcal{DS}(d_2|_{p.q})$ and therefore the substitution $\sigma' = \{x_q \mapsto d_2|_{p.q}\}_{l|_q \in \mathcal{X}}$ is a well-sorted dynamic substitution and thus $d_2|p \simeq \sigma' l$, $l \sqsubseteq^W d_2|_p$ and so $d_2 \rightarrow^D d_2'$. For upward strong matching, note that if $l \sqsubseteq^U d_1$ and $d_2 \trianglelefteq d_1$ then $l \sqsubseteq^U d_2$.

Now $d_1' = d_1[p \leftarrow \sigma r] \downarrow_p S \cup \mathcal{DS}(d_1|_p)$ and $d_2' = d_2[p \leftarrow \sigma r] \downarrow_p S \cup \mathcal{DS}(d_2|_p)$ Clearly, $d_1' \simeq d_2'$, since $d_1 \simeq d_2$. Consider paths q in d_1'. If $p \bowtie q$, then $d_1'|_q = d_1|_q \trianglerighteq d_2|_q = d_2'|_q$. If $p < q$, $q = p.p'$ say, and p' is at position at or below a variable in r, then $d_1'|_q$ is the result of the substitution and hence is a subterm of d_1. $d_2'|_q$ is the equivalent subterm in d_2, and hence $d_1'|_q \trianglerighteq d_2'|_q$. If p' is a non-variable position in r, then $d_1'|_q = \sigma r|_{p'}$ and $d_2'|_q = \sigma' r|_{p'}$, and thus $\mathcal{DS}(d_1'|_q) = \mathcal{DS}(r|_{p'}) = \mathcal{DS}(d_2'|_q)$. If $q = p$ then $\mathcal{DS}(d_1'|_q) = S \cup \mathcal{DS}(d_1|_p) \geq S \cup \mathcal{DS}(d_2|_p) = \mathcal{DS}(d_2'|_q)$. If $p > q$ then $d_1'|_q = d_1|_q \trianglerighteq d_2|_q = d_2'|_q$. Thus $d_1' \trianglerighteq d_2'$.

A similar proof exists for sort-rewriting, with the observation that if $\mathcal{DS}(d_2|_p) \leq S$, then since $\mathcal{DS}(d_2|_p) \leq \mathcal{DS}(d_1|_p)$ then $\mathcal{DS}(d_2|_p) \leq \mathcal{DS}(d_1|_p) \cup S = \mathcal{DS}(d_1'|_p)$, so $d_1' \trianglerighteq d_2$. □

This property has already been shown for sort propagation, in Lemma 5.43.

Lemma 7.22. If we have dynamic substitutions $\pi \trianglelefteq \theta$ and $\theta d \xrightarrow{*}^D d'$, then $\pi d \xrightarrow{*}^D d''$ and $d'' \trianglelefteq d'$.

Proof (7.22). If $\pi \trianglelefteq \theta$, then $\forall x \in Dom(\pi)$, $\pi x \trianglelefteq \theta x$. Hence $\pi d \trianglelefteq \theta d$. We need only show the lemma for one step rewriting. Given a term-rewriting step $\theta d \rightarrow^{T}_{l \rightarrow_{Sr,\sigma,p}} d'$, then, for each path q such that $l|_q = x \in \mathcal{X}$, $\mathcal{DS}(\pi d|_{p.q}) \leq \mathcal{DS}(\theta d)|_{p.q} \leq s(x)$. Form a matching substitution σ' such that $\sigma' x = \pi d|_{p.q}$, and rewrite $\pi d \rightarrow^{T}_{l \rightarrow_{Sr,\sigma',p}} d''$. Clearly $\sigma' \trianglelefteq \sigma$, and $d'' \trianglelefteq d'$ follows. Similar proofs exist for sort rewriting and sort propagation, except if $\mathcal{DS}(\pi d|_p) \leq S$ where S is the sort to be added, then no sort-rewriting step (respectively propagation step) exists, in which case let $d'' = \pi d$ and the result follows. □

Corollary 7.23. If $d_1, d_2 \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, $d_1 \trianglerighteq d_2$, and $d_1 \xrightarrow{*}^{D}_{l \rightarrow_{Sr,\sigma,p}} d_1'$ using weak or upward strong matching, then there is a d_2' such that $d_2 \xrightarrow{*}^{D}_{l \rightarrow_{Sr,\sigma',p}} d_2'$ and $d_1' \trianglerighteq d_2'$ and $\sigma \trianglerighteq \sigma'$.

Proof (7.23). Immediate from the definition of $\trianglelefteq$ and the previous Lemmas. □

We also have the following similar result.

Lemma 7.24. If $d \trianglerighteq B_\Sigma(d')$ and $d \rightarrow_{l \rightarrow_{Sr,p}} d_1$ for weak and strong rewriting, then $d' \rightarrow_{l \rightarrow_{Sr,p}} d_1'$ and $d_1 \trianglerighteq B_\Sigma(d_1')$.

Proof (7.24). We note that from the definition of canonical matches, if $l \sqsubseteq^{U}_{\sigma} B_\Sigma(d')$ then $l \sqsubseteq^{U}_{\sigma} d'$, and so the lemma follows from Corollary 7.23. □

To give a notion of the replacement of equals for equals, we use the reflexive symmetric transitive closure of $\rightarrow^{D}$ restricted to well-sorted terms.

Definition 7.25. Given a set of dynamic rewrite rules R, let $\longleftrightarrow^{D}_{R}$ be the symmetric closure of $\rightarrow^{D}_{R}$ restricted to well-sorted terms. That is

$$d_1 \overset{D}{\underset{R}{\longleftrightarrow}} d_2 \text{ if and only if } d_1, d_2 \in \mathcal{D}_{\mathcal{S}}(\mathcal{X}) \text{ and } d_1 \rightarrow^{D}_{R} d_2 \text{ or } d_2 \rightarrow^{D}_{R} d_1$$

We write $\overset{n}{\longleftrightarrow}$, for the n-fold transitive closure of $\longleftrightarrow$, $\overset{+}{\longleftrightarrow}$ for its transitive closure, and $\overset{*}{\longleftrightarrow}$ for its transitive reflexive closure.

One aspect of dynamic rewriting which differs from standard rewriting theories is that this relation is not the same as the similar relation $\rightarrow_R \cup \rightarrow_{R^{-1}}$, since the latter relation is only

symmetric in the case of strict rewriting.

Example 7.26. Considering the specification of Example 7.18, note that $a \bullet A \longleftrightarrow b \bullet B$ since $b \bullet B \rightarrow a \bullet A$, but it is not the case that $a \bullet A \rightarrow_{R \cup R^{-1}} b \bullet B$. □

However, it is sufficient to consider $\overset{*}{\longleftrightarrow}$ for the soundness and completeness of rewriting for any matching algorithm. The notation is extended to allow the relation to stand over sets of dynamic equations.

Definition 7.27. Given a set of dynamic equations E, let $R(E) = \{l \rightarrow_S r | l =_S r \in E\}$. Then we define: $\longleftrightarrow^D_E = \longleftrightarrow^D_{R(E)}$, and similarly for $\overset{n}{\longleftrightarrow}_E$, $\overset{+}{\longleftrightarrow}_E$ and $\overset{*}{\longleftrightarrow}_E$.

Not all dynamic terms can be reached using these relations from least-syntactic forms. We use these definitions to define the following subsets of the well-sorted dynamic terms.

Definition 7.28. Given a set of dynamic rewrite rules R, the set of *Reachable Dynamic Terms* $\mathcal{E}_R(\mathcal{X})$ is given by:

$$\mathcal{E}_R(\mathcal{X}) = \{d \mid \exists t \in \overline{T}_{\overline{\Sigma}}(\mathcal{X}) \cdot L(t) \overset{*}{\longleftrightarrow}_R d\}$$

Further, the set of *Rewrite Reachable Dynamic Terms* $\vec{\mathcal{E}}_R(\mathcal{X})$, is given by:

$$\vec{\mathcal{E}}_R(\mathcal{X}) = \{d \mid \exists t \in \overline{T}_{\overline{\Sigma}}(\mathcal{X}) \cdot L(t) \overset{*}{\longrightarrow}_R d\}$$

7.2 Soundness and Completeness of Dynamic Rewriting

In this section we give a version of Birkhoff's theorem which is suitable for the dynamic rewriting framework. This demonstrates that dynamic rewriting is sound and complete with respect to the dynamic order-sorted equational logic presented in Chapter 4. The properties of soundness and completeness hold for all the varieties of rewriting.

7.2.1 Soundness

If we restrict our dynamic rewrite relation to well-sorted dynamic terms then we are guaranteed to make only correctly sorted equalities between underlying ordinary terms and only in sorts which are valid for those terms. This is expressed in the following soundness result. Since weak rewriting is the largest of the (decidable) rewrite relations defined above it suffices to consider weak rewriting for the soundness result.

Theorem 7.29. Given a specification $\mathcal{S} = (\Sigma, E)$, and $d_1, d_2 \in \mathcal{DS}(\mathcal{X})$ if $d_1 \overset{*}{\longleftrightarrow}^{W}_{\Psi(E)} d_2$, and $s \in \mathcal{DS}(d_1) \cup \mathcal{DS}(d_2)$ then $E \vdash_\Sigma \phi(d_1) =_s \phi(d_2)$.

Proof (7.29). It will be sufficient to show that the one-step relation $\longleftrightarrow_{\Psi(E)}$ preserves the property. The transitive closure will follow by transitivity of $E \vdash_\Sigma$.

We give the proof for $d_1 \rightarrow^{D}_{p} d_2$, where $p \in O(d_1)$. A similar argument holds for $d_1 \overset{D}{p}\!\leftarrow d_2$. We consider cases on the length of path where rewriting takes place.

Sort Rewriting at Root. $d_1 \rightarrow^{S}_{l \rightarrow_S r, \sigma, \epsilon} d_1 \downarrow_\epsilon S$ for some $l \rightarrow_S r \in \Psi(E)$, where $r \sqsubseteq^{W}_{\sigma} d_1$. Thus $\forall s \in S$, $E \vdash_\Sigma \phi(l) =_s \phi(r)$, by definition of dynamic equations, and by substitutivity, $E \vdash_\Sigma \sigma(\phi(l)) =_s \sigma(\phi(r))$. Since $r \sqsubseteq^{W}_{\sigma} d_1$, $\phi(\sigma r) = \phi(d_1)$ and thus $E \vdash_\Sigma \phi(d_1) =_{\mathcal{LS}(\phi(d_1))} \sigma(\phi(r))$ by reflexivity. Thus by symmetry and transitivity $E \vdash_\Sigma \phi(d_1) =_s \phi(d_2)$. Also if $s \in \mathcal{DS}(d_1 \downarrow_\epsilon S)$, but $s \notin S$, then $s \in \mathcal{DS}(d_1)$. Then since $d_1 \in \mathcal{D}_\mathcal{S}(\mathcal{X})$, so $\forall s \in \mathcal{DS}(d_1)$, $\exists t \in T_\Sigma(\mathcal{X})$ such that $E \vdash_\Sigma t =_s \phi(d_1)$ and thus $E \vdash_\Sigma \phi(d_1) =_s \phi(d_2)$.

Term Rewriting at Root. $d_1 \rightarrow^{T}_{l \rightarrow_S r, \sigma, \epsilon} d_1[\epsilon \leftarrow \sigma r] \downarrow_\epsilon S \cup \mathcal{DS}(d_1|_\epsilon)$ for some $l \rightarrow_S r \in E$, where $l \sqsubseteq^{W}_{\sigma} d_1$. Thus $d_1 \rightarrow^{T} \sigma r \downarrow_\epsilon S \cup \mathcal{DS}(d_1) = d_2$. Consider $s \in S$.

If σ is the identity substitution, then $E \vdash_\Sigma \phi(l) =_s \phi(r)$, and thus $E \vdash_\Sigma \phi(d_1) =_s \phi(d_2)$. If σ is not the identity, then since σ is well-sorted, then $\phi(\sigma)$ is a $\mathcal{W}$-substitution and using the substitutivity rule $E \vdash_\Sigma \sigma(\phi(l)) =_s \sigma(\phi(r))$, and since $\phi(d_1) = \sigma(\phi(l))$ and $\phi(d_2) = \sigma(\phi(r))$, $E \vdash_\Sigma \phi(d_1) =_s \phi(d_2)$.

If $s \in \mathcal{DS}(d_2)$, but $s \notin S$ then $s \in \mathcal{DS}(d_1)$. Then, if $s' \in \mathcal{DS}(d_1)$, and d_1 is well-sorted, for some $t \in T_\Sigma(\mathcal{X})$ $E \vdash_\Sigma t =_{s'} \phi(d_1)$. From the above, for $s \in \mathcal{DS}(\sigma r)$ we have

$E \vdash_\Sigma \phi(d_1) =_s \phi(d_2)$. Thus by transitivity, $E \vdash_\Sigma \phi(d_1) =_{s'} \phi(d_2)$.

Sort Propagation at Root. If $d_1 \rightarrow_\epsilon^\Sigma d_2$, let $d_1 = f(d_1|_1, \ldots, d_1|_k)\bullet S_1$, then there is a rank for f in Σ, $f : s'_1, \ldots, s'_k \rightarrow s'$ such that $\exists s_j \in \mathcal{DS}(d_2|_j)$ which $s_1 \ldots s_k \leq s'_1, \ldots, s'_k$ and $\mathcal{DS}(d_1) \not\leq s'$. Since also $\forall j \cdot E \vdash_\Sigma d_1|_j =_s d_2|_j$, then by the Congruence rule $E \vdash_\Sigma d_1 =_{s'} d_2$. Again if $s \in \mathcal{DS}(d_1)$, then $E \vdash_\Sigma d_1 =_s d_2$ since d_1 is a well-sorted term.

Induction Step. Assume the theorem holds for dynamic rewriting at paths of up to length n. Let $d_1 = f(d_1|_1, \ldots, d_1|_k)\bullet S_1$ and p be a path of length $n+1$, then there is a path q of length n such that for some $1 \leq i \leq k$, $p = i.q$. If $d_1 \rightarrow_p^D d_2$, then $d_1|_i \rightarrow_q^D d_2|_i$. By the induction hypothesis, $\forall s \in \mathcal{DS}(d_1|_i) \cdot E \vdash_\Sigma \phi(d_1|_i) =_s \phi(d_2|_i)$. Hence, by Congruence $\forall s \in \mathcal{LS}(d_1) \cdot E \vdash_\Sigma \phi(d_1) =_s \phi(f(d_1|_1, \ldots, d_2|_i, \ldots, d_1|_k)\bullet S_1)$. □

7.2.2 Completeness

The dynamic rewriting relation restricted to well-sorted dynamic terms is also complete. It suffices to prove completeness for strict rewriting as this is the smallest rewriting relation.

Theorem 7.30. Completeness of Dynamic Rewriting.
Given a specification $\mathcal{S} = (\Sigma, E)$, and $R = \Psi(E)$ then $\forall t_1, t_2 \in T_\Sigma(\mathcal{X})$ if for some $s \in S_\Sigma$:

$$E \vdash_\Sigma \forall Y.t_1 =_s t_2$$

then

$$L(t_1) \overset{*}{\longleftrightarrow}_R^{St} L(t_2)$$

where there exists a *chain of intermediate terms* $d_i \in \mathcal{D}_\mathcal{S}(\mathcal{X})$ such that

$$L(t_1) = d_1 \longleftrightarrow_R^{St} d_2 \longleftrightarrow_R^{St} \cdots \longleftrightarrow_R^{St} d_{n-1} \longleftrightarrow_R^{St} d_n = L(t_2)$$

and for some d_i, $s \geq \mathcal{DS}(d_i)$.

Completeness is proven by induction over proof trees in the equational logic.

Proof (7.30). Assume that the strict rewriting relation is used.

Base Cases.

Reflexivity: $E \vdash_\Sigma \ \forall Y.t =_s t$ where $t : s$ then $L(t) \overset{0}{\longleftrightarrow}_R L(t)$, and if $t : s$, then $\mathcal{LS}(t) \leq s$ and so $\mathcal{DS}(L(t)) \leq s$. Note that this is trivially a strict match.

Direct Derivation: If $\forall Y.t_1 = t_2 \in E$ then $E \vdash_\Sigma \ \forall Y.t_1 =_s t_2$ for all $s \in S = \mathcal{LS}(t_1) \cup \mathcal{LS}(t_2)$. Let $l = L(t_1)$, $r = L(t_2)$, then $l \rightarrow_S r \in \Psi(E)$ and

$$L(t_1) \rightarrow^T_{l \rightarrow_S r, \iota, \epsilon} L(t_1)[\epsilon \leftarrow_\iota L(t_2)] \downarrow_\epsilon S = L(t_2) \downarrow_\epsilon S$$

where ι is the identity substitution, and also

$$L(t_2) \rightarrow^S_{l \rightarrow_S r, \iota, \epsilon} L(t_2) \downarrow_\epsilon S = L(t_2) \downarrow_\epsilon S$$

These are both strict rewritings, so we have:

$$L(t_1) \rightarrow^T_{l \rightarrow_S r, \iota, \epsilon} L(t_2) \downarrow_\epsilon S \ {}^S\!\leftarrow_{l \rightarrow_S r, \iota, \epsilon} L(t_2)$$

thus $L(t_1) \overset{*}{\longleftrightarrow}_R L(t_2)$ and we are done. Note the crucial role of sort rewriting in this step: without it the equivalence cannot be established.

Induction Steps.

Symmetry: Assume the theorem holds for $E \vdash_\Sigma \ \forall Y.t_1 =_s t_2$, so $L(t_1) \overset{*}{\longleftrightarrow} L(t_2)$, and there is an intermediate term d_i such that $\mathcal{DS}(d_i) \leq s$. Consider the symmetric inference $E \vdash_\Sigma \ \forall Y.t_2 =_s t_1$. Since the relation $\overset{*}{\longleftrightarrow}$ is symmetric, each step of the chain can be reversed, so $L(t_2) \overset{*}{\longleftrightarrow} L(t_1)$, and as the same terms are in the chain, d_i is in the chain.

Transitivity: Assume the theorem holds for $E \vdash_\Sigma \ \forall Y.t_1 =_s t_2$, and $E \vdash_\Sigma \ \forall Y.t_2 =_{s'} t_3$, so $L(t_1) \overset{*}{\longleftrightarrow} L(t_2)$, and there is an intermediate term d_i such that $\mathcal{DS}(d_i) \leq s$, and also $L(t_2) \overset{*}{\longleftrightarrow} L(t_3)$, and there is an intermediate term d'_j such that $\mathcal{DS}(d_j) \leq s'$. Consider the transitive inference $E \vdash_\Sigma \ \forall Y.t_1 =_s t_3$. Since the relation $\overset{*}{\longleftrightarrow}$ is transitive, $L(t_1) \overset{*}{\longleftrightarrow} L(t_3)$ with intermediate terms the union of the two sets of intermediate term. Hence, d_i is an intermediate term such that $\mathcal{DS}(d_i) \leq s$.

Congruence: Assume the theorem holds for $E \vdash_\Sigma \ \forall Y.t_i =_{s_i} t'_i$, for $i = 1 \ldots n$ so for each i, $L(t_i) \overset{*}{\longleftrightarrow} L(t'_i)$, and there is an intermediate term d_{ij_i} such that $\mathcal{DS}(d_{ij_i}) \leq s_i$. Consider the congruence inference $E \vdash_\Sigma \ \forall Y.f(t_1, \ldots t_n) =_s f(t'_1, \ldots, t'_n)$ where $(W, s) \in \mathit{ranks}(f)$ and $\langle s_1, \ldots, s_n \rangle \leq W$. Note that since d_{ij_i} is an intermediate term for $L(t_i) \overset{*}{\longleftrightarrow} L(t'_i)$,

$L(t_i) \overset{*}{\longleftrightarrow} d_{ij_i} \overset{*}{\longleftrightarrow} L(t_i')$. Hence we can generate the chain of dynamically equivalent terms:

$$\begin{aligned}
f(L(t_1),\ldots,L(t_n))\bullet A &\overset{*}{\longleftrightarrow} f(d_{1j_1},\ldots,L(t_n))\bullet A \\
&\overset{*}{\longleftrightarrow} f(d_{1j_1},d_{2j_2}\ldots,L(t_n))\bullet A \\
&\overset{*}{\longleftrightarrow} f(d_{1j_1},\ldots,d_{nj_n})\bullet A \\
&\overset{*}{\longleftrightarrow} f(d_{1j_1},\ldots,d_{nj_n})\bullet A\cup B \\
&\rightarrow^{\Sigma} f(d_{1j_1},\ldots,d_{nj_n})\bullet A\cup B\cup\{s\} \\
&{}^{\Sigma}\leftarrow f(d_{1j_1},\ldots,d_{nj_n})\bullet B\cup A \\
&\overset{*}{\longleftrightarrow} f(d_{1j_1},\ldots,d_{nj_n})\bullet B \\
&\overset{*}{\longleftrightarrow} f(L(t_1'),L(t_2'),\ldots,d_{nj_n})\bullet B \\
&\overset{*}{\longleftrightarrow} f(L(t_1'),\ldots,L(t_n'))\bullet B
\end{aligned}$$

where $A = \mathcal{LS}(f(t_1,\ldots,t_n))$ and $B = \mathcal{LS}(f(t_1',\ldots,t_n'))$. Note that

$$\begin{aligned}
f(d_{1j_1},\ldots,d_{nj_n})\bullet A &\overset{*}{\longleftrightarrow} f(L(t_1'),\ldots,L(t_n'))\bullet A \\
&\rightarrow^{\Sigma} f(L(t_1'),\ldots,L(t_n'))\bullet A\cup B \\
&\overset{*}{\longleftrightarrow} f(d_{1j_1},\ldots,d_{nj_n})\bullet A\cup B
\end{aligned}$$

and similarly for: $f(d_{1j_1},\ldots,d_{nj_n})\bullet B \overset{*}{\longleftrightarrow} f(d_{1j_1},\ldots,d_{nj_n})\bullet B\cup A$.

Thus $f(L(t_1),\ldots,L(t_n))\bullet A \overset{*}{\longleftrightarrow} f(L(t_1'),\ldots,L(t_n'))\bullet B$, with intermediate term d_i such that $\mathcal{DS}(d_i) \leq s$.

Substitutivity: Assume the theorem holds for $E \vdash_\Sigma \ \forall Y.t_1 =_s t_2$, so $L(t_1) \overset{*}{\longleftrightarrow} L(t_2)$, and there is an intermediate term d_i such that $\mathcal{DS}(d_i) \leq s$. By the substitutivity rule $E \vdash_\Sigma \ \forall Y.\sigma(t_1) =_s \sigma(t_2)$ where σ is a Σ-substitution. Note by Lemma 4.24, only Σ-substitutions need be considered. Let $\rho \in DSubst_{\mathcal{S}}$, such that $\forall x \in \mathcal{X}.\rho x = L(\sigma x)$ if $x \in Dom(\sigma)$ and $\rho x = x$ if $x \notin Dom(\sigma)$. Then $\forall t \in T_\Sigma(\mathcal{X}).L(\sigma t) = \rho(L(t))$. So

$$L(t_1) \longleftrightarrow d_1 \longleftrightarrow \ldots \longleftrightarrow d_i \longleftrightarrow \ldots \longleftrightarrow L(t_2)$$

Then by stability of strict rewriting:

$$L(\sigma t_1) = \rho(L(t_1)) \longleftrightarrow \rho d_1 \longleftrightarrow \ldots \longleftrightarrow \rho d_i \longleftrightarrow \ldots \longleftrightarrow \rho(L(t_2)) = L(\sigma t_2)$$

since ρ is a well-sorted substitution. Hence $L(\sigma t_1) \overset{*}{\longleftrightarrow} L(\sigma t_2)$, and $\mathcal{DS}(\rho d_i) \leq s$. □

We can prove the following lemma concerning the dynamic sorts of terms.

Corollary 7.31. Given a specification $\mathcal{S} = (\Sigma, E)$, and $t_1, t_2 \in T_\Sigma(\mathcal{X})$ if $E \vdash_\Sigma t_1 =_s t_2$, then $\exists d \in \mathcal{D}_\mathcal{S}(\mathcal{X})$ such that $L(t_1) \overset{*}{\longleftrightarrow}_{\Psi(E)} d$, $L(t_2) \overset{*}{\longleftrightarrow}_{\Psi(E)} d$, and $s \geq_\Sigma \mathcal{DS}(d)$, for all dynamic rewriting relations.

Proof (7.31). Directly from the completeness of the dynamic equivalence relation. □

7.2.3 Birkhoff's Theorem for Dynamic Rewriting

Thus there is a version of Birkhoff's theorem for dynamic rewriting of any kind: the relation generated by the equational logic and that generated by replacing equals by equals are in a strong sense equivalent over well-sorted terms.

Theorem 7.32. Birkhoff's Theorem for Dynamic Rewriting.
Given an order-sorted specification $\mathcal{S} = (\Sigma, E)$, the set of theorems

$$T_E = \{t = t' | E \vdash_\Sigma t =_s t'\}$$

is the same as the set of equalities resulting from replacing equals by equals,

$$T_D = \{\phi(d) = \phi(d') | d \overset{*}{\longleftrightarrow}_{\Psi(E)} d' \text{ where } d, d' \in \mathcal{D}_\mathcal{S}(\mathcal{X})\}$$

Proof (7.32). Immediate from the soundness and completeness of $\overset{*}{\longleftrightarrow}_{\Psi(E)}$. □

Thus the $\overset{*}{\longleftrightarrow}_{\Psi(E)}$ relation can be used for proofs in the equational logic. However, this relation is too general for constructing automated proofs as it has too large a search space and there is no control over its search. It is impractical to search for appropriate intermediate 'peak' terms. It would be preferable to use a rewriting relation only, and showing that the rewriting relation is strong enough to contain rewrite proofs of all equalities is the subject of the following sections.

7.3 Termination of Dynamic Rewriting

In this section the termination of the dynamic rewriting relations is discussed. It is assumed that any of the dynamic rewrite relations apply here.

Definition 7.33. Given a set of dynamic terms $E \subseteq \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ and a set of Dynamic Rewrite Rules R, then the associated term-rewriting relation $\rightarrow^D$ *terminates in* E (is *well-founded in* E) if there are no infinite chains of dynamic terms $\{d_i\}$ such that

$$d_1 \rightarrow^D d_2 \rightarrow^D d_3 \rightarrow^D d_4 \rightarrow^D \cdots$$

such that $d_i \in E$.

For example, $\rightarrow^D$ terminates in $\mathcal{D}_\mathcal{S}(\mathcal{X})$ if there are no such chains contained within $\mathcal{D}_\mathcal{S}(\mathcal{X})$. A conventional definition of a termination ordering applies.

Definition 7.34. A well-founded relation $>$ on $\overline{\mathcal{D}}_\Sigma(\mathcal{X})$ is a *dynamic termination ordering* if it has the following properties:

1. **Monotonicity.** $\forall f \in \mathcal{F}_\Sigma$ and $S \in I\!P(S_\Sigma)$, if $d_1 > d_2$, then $f(\cdots, d_1, \cdots)\bullet S > f(\cdots, d_2, \cdots)\bullet S$.
2. **Stability.** $\forall \sigma : \mathcal{X} \rightarrow \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, if $d_1 > d_2$, then $\sigma d_1 > \sigma d_2$.

An additional property of dynamic termination orderings is as follows.

Definition 7.35. A dynamic termination ordering $>$ is *coherent modulo sort* if given $d_1, d_2 \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ such that $d_1 \simeq d_2$ we have for any $d \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ such that $d \not\simeq d_1$:

1. if $d > d_1$ then $d > d_2$.
2. if $d_1 > d$ then $d_2 > d$.

As mentioned above, sort normalisation and propagation terminate given a finite signature. Thus ordinary term-orderings can be used to establish termination.

Definition 7.36. If $\geq_\Sigma$ on the sorts of $\mathcal{S}$ is a well-founded relation, and given a well-founded relation $>_t$ on $\overline{T}_{\overline{\Sigma}}(\mathcal{X})$, then the *dynamic extension* of $>_t$, $>_d$ on $\overline{\mathcal{D}}_\Sigma(\mathcal{X})$, is:

$$d_1 >_d d_2 \text{ if and only if } \phi(d_1) >_t \phi(d_2) \text{ or } d_1 \rhd d_2$$

Further, we write $d_1 >_t d_2$ if $\phi(d_1) >_t \phi(d_2)$.

Lemma 7.37. Given a well-founded ordering $>_t$ on $\overline{T}_{\overline{\Sigma}}(\mathcal{X})$, then the dynamic extension of $>_t$ to $\overline{\mathcal{D}}_\Sigma(\mathcal{X})$ is coherent modulo sorts.

Proof (7.37). Immediate from definitions of dynamic extension and ordering coherent modulo sorts. □

Lemma 7.38. For a well-founded relation $>_t$ on $\overline{T}_{\overline{\Sigma}}(\mathcal{X})$, the dynamic extension $>_d$ to $\overline{\mathcal{D}}_\Sigma(\mathcal{X})$ is also well-founded.

Proof (7.38). Note that if $d_1 \rhd d_2$, then $\phi(d_1) = \phi(d_2)$ so $\phi(d_1) \not>_t \phi(d_2)$ and also vice versa. Thus $>_d$ is the disjoint union of the two well-founded relations on $\overline{\mathcal{D}}_\Sigma(\mathcal{X})$. Assume that there is an infinite chain in $>_d$.

$$d_1 >_d d_2 >_d d_3 >_d d_4 >_d \ldots$$

since for a finite signature $\rhd$ is well-founded, then there are finite subchains in $\rhd$ within this chain, such as:

$$d_{i-1} >_t d_i \rhd d_{i+1} \rhd \ldots \rhd d_{i+n} >_t d_{i+n+1}$$

Since $>_d$ is coherent modulo sorts, we can replace this subchain by $d_{i-1} >_t d_i >_t d_{i+n+1}$. If this replacement is done to all subchains in $\rhd$, then we result in the infinite chain in $>_t$:

$$d_{i_1} >_t d_{i_2} >_t d_{i_3} >_t d_{i_4} >_t \ldots$$

which contradicts the assumption that $>_t$ is well-founded. □

Theorem 7.39. Given a termination ordering $>_t$ on $\overline{T}_{\overline{\Sigma}}(\mathcal{X})$, the dynamic extension $>_d$ to $\overline{\mathcal{D}}_\Sigma(\mathcal{X})$ is a dynamic termination ordering.

Proof (7.39). $>_d$ is well-founded from Lemma 7.38, and stable and monotonic from the stability and monotonicity of $>_t$ and $\rhd$. □

Note also that from the definition of sort-rewriting, and sort-propagation, for any rewrite relation R, $\rightarrow_R^S \subseteq \rhd$ and $\rightarrow_R^\Sigma \subseteq \rhd$. Consequently, to show termination of a dynamic rewriting relation, it is sufficient to show that the resolvent of the term-rewriting relation, $\Phi(\rightarrow_R^T)$, is contained in some well-founded ordering on terms $>_t$. In particular, if $>_t$ is a (monotonic and stable) termination ordering on $\overline{T}_\Sigma(\mathcal{X})$, then it is sufficient (but not necessary) to show that for each rule $l \rightarrow_S r \in R$ is such that $\phi(l) >_t \phi(r)$. Thus standard termination orderings such as the Knuth-Bendix Ordering, or the Recursive Path Ordering can be used to establish the termination of dynamic rewriting relations.

As discussed in [Gna92b], the added structure of order-sorted specifications allows termination to be established in cases where it cannot if the sorts of terms are ignored. It is conjectured that this can be extended to dynamic order-sorted rewriting. This problem is not considered further and remains an area of further research.

7.4 Church-Rosser Property and Confluence

To carry out automated rewrite proofs, we require that the rewrite system satisfies the Church-Rosser Property. We assume that any consistently applied dynamic rewriting relation is valid in the following, unless explicitly stated.

Definition 7.40. If $d, d_1, d_2 \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ then if $d_1 \ {}_R\!\xleftarrow{*} d \xrightarrow{*}_R d_2$ the terms form a *peak*, if $d_1 \ {}_R\!\leftarrow d \simeq d_2$ a *cliff* and if $d_1 \xrightarrow{*}_R d \ {}_R\!\xleftarrow{*} d_2$ a *valley*.

A proof of the equality of terms $t_1, t_2 \in T_\Sigma(\mathcal{X})$ is a *rewrite proof* if $L(t_1) \xrightarrow{*}_R d \ {}_R\!\xleftarrow{*} L(t_2)$, and a *rewrite proof modulo sorts* if $L(t_1) \xrightarrow{*}_R d_1 \simeq d_2 \ {}_R\!\xleftarrow{*} L(t_2)$.

The Church-Rosser property, and subsequent definitions are given relative to a give set of dynamic terms.

Definition 7.41. Church-Rosser Property.
Given a set of dynamic terms E, a set of dynamic rewrite rules R is *Church-Rosser on E* if given $d_1, d_2 \in E$, if $d_1 \xleftrightarrow{*}{}^D_R d_2$ then there exists $d \in E$ such that $d_1 \xrightarrow{*}{}^D_R d$, $d_2 \xrightarrow{*}{}^D_R d$

in E.

For example, we have the Church-Rosser property on the well-sorted terms if $E = \mathcal{D}_{\mathcal{S}}(\mathcal{X})$. Closely associated with the Church-Rosser property is the concept of confluence (Figure 7.1).

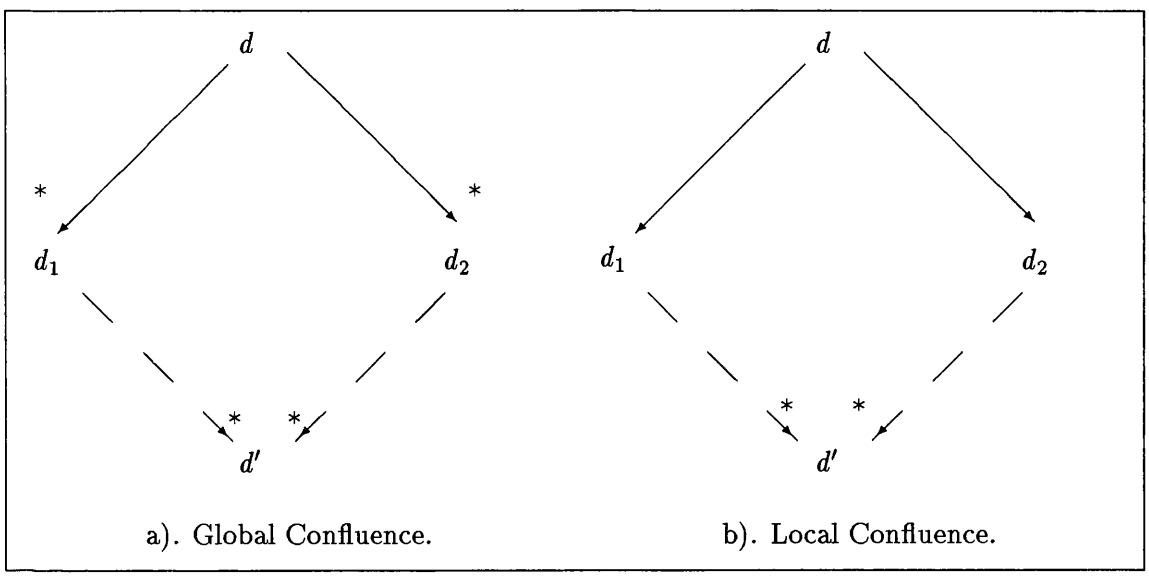


Figure 7.1: Confluence.

Definition 7.42. Confluence.

Given a set of dynamic terms E, a set of dynamic rewrite rules R is *Confluent on* E if given $d, d_1, d_2 \in E$, such that $d \xrightarrow{*}{}^{D}_{R} d_1$, $d \xrightarrow{*}{}^{D}_{R} d_2$ in E, then there exists $d' \in E$ such that $d_1 \xrightarrow{*}{}^{D}_{R} d'$, $d_2 \xrightarrow{*}{}^{D}_{R} d'$ in E.

Theorem 7.43. R is Church-Rosser on E if and only if R is Confluent on E.

Proof (7.43). The proof follows the standard proof method for showing the equivalence of the Church-Rosser and confluence properties.

Clearly, if R is Church-Rosser on E then it is confluent on E . To show the converse, assume that $d \xleftrightarrow{*} d'$ and consider the number of peaks in the chain:

$$d = d_1 \longleftrightarrow d_2 \longleftrightarrow \cdots \longleftrightarrow d_{n-1} \longleftrightarrow d_n = d'$$

where $d_i \in E$. Let a maximal peak in this chain be a triple (d_i, d_j, d_k), $1 \leq i < j < k \leq n$

be such that

$$d_i \leftarrow d_{i+1} \leftarrow \cdots \leftarrow d_{j-1} \leftarrow d_j \rightarrow d_{j+1} \rightarrow \cdots \rightarrow d_{k-1} \rightarrow d_k$$

and if $1 < i$, then $d_{i-1} \rightarrow d_i$ and if $k < n$ then $d_k \leftarrow d_{k+1}$. Let M be the number of such maximal peaks in this chain. If (d_i, d_j, d_k) is a maximal peak, then since R is confluent on E, there exists a term $d'_j \in E$ such that $d_i \xrightarrow{*} d'_j \xleftarrow{*} d_k$ in E. Form the new chain:

$$d_1 \xleftrightarrow{*} d_i \xrightarrow{*} d'_j \xleftarrow{*} d_k \xleftrightarrow{*} d_n$$

and the number of maximal peaks is now $M - 1$. This process can be repeated until the number of maximal peaks, and thus peaks, is zero, and this chain is of the form:

$$d_1 \xrightarrow{*} d \xleftarrow{*} d_n$$

for some term $d \in E$. This establishes the Church-Rosser Property on E. □

Again from standard rewriting theory we have the property of local confluence.

Definition 7.44. Local Confluence.
Given a set of dynamic terms E, a set of dynamic rewrite rules R is *Locally Confluent on* E if given $d, d_1, d_2 \in E$, such that $d \rightarrow^D_R d_1$, $d \rightarrow^D_R d_2$, then there exists $d' \in E$ such that $d_1 \xrightarrow{*}{}^D_R d'$, $d_2 \xrightarrow{*}{}^D_R d'$ in E.

Theorem 7.45. If R is terminating in E, it is confluent on E if and only if it is locally confluent on E.

Proof (7.45). Clearly if R is confluent on E it is locally confluent on E. We show that local confluence is equivalent to the Church-Rosser property. If R is terminating in E then there is a well-founded term $>$ ordering on dynamic terms such that $\rightarrow^D \subseteq >$ (just take $>$ to be $\rightarrow^D$). If $d_0 \xleftrightarrow{} d_1 \xleftrightarrow{} \cdots \xleftrightarrow{} d_{n-1} \xleftrightarrow{} d_n$ in E, then let M be the multiset of terms in the chain $\{d_i\}$. If for some i such that $1 \leq i < n$, $d_{i-1} \leftarrow d_i \rightarrow d_{i+1}$ is a peak in the chain, then since R is locally confluent in E replace d_i by $d_{i-1} \rightarrow d_{i1} \rightarrow \cdots d_{ij} \cdots \leftarrow d_{ik} \leftarrow d_{i+1}$ in E. Since for every $m \in 1..m$ $d_i \xrightarrow{*} d_{im}$ then $d_i > d_{im}$. Using the multiset extension of $>$, the multiset $M' = \{d_0, \ldots, d_{i-1}, d_{i1}, \ldots, d_{ik}, d_{i+1}, \ldots, d_n\}$ is less than M. Thus every peak reduction reduces the multiset in the multiset ordering. However, since this multiset

must be well-founded, there must be a minimum chain, which contains no peaks and must contain d_0 and d_n, and thus is a rewrite proof $d_0 \xrightarrow{*} d' \xleftarrow{*} d_n$ in E for some $d' \in E$. □

With dynamic term-rewriting systems, confluence not only guarantees unique normal forms and the Church-Rosser Property, but also the sorting of terms becomes decidable.

Theorem 7.46. If $\rightarrow_R$ is a dynamic rewrite relation confluent on $\mathcal{D}_{\mathcal{S}}(\mathcal{X})$ over specification $\mathcal{S}$, then

$$\forall t \in W_{\mathcal{S}}(\mathcal{X}).\mathcal{DS}(L(t)\downarrow_R) = \mathcal{SS}(t)$$

Proof (7.46). Consider a sort $s \in \mathcal{SS}(t)$. If $s \in \mathcal{LS}(t)$, then by Lemma 7.16, $\mathcal{DS}(L(t)\downarrow_R) \leq s$. But as $s \in \mathcal{SS}(t)$, then it is minimal, and thus $s \in \mathcal{DS}(L(t)\downarrow_R)$. If $s \notin \mathcal{LS}(t)$ then there is some $t' \in W_{\mathcal{S}}(\mathcal{X})$ such that $E \vdash_\Sigma t =_s t'$ and $s \in \mathcal{LS}(t')$. By completeness $\mathrm{L}(t) \xleftrightarrow{*}_R \mathrm{L}(t')$ and by confluence, $\mathrm{L}(t) \xrightarrow{*}_R L(t)\downarrow_R \ {}_R\xleftarrow{*} \mathrm{L}(t')$ in $\mathcal{D}_{\mathcal{S}}(\mathcal{X})$. Then again by Lemma 7.16 and minimality of $\mathcal{SS}(t)$, $s \in \mathcal{DS}(L(t)\downarrow_R)$. □

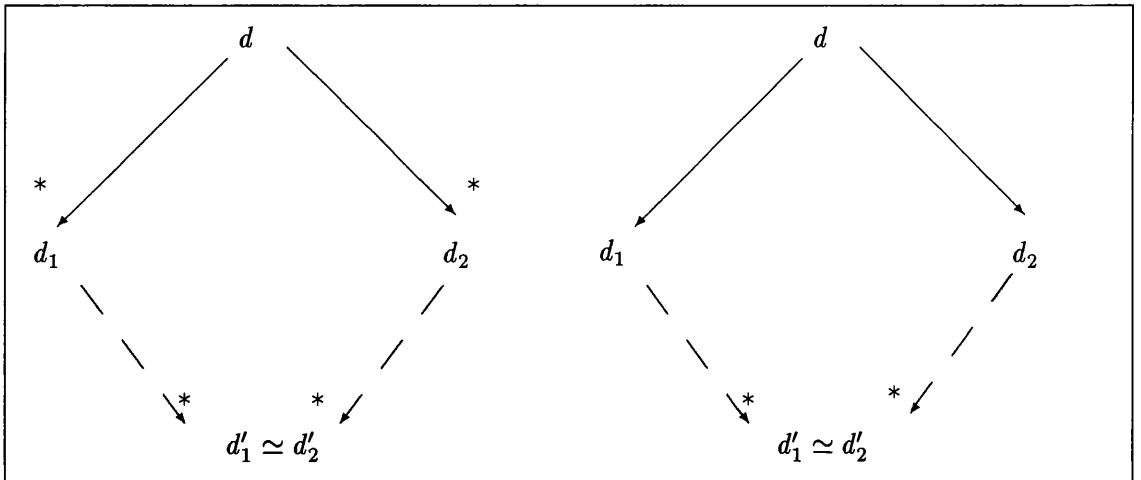
7.4.1 Church-Rosser Property and Confluence Modulo Sorts

An alternative approach to the previous section is not to insist on identical dynamic terms in rewriting proofs. Recall that the motivation of dynamic sorting is to decide equality theorems $t_1 = t_2$ in the underlying order-sorted algebra, using sorts as an aid to the computation. The previous section insists that the sort information derived in the rewrite proof is the same in both directions. This condition can be relaxed and still decide theorems in $T_\Sigma(\mathcal{X})$. The procedure would be to take the query equality $t_1 = t_2$, convert to dynamic terms using the conversion function L and then test whether the normal forms are equivalent modulo resolvents $L(t_1)\downarrow \simeq L(t_2)\downarrow$. This relaxation of the conditions leads to a simpler decision procedure.

Definition 7.47. Church-Rosser Property modulo sorts.
Given a set of dynamic terms E, a set of dynamic rewrite rules R is *Church-Rosser modulo sorts on* E if given $d_1, d_2 \in E$, such that $d_1 \xleftrightarrow{*}_R d_2$ in E then there are $d_1', d_2' \in E$ such

that $d_1 \xrightarrow{*}{}^{D}_{R} d_1' \simeq d_2' \,{}^{D}_{R}\!\xleftarrow{*} d_2$ in E.

Again the concept of confluence modulo sorts is closely associated with the Church-Rosser modulo sorts property.



a). Global Confluence modulo Sorts. b). Local Confluence modulo Sorts.

Figure 7.2: Confluence modulo sorts.

Definition 7.48. Confluence modulo sorts.
Given a set of dynamic terms E, a set of dynamic rewrite rules R is *Confluent modulo sorts on* E if a given $d, d_1 d_2 \in E$ such that $d \xrightarrow{*}{}^{D}_{R} d_1$, $d \xrightarrow{*}{}^{D}_{R} d_2$ in E, then there exist $d_1', d_2' \in E$ such that $d_1 \xrightarrow{*}{}^{D}_{R} d_1' \simeq d_2' \,{}^{D}_{R}\!\xleftarrow{*} d_2$ in E.

Confluence and the Church-Rosser property modulo sorts are equivalent if all terms have at least one normal form, as shown in the following theorem.

Theorem 7.49. If R is normalising in E, then it is Church-Rosser modulo sorts in E if and only if R is Confluent modulo sorts in E.

Proof (7.49). Clearly, if R is Church-Rosser modulo sorts on E then it is confluent modulo sorts on E. Recall the definition of maximal peak from the proof of Theorem 7.43. The proof proceeds by induction on the number of maximal peaks in the chain $d_0 \longleftrightarrow d_1 \longleftrightarrow \cdots d_{n-1} \longleftrightarrow d_n$ in E. If there is one maximal peak, then we have some term d such that $d_0 \xrightarrow{*} d_i \xleftarrow{*} d \xrightarrow{*} d_j \xleftarrow{*} d_n$. Then from the confluence modulo sorts of R

there are some $d'_0, d'_n \in E$ such that $d_0 \xrightarrow{*} d'_0 \simeq d'_n \xleftarrow{*} d_n$ in E.

Assume that the theorem holds for all chains with up to N maximal peaks. Consider a chain with $N+1$ maximal peaks, as in Figure 7.3. This chain is of the form:

$$d_0 \longleftrightarrow d_1 \longleftrightarrow \cdots d_{k-1} \rightarrow d_k \xleftarrow{*} d_i \xrightarrow{*} d_j \xleftarrow{*} d_n$$

where there are N maximal peaks in the chain $d_0 \xleftrightarrow{*} d_k$. Since R is normalising, this chain can be changed to one with the normal form of d_k.

$$d_0 \longleftrightarrow d_1 \longleftrightarrow \cdots d_{k-1} \rightarrow d_k \xrightarrow{*} d_k\!\downarrow \xleftarrow{*} d_k \xleftarrow{*} d_i \xrightarrow{*} d_j \xleftarrow{*} d_n$$

in E. Thus by the induction hypothesis there are terms $d'_0, d'_k \in E$ such that $d_0 \xrightarrow{*} d'_0 \simeq d'_k \xleftarrow{*} d_k\!\downarrow$ in E. Also by confluence modulo sorts there are terms $d''_k, d'_j \in E$ such that $d_k\!\downarrow \xrightarrow{*} d''_k \simeq d'_j \xleftarrow{*} d_j$ in E. Since $d_k\!\downarrow$ is a normal form, $d'_k = d_k\!\downarrow = d''_k$ and thus $d'_0 \simeq d_k\!\downarrow \simeq d'_j$ and we are done. □

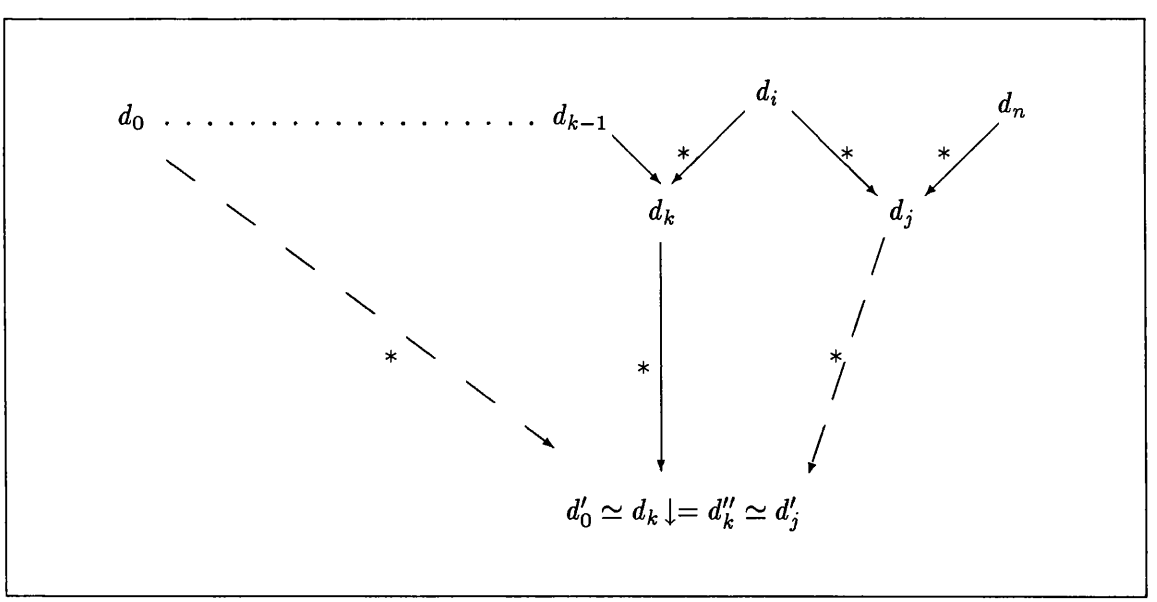


Figure 7.3: Church-Rosser and Confluence modulo Sorts

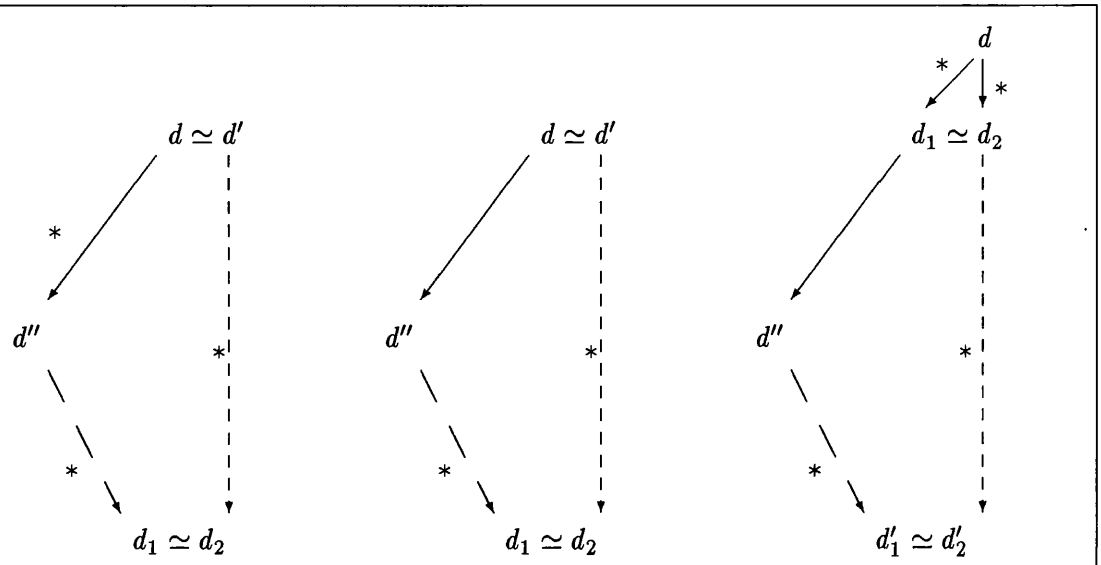
Similarly, we define local confluence property modulo sorts.

Definition 7.50. Local Confluence modulo sorts.

Given a set of dynamic terms E, a set of dynamic rewrite rules R is *Locally Confluent modulo sorts on* E if given $d, d_1, d_2 \in E$, such that $d \rightarrow^D_R d_1$, $d \rightarrow^D_R d_2$ in E, then there exist $d'_1, d'_2 \in E$ such that $d_1 \xrightarrow{*}{}^D_R d'_1$, $d_2 \xrightarrow{*}{}^D_R d'_2$ in E and $d'_1 \simeq d'_2$.

Additionally, R is *Normalised Locally Confluent modulo sorts in* E if given $d, d_1, d_2 \in E$, such that $d \rightarrow_R^D d_1$, $d \rightarrow_R^D d_2$ in E, then there exist *normal forms* $d_1 \downarrow_R, d_2 \downarrow_R \in E$ such that $d_1 \xrightarrow{*}_R^D d_1 \downarrow_R$, $d_2 \xrightarrow{*}_R^D d_2 \downarrow_R$ in E, and $d_1 \downarrow_R \simeq d_2 \downarrow_R$.

However, this property is not sufficient to give confluence modulo sorts. The extra property of coherence modulo sorts is needed.



a). Global Coherence modulo Sorts.

b). Local Coherence modulo Sorts.

c). Reachable Local Coherence modulo Sorts.

Figure 7.4: Coherence Modulo Sorts.

Definition 7.51. Coherence modulo sorts.

Given a set of dynamic terms E, a set of dynamic rewrite rules R is *Coherent modulo sorts on* E if given $d, d', d'' \in E$, such that $d' \simeq d \xrightarrow{*}^D d''$ in E, then there exist $d_1, d_2 \in E$ such that $d' \xrightarrow{*}^D d_1 \simeq d_2 {}^D\!\xleftarrow{*} d''$ in E.

Definition 7.52. Local Coherence modulo sorts.

Given a set of dynamic terms E, a set of dynamic rewrite rules R is *Local Coherent modulo sorts on* E if given dynamic terms $d, d', d'' \in E$, such that $d' \simeq d \rightarrow^D d''$ in E, then there exist $d_1, d_2 \in E$ such that $d' \xrightarrow{*}^D d_1 \simeq d_2 {}^D\!\xleftarrow{*} d''$ in E.

Definition 7.53. Reachable Local Coherence modulo sorts.

Given a set of dynamic terms E, a set of dynamic rewrite rules R is *Reachably Local Coherent modulo sorts on* E if given dynamic terms $d, d_1, d_2, d' \in E$, such that $d_1 \overset{D}{\xleftarrow{*}} d \xrightarrow{*}{}^{D} d_2$, and $d_2 \simeq d_1 \rightarrow^{D} d'$ in E, then there exist $d'_1, d'_2 \in E$ such that $d' \xrightarrow{*}{}^{D} d'_1 \simeq d'_2 \overset{D}{\xleftarrow{*}} d_2$ in E.

We give three theorems on coherence modulo sorts.

Theorem 7.54. Given a set of dynamic terms E, and a set of dynamic rewrite rules R, if $>$ is a well-founded coherent ordering on E and $R \subseteq >$ and R is coherent modulo sorts on E, then R is confluent modulo sorts on E if and only if it is locally confluent modulo sorts on E.

Proof (7.54). This proposition follows as a consequence of the next theorem. □

Theorem 7.55. Given a set of dynamic terms E, and a set of dynamic rewrite rules R, if $>$ is a well-founded coherent ordering on E such that $R \subseteq >$, and R is locally coherent modulo sorts on E, then R is confluent modulo sorts on E if and only if it is locally confluent modulo sorts on E.

Proof (7.55). Clearly if R is confluent modulo sorts on E it is locally confluent modulo sorts on E . We show that local confluence establishes the Church-Rosser property modulo sorts. Let $>_d$ be the well-founded coherent term ordering on E such that $\rightarrow^{D} \subseteq >$. If $\{d_k\}_{k=1...n}$ is a chain in $\longleftrightarrow \cup \simeq$ in E then let M be the multiset of terms in this chain.

Peak Reduction. If for some i such that $1 \leq i < n$, $d_{i-1} \leftarrow d_i \rightarrow d_{i+1}$ is a peak in the chain, then since R is locally confluent modulo sorts in E, we can build the new chain by replacing d_i by $d_{i-1} \rightarrow d_{i1} \rightarrow \cdots \rightarrow d_{ij} \simeq d'_{ik} \leftarrow \cdots \leftarrow d'_{i1} \leftarrow d_{i+1}$ in E. Since for every $m \in 1..j$ $d_i \xrightarrow{*} d_{im}$ and $m \in 1..k$ $d_i \xrightarrow{*} d'_{im}$ then $d_i > d_{im}$ and $d_i > d'_{im}$. Thus by multiset extension of $>$ the multiset $M' = \{d_0, \ldots, d_{i-1}, d_{i1}, \ldots, d_{ij}, d'_{ik}, \ldots, d'_{i1}, d_{i+1}, \ldots, d_n\}$ is less than M. Thus every peak reduction reduces the multiset in the multiset ordering.

Cliff Reduction.

If for some i such that $1 \leq i < n$, $d_{i-1} \leftarrow d_i \simeq d_{i+1}$ is a cliff in the chain, then since R is locally coherent modulo sorts in E, we can replace d_i by $d_{i-1} \rightarrow d_{i1} \rightarrow \cdots \rightarrow d_{ij} \simeq d'_{ik} \leftarrow \cdots \leftarrow d'_{i1} \leftarrow d_{i+1}$ in E. Since for every $m \in 1..j$ $d_i \xrightarrow{*} d_{im}$ and $m \in 1..k$ $d_i \simeq d_{i+1} \xrightarrow{*} d'_{im}$ then $d_i > d_{im}$ and $d_i > d'_{im}$ since $>$ is coherent modulo sorts. Thus by multiset extension of $>$ the multiset $M' = \{d_0, \ldots, d_{i-1}, d_{i1}, \ldots, d_{ij}, d'_{ik}, \ldots, d'_{i1}, d_{i+1}, \ldots, d_n\}$ is less than M. Thus every cliff reduction reduces the multiset in the multiset ordering.

Since $>$ is well-founded so must this multiset ordering and so there must be a minimum chain. This can contain no peaks or cliffs and must contain d_0 and d_n, and thus must be a rewrite proof $d_0 \xrightarrow{*} d'_1 \simeq d'_2 \xleftarrow{*} d_n$ in E for some $d'_1, d'_2 \in E$. □

Theorem 7.56. Given a set of dynamic terms E, and a set of dynamic rewrite rules R, if $>$ is a well-founded coherent ordering on E and $R \subseteq >$ and R is Reachably Locally Coherent modulo sorts on E, then R is confluent modulo sorts on E if and only if it is locally confluent modulo sorts on E.

Proof (7.56). In order to show the Church-Rosser theorem, it is sufficient to consider chains in $\longleftrightarrow \cup \simeq$ which are derived from chains in $\longleftrightarrow$ in E alone by peak reduction or cliff reduction. Cliff reduction preserves the number of occurrences of $\simeq$ in the chain. Thus any cliff in such a chain $d_{i-1} \leftarrow d_i \simeq d_{i+1}$ must have originally been inserted into the chain via peak reduction and thus there is a d such that $d \xrightarrow{*} d_i$ and $d \xrightarrow{*} d_{i+1}$. Thus in the proof of Theorem 7.55 it is sufficient to assume reachable local coherence. □

Note also that if the Church-Rosser theorem is restricted to proofs of the form: $L(t_1) \xleftrightarrow{*}_R L(t_2)$ where $t_1, t_2 \in \overline{T}_{\overline{\Sigma}}(\mathcal{X})$, then it is sufficient to have Local Coherence in $\mathcal{E}_R(\mathcal{X})$.

The condition of local coherence on Theorem7.55 can be dropped by assuming normalised local confluence modulo sorts.

Theorem 7.57. Given a set of dynamic terms E, and a set of dynamic rewrite rules R, if $>$ is a well-founded coherent ordering on E and $R \subseteq >$, then R is confluent modulo sorts on E if and only if it is normalised locally confluent modulo sorts on E.

Proof (7.57). Clearly if R is confluent modulo sorts on E it is normalised locally confluent modulo sorts on E. We show that normalised local confluence establishes the Church-Rosser property modulo sorts. Let $>_d$ be the well-founded coherent term ordering in E such that $\rightarrow^D \subseteq >$. If $\{d_k\}_{k=1...n}$ is a chain in $\longleftrightarrow \cup \simeq$ in E then let M be the multiset of terms in this chain.

If given a chain C where for some i such that $1 \leq i < n$, $d_{i-1} \leftarrow d_i \rightarrow d_{i+1}$ is a peak in the chain, then since R is normalised locally confluent modulo sorts in E we can build the chain C' by replacing d_i in C by $d_{i-1} \rightarrow d_{i1} \rightarrow \cdots \rightarrow d_{ij} \simeq d'_{ik} \leftarrow \cdots \leftarrow d'_{i1} \leftarrow d_{i+1}$ in E, where d_{ij} and d'_{ik} are in normal form. Since for every $m \in 1..j$ $d_i \xrightarrow{*} d_{im}$ and $m \in 1..k$ $d_i \xrightarrow{*} d'_{im}$ then $d_i > d_{im}$ and $d_i > d'_{im}$. Thus by multiset extension of $>$ the multiset $M' = \{d_0, \ldots, d_{i-1}, d_{i1}, \ldots, d_{ij}, d'_{ik}, \ldots, d'_{i1}, d_{i+1}, \ldots, d_n\}$ is less than M. Thus every peak reduction reduces the multiset in the multiset ordering. We say that chain C derives C', written $C \rightsquigarrow C'$.

Assume that $d_0 \xleftrightarrow{*} d_n$, with the initial chain of terms $C_0 = \{d_k\}_{k=0...n}$. We show that for any chain in C_k such that $C_0 \overset{*}{\rightsquigarrow} C_k$ there are no cliffs in C_k. Clearly there are no cliffs in C_0. Assume that there are no cliffs in C_i, and without loss of generality that there is a peak $d_{ij-1} \leftarrow d_{ij} \rightarrow d_{ij+1} \simeq d_{ij+2}$ in C_i, then by normalising local confluence, $d_{ij-1} \xrightarrow{*} d'_{ij-1} \simeq d'_{ij+1} \xleftarrow{*} d_{ij+1} \simeq d_{ij+2}$. However, since $C_0 \overset{*}{\rightsquigarrow} C_i$ d'_{ij+1} is in normal form, thus $d'_{ij+1} = d_{ij+1}$, and $d_{ij-1} \xrightarrow{*} d'_{ij-1} \simeq d_{ij+1} \simeq d_{ij+2}$, and no cliff is introduced into C_{i+1}. Cliffs can be introduced into C_k in no other way.

Since $>$ is well-founded in E, so must this multiset ordering and so there must be a minimum chain. This can contain no peaks or cliffs and must contain d_0 and d_n, and thus must be a rewrite proof $d_0 \xrightarrow{*} d'_1 \simeq d'_2 \xleftarrow{*} d_n$ in E for some $d'_1, d'_2 \in E$. □

However, in general normalised local confluence requires coherence conditions.

7.4.2 Establishing Coherence

Coherence is thus an important property of a dynamic rewriting system and it should be established before rewrite proofs can be given. One sufficient condition for coherence is Resolvent Confluence.

Definition 7.58. Given a set of dynamic rewrite rules R, a set of dynamic terms E is *rewrite closed with respect to* R if $\forall d \in E$ and $d \rightarrow_R d'$ then $d' \in E$.

Note that in particular the set of well-sorted dynamic terms $\mathcal{D}_{\mathcal{S}}(\mathcal{X})$ is rewrite closed.

Definition 7.59. Given a set of dynamic terms E, a set of dynamic rewrite rules R is *Resolvent Confluent on* E if given $d_1, d_2 \in E$, such that $d_1 \simeq d_2$, then $\exists d' \in E$ such that $d_1 \trianglerighteq d' \trianglelefteq d_2$ and $d_1 \xrightarrow{*}{}^{D}_{R} d' \overset{D}{R}\!\xleftarrow{*} d_2$ in E.

Lemma 7.60. Given a set of dynamic terms E, if a set of dynamic rewrite rules R is Resolvent Confluent on E, and E is rewrite closed with respect to R, then for weak and upward strong rewriting:

1. R is coherent in E.

2. If R is confluent modulo sorts on E, then it is confluent on E.

Proof (7.60). 1. If $d_1, d_2, d \in E$, such that $d_1 \simeq d_2 \xrightarrow{*}{}^{D} d$ in E, then by resolvent confluence, there is a $d' \in E$ such that $d_1 \trianglerighteq d' \trianglelefteq d_2$ and $d_1 \xrightarrow{*}{}^{D}_{R} d' \overset{D}{R}\!\xleftarrow{*} d_2$ in E. If $d_2 \rightarrow_{l \rightarrow_{sr}} d_i \xrightarrow{*} d$, d' must also weak or upward strong match using the same rule and therefore $d' \rightarrow_{l \rightarrow_{sr}} d'_i$ and $d'_i \trianglelefteq d_i$. This can be repeated and so $d' \xrightarrow{*} d'''$ in E, as E is rewrite closed, and $d'' \simeq d'$. Thus R is coherent in E.

2. If $d_1, d_2, d \in E$ are such that $d_1 \xleftarrow{*} d \xrightarrow{*} d_2$, then by confluence modulo sorts $\exists d'_1, d'_2 \in E$ such that $d_1 \xrightarrow{*} d'_1 \simeq d'_2 \xleftarrow{*} d_2$, in E and by resolvent confluence, $\exists d' \in E$ such that $d'_1 \xrightarrow{*} d' \xleftarrow{*} d'_2$. □

Coherence can also be established on restrictions to the rewrite relation. For example,

consider the rewrite relation restricted to Σ-substitutions (Definition 5.30).

Definition 7.61. Given a set of dynamic rewrite rules R, and matching algorithm X, the *X-L-rewrite relation* $\rightarrow^{X-L} \subseteq \rightarrow^{X}$ is the relation $\rightarrow^{X}$ restricted to Σ-substitutions. That is if $d_1 \rightarrow^{X-L}_{l \rightarrow_{Sr},\sigma} d_2$, then σ is a Σ-substitution, $\sigma : \mathcal{X} \rightarrow \mathcal{L}_{\Sigma}(\mathcal{X})$.

Lemma 7.62. The L-rewrite relation is coherent on $\overline{\mathcal{D}}_{\Sigma}(\mathcal{X})$, for weak or, if the rules are in canonical form, upward strong matching.

Proof (7.62). Consider term rewriting. For some $d_1, d_2, d_2' \in \overline{\mathcal{D}}_{\Sigma}(\mathcal{X})$, let $d_1 \simeq d_2 \rightarrow^{TX-L}_{l \rightarrow_{Sr},\sigma,p} d_2'$. If $d_1 \not\trianglelefteq L(d_1)$ then $d_1 \xrightarrow{*}^{\Sigma} P_{\Sigma}(d_1) \trianglelefteq L(d_1)$. So we can assume that $d_1 \trianglelefteq L(d_1)$. If σ is a Σ-substitution, then if $l \sqsubseteq^{X}_{\sigma} d_2|_p$, as $l = L(l)$, $l \sqsubseteq^{X}_{\sigma} L(d_2)|_p$ as σ is a Σ-substitution. $L(d_2) = L(d_1)$, so $l \sqsubseteq^{X}_{\sigma} L(d_1)|_p$ and as $d_1 \trianglelefteq L(d_1)$, for weak and upward strong matching, $l \sqsubseteq^{X}_{\sigma} d_1|_p$, so $d_1 \rightarrow^{TX-L}_{l \rightarrow_{Sr},\sigma,p} d_2' \simeq d_2'$. Sort rewriting and sort propagation are trivial, so L-rewriting is coherent. □

7.4.3 Local Confluence Results

In order to generate rewrite proofs, that is proofs which can be produced automatically by a rewriting system without any specific control, it is necessary for a rewrite system to be Church Rosser. That is, all proofs can be given via a rewriting sequence without any peaks. To analyse the peaks which can occur in rewriting, we consider the possible overlaps between rules to test the local confluence of such overlaps. Given a terminating rewrite system, this is sufficient to guarantee a Church Rosser rewrite system. We assume in this section that rewriting is on $\overline{\mathcal{D}}_{\Sigma}(\mathcal{X})$.

With three forms of rewriting in the dynamic rewriting relation, there are several overlaps to consider. However, for some of these overlaps, local confluence results can be given. Firstly, the sort propagation relation is locally confluent.

Lemma 7.63. Local Confluence of Sort Propagation.
$\forall d \in \overline{\mathcal{D}}_{\Sigma}(\mathcal{X})$, if $d_1 \,{}^{\Sigma}\!\leftarrow d \rightarrow^{\Sigma} d_2$, then there is a $d' \in \overline{\mathcal{D}}_{\Sigma}(\mathcal{X})$ such that $d_1 \xrightarrow{*}^{\Sigma} d' \,{}^{\Sigma}\!\xleftarrow{*} d_2$.

Proof (7.63). Immediate from the lemmas in Chapter 5. □

When the rewriting occurs at disjoint subterms, the relation is locally confluent whichever varieties of rewriting and matching are used.

Lemma 7.64. $\forall d \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, if $d_1 \overset{D}{p}\!\leftarrow d \rightarrow^D_q d_2$, and $q \bowtie p$, then there is a $d' \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ such that $d_1 \overset{*}{\longrightarrow}^D d' \,{}^D\!\overset{*}{\longleftarrow} d_2$.

Proof (7.64). Clearly, $d_1|_q = d|_q$, so $d_1|_q \rightarrow^D_\epsilon d_2|_q$, and thus

$$d_1 \rightarrow^D_q d_1[q \leftarrow d_2|_q] \;=\; d[p \leftarrow d_1|_p][q \leftarrow d_2|_q]$$

Similarly $d_2|_p = d|_p$, $d_2|_p \rightarrow^D_\epsilon d_1|_p$, and so:

$$d_2 \rightarrow^D_p d_2[p \leftarrow d_1|_p] \;=\; d[q \leftarrow d_2|_q][p \leftarrow d_1|_p]$$

Clearly $d[p \leftarrow d_1|_p][q \leftarrow d_2|_q] = d[q \leftarrow d_2|_q][p \leftarrow d_1|_p]$. □

An important class of terms which can be rewritten in two ways are the variable overlaps.

Definition 7.65. Given a set of dynamic rewrite rules R, a *variable overlap* between two rules $l_1 \rightarrow_{S_1} r_1$ and $l_2 \rightarrow_{S_2} r_2$ is a triple of dynamic terms (d, d_1, d_2) formed in two ways:

1. $d \rightarrow^T_{l \rightarrow_S r, \sigma, p} d_1$, and $d \rightarrow^D_q d_2$ where $\exists p' \in O(l)$ such that $l|_{p'} \in \mathcal{X}$ and $p.p' \leq q$.
2. $d \rightarrow^S_{l \rightarrow_S r, \sigma, p} d_1$, and $d \rightarrow^D_q d_2$ where $\exists p' \in O(r)$ such that $r|_{p'} \in \mathcal{X}$ and $p.p' \leq q$.

A variable overlap is *trivial* if there is a dynamic term d' such that $d_1 \overset{*}{\longrightarrow}_R d' \,{}_R\!\overset{*}{\longleftarrow} d_2$.

All variable overlaps are trivial whichever varieties of rewriting and matching are used.

Lemma 7.66. The Variable Overlap Lemma. Given a set of rules R, then in the dynamic rewriting relation $\rightarrow^D$, all variable overlaps are trivial.

Proof (7.66). For any matching algorithm M, let term $d_2 \overset{M}{l_2 \rightarrow r_2, \sigma_2, q} \!\leftarrow d \rightarrow^M_{l_1 \rightarrow r_1, \sigma_1, p} d_1$, where $p.p'.p'' = q$, and let $l_1|_{p'} = x \in \mathcal{X}$. As the set of variables of the two rules are disjoint, then we can let $\sigma = \sigma_1 o \sigma_2$.

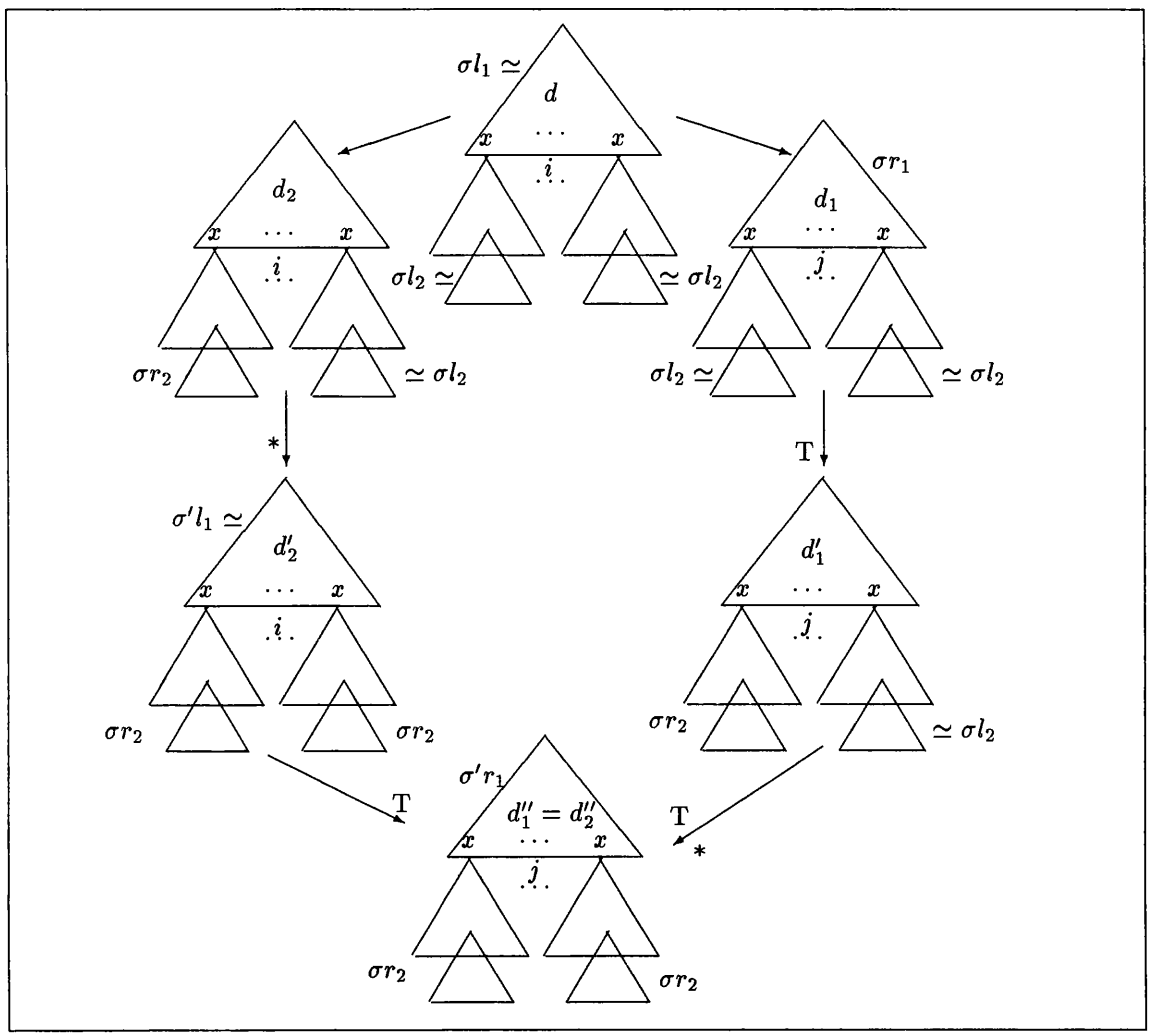


Figure 7.5: Variable Overlaps: Term-Term Case.

1. **Term Rewriting.** Since there is a match $l_1 \sqsubseteq^M d|_p$, then for all other paths $p'_i \in O(l_1)$ such that $l_1|_{p'_i} = x$, $d|_{p.p'_i} = d|_{p.p'}$ and thus $d|_{p.p'_i.p''} \rightarrow^D_\epsilon d_2|_q$. Hence d can be rewritten at these paths $d \rightarrow^D_q d_2 \rightarrow^D_{p.p'_1.p''} \ldots \rightarrow^D_{p.p'_k.p''} d'$. Also there is a match $l_1 \sqsubseteq^M_{\sigma'} d'_2|_p$, where $\sigma'(y) = \sigma(y)$ if $y \neq x$, and $\sigma'(x) = d'_2|_{p.p'}$. Thus $d'_2 \rightarrow^T_{l \rightarrow S^r, \sigma', p} d''_2$ and for each path q_j such that $r|_{q_j} = x$, $d''_2|_{p.q_j.p''} = d'_2|_q$

If $r|_{q_j} = x$ then $d_1|_{p.q_j.p''} = d|_q$ and at each path $p.q_j.q$ $d_1|_{p.q_j.q} \rightarrow^D_\epsilon d_2|_q$ and thus $d_1 \xrightarrow{*} d''_1$. $d''_1|_{p.q_j.p''} = d'_2|_q$, and thus $d''_2 = d''_1$. This case is shown pictorally in Figure 7.5

2. **The Sort Rewriting.** The cases of sort above sort and sort above propagation are covered in the lemmas above. If there is a sort-rewriting step with a term-rewriting step in a variable path below, then the proof is similar to the term-rewriting case. □

Sort propagation is also locally confluent in combination with dynamic sort-rewriting, when weak or upward strong matching is used.

Lemma 7.67. $\forall d \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$, if $d_1 \overset{S}{\leftarrow} d \rightarrow^{\Sigma} d_2$, then there is a $d' \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$ such that $d_1 \overset{*}{\longrightarrow}^{D} d' \overset{D}{\leftarrow}^{*} d_2$, for weak or upward strong rewriting.

Proof (7.67). Let $d \rightarrow^{S}_{l \rightarrow_{S} r,\sigma,p} d_1$, then $d_1 = d\downarrow_p S$ and let $d \rightarrow^{\Sigma}_{q} d_2$, then $d_2 = P_{q,\langle\omega,s\rangle}(d)$. If $p \bowtie q$ or else $q < p$ then clearly

$$d_2 = P_{q,\langle\omega,s\rangle}(d) \rightarrow^{S} P_{q,\langle\omega,s\rangle}(d)\downarrow_p S = P_{q,\langle\omega,s\rangle}(d\downarrow_p S) \overset{\Sigma}{\leftarrow} d\downarrow_p S = d_1$$

Also if $p \leq q$ then since $\mathcal{DS}(d_2) \leq \mathcal{DS}(d)$ there is still a match of r onto d_2 at p and d_2 can sort-rewrite. For downward strong or strict rewriting, there is no such match. □

Similarly for the combination of propagation and term rewriting.

Lemma 7.68. $\forall d \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$, if $d_1 \overset{T}{p}\leftarrow d \rightarrow^{\Sigma}_{q} d_2$, and $p \neq q$ then there is a $d' \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$ such that $d_1 \overset{*}{\longrightarrow}^{D} d' \overset{D}{\leftarrow}^{*} d_2$, for weak or upward strong rewriting.

Proof (7.68). Let $d \rightarrow^{T}_{l \rightarrow_{S} r,\sigma,p} d_1$, then $d_1 = d[p \leftarrow \sigma r]\downarrow_p S \cup \mathcal{DS}(d|_p)$, and let $d \rightarrow^{\Sigma}_{q,\langle\omega,s\rangle} d_2$, then $d_2 = P_{q,\langle\omega,s\rangle}(d)$. If $p \bowtie q$ then Lemma 7.64 applies.

If $p < q$, $q = p.p'$ say (term above propagation), and $l|_{p'}$ is not at or below a variable path, then since $d[p \leftarrow \sigma r]\downarrow_p S \cup \mathcal{DS}(d|_p)|_q = \sigma r\downarrow_\epsilon S \cup \mathcal{DS}(d|_p)|_{p'} = \sigma r|_{p'}$, there is no propagation step. However, $d_2 \rightarrow^{T}_{l \rightarrow_{S} r,\sigma,p} d_2[p \leftarrow \sigma r]\downarrow_p S \cup \mathcal{DS}(d_2|_p)$ since the propagation has taken place in a non-variable path in l, $\sigma l \simeq d_2|_p$. Thus

$$\begin{aligned} d_2 \;=\; & P_{q,\langle\omega,s\rangle}(d) \rightarrow^{T} P_{q,\langle\omega,s\rangle}(d)[p \leftarrow \sigma r]\downarrow_p S \cup \mathcal{DS}(P_{q,\langle\omega,s\rangle}(d)|_p) = \\ & d[p \leftarrow \sigma r]\downarrow_p S \cup \mathcal{DS}(d|_p) = d_1 \end{aligned}$$

If $q < p$ then $p = q.q'$ say (propagation above term). If $d|_p$ is an immediate subterm of $d|_q$ then $q' = i$ for some $i \in \mathbb{N}$, since $\mathcal{DS}(d_1|_p) \leq S \cup \mathcal{DS}(d|_p) \leq \mathcal{DS}(d|_p)$, the sort propagation step still applies, so:

$$d_1 \rightarrow^{\Sigma}_{q,\langle\omega,s\rangle} P_{q,\langle\omega,s\rangle}(d_1)$$

Clearly the term rewriting step applies to the (unchanged) subterm $d_2|_p$, so:

$$\begin{aligned} d_2 &= P_{q,\langle\omega,s\rangle}(d) \rightarrow^T P_{q,\langle\omega,s\rangle}(d)[p \leftarrow \sigma r]\downarrow_p S \cup \mathcal{DS}(P_{q,\langle\omega,s\rangle}(d)|_p) = \\ & P_{q,\langle\omega,s\rangle}(d[p \leftarrow \sigma r]\downarrow_p S \cup \mathcal{DS}(d|_p)) \stackrel{\Sigma}{\leftarrow} d[p \leftarrow \sigma r]\downarrow_p S \cup \mathcal{DS}(d|_p) = d_1 \end{aligned}$$

and we are done. If $q' = i.r$ for some non-empty path r, then as $\mathcal{DS}(d_1|_{q.j}) = \mathcal{DS}(d|_{q.j})$ for all $j \in 1 \ldots |\omega|$, the propagation step still applies.

The variable overlap cases hold via Lemma 7.66. □

Propagation and term-rewriting are not locally confluent if they are both applied at the same path as the following example demonstrates.

Example 7.69. Consider the following specification:

Sorts:	$A\ B$	
Subsorts:	$A \leq B$	
Operators:	$a : \rightarrow A$	$b : \rightarrow B$
	$f : A \rightarrow A$	$f : B \rightarrow B$
Rules:	$f(x : B) \bullet B \rightarrow_B b \bullet B$	

Then the following non-confluent peak is formed by rewriting two ways at root.

$$b \bullet A \stackrel{T}{\leftarrow} f(a \bullet A) \bullet A \stackrel{\Sigma}{\epsilon} \leftarrow f(a \bullet A) \bullet B \rightarrow^T_\epsilon b \bullet B$$

□

The sort-rewriting relation is locally confluent, for weak and upward strong rewriting.

Lemma 7.70. Local Confluence of Sort Rewriting.
$\forall d \in \mathcal{D}_\mathcal{S}(\mathcal{X})$, if $d_1 \stackrel{S}{\leftarrow}_{l_1 \rightarrow_{S_1} r_1, \sigma_1, p} d \rightarrow^S_{l_2 \rightarrow_{S_2} r_2, \sigma_2, q} d_2$, then there is a $d' \in \mathcal{D}_\mathcal{S}(\mathcal{X})$ such that $d_1 \stackrel{*}{\longrightarrow}^S d' \stackrel{S}{\leftarrow}^* d_2$, for weak or upward strong rewriting.

Proof (7.70). $d_1 = d\downarrow_p S_1$ and $d_2 = d\downarrow_q S_2$. We consider paths p and q. If $p \bowtie q$ then clearly $d\downarrow_p S_1|_q = d|_q$ and $d\downarrow_q S_2|_p = d|_p$, so the other rewrite still applies to give us:

$$d\downarrow_p S_1 \rightarrow^S d\downarrow_p S_1 \downarrow_q S_2 = d\downarrow_q S_2 \downarrow_p S_1 \stackrel{S}{\leftarrow} d\downarrow_q S_2$$

If $p \leq q$, then clearly $d\downarrow_p S_1|_q = d|_q$, so $d\downarrow_p S_1 \rightarrow^S d\downarrow_p S_1\downarrow_q S_2$. Also, since $\mathcal{DS}(d\downarrow_q S_2|_q) \leq \mathcal{DS}(d|_q)$, there is still a match of r_1 on $d\downarrow_q S_2$, so $d\downarrow_q S_2 \rightarrow^S d\downarrow_q S_2\downarrow_p S_1$. If downward strong or strict rewriting is used, then the matches are not necessarily preserved. □

One other combination is also locally confluent for weak and upward strong rewriting: a term-rewrite above a sort-rewrite.

Lemma 7.71. If $d_1 \overset{T}{l\rightarrow_{S'\tau,\sigma,p}} \leftarrow d \rightarrow \overset{S}{l'\rightarrow_{S''\tau',\sigma',q}} d_2$ and $p < q$, then $d_2 \rightarrow \overset{T}{l\rightarrow_{S'\tau,\sigma,p}} d_1$, for weak and upward strong rewriting.

Proof (7.71). If $p < q$, $q = p.p'$ say, and $l|_{p'}$ is not at or below a variable position, then since $d[p\leftarrow\sigma r]\downarrow_p S \cup \mathcal{DS}(d|_p)|_q = \sigma r\downarrow_\epsilon S \cup \mathcal{DS}(d|_p)|_{p'} = \sigma r|_{p'}$, there can be no sort-rewriting step. However, $d_2 \rightarrow \overset{T}{l\rightarrow_{S'\tau,\sigma,p}} d_2[p\leftarrow\sigma r]\downarrow_p S \cup \mathcal{DS}(d_2|_p)$ since the sort-rewriting has taken place in a non-variable position in l, $\sigma l \simeq d_2|_p$. Thus

$$d_2 = d\downarrow_q S' \rightarrow^T d\downarrow_q S'[p\leftarrow\sigma r]\downarrow_p S \cup \mathcal{DS}(d\downarrow_q S'|_p) = d[p\leftarrow\sigma r]\downarrow_p S \cup \mathcal{DS}(d|_p) = d_1$$

□

	Rewrite Relation			
Overlap	Weak	Upward Strong	Downward Strong	Strict
Variable	T	T	T	T
Disjoint ($p \bowtie q$)	T	T	T	T
$\Sigma - \Sigma$	T	T	T	T
$\Sigma \leq S$	T	T	T	T
$\Sigma < T$	T	T	T	T
$S - S$	T	T	N	N
$S \leq \Sigma$	T	T	N	N
$S \leq T$	N	N	N	N
$T - T$	N	N	N	N
$T = \Sigma$	N	N	N	N
$T < \Sigma$	T	T	N	N
$T \leq S$	T	T	N	N

Figure 7.6: Summary of Local Confluence Properties.

These results are summarised in the table in Figure 7.6. The left-hand column represents the form of overlap, T for term-rewriting, S for sort-rewriting and Σ for sort-propagation, with $A \leq B$ representing a rewrite of kind A above (at a subpath of) a rewrite of sort B, with $A < B$ if the overlap is strict. In addition there are the special cases of variable

and disjoint path overlaps. The entries in the table represent the behaviour of each kind of overlap under the four decidable dynamic matching algorithms. A 'T' entry marks that overlaps are trivial, whilst a 'N' entry stands for non-trivial. The weak and upward strong rewrite relations converge in more cases than the downward strong and strict relations.

7.4.4 Local Confluence Modulo Sorts Results

Further results can be established if confluence modulo sorts is considered. The combination of propagation and term-rewriting has a full result.

Lemma 7.72. $\forall d \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$, if $d_1 \overset{T}{p}\!\leftarrow d \rightarrow^{\Sigma}_{q} d_2$, and $p \neq q$ then there are $d', d'' \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$ such that $d_1 \xrightarrow{*}^{D} d' \simeq d''\, {}^{D}\!\xleftarrow{*} d_2$, for weak or upward strong rewriting.

Proof (7.72). If $p \neq q$ then the result is immediate from Lemma 7.68. If $p = q$ then without loss of generality, let $p = \epsilon$. Then $d_1 = \sigma r \downarrow_\epsilon S \cup \mathcal{DS}(d)$ for some rule $l \rightarrow_{\mathcal{S}} r$,and $d_2 = d \downarrow_\epsilon s \cup \mathcal{DS}(d)$ for some s. Thus l still weak or upward strong matches with d_2, and $d_2 \rightarrow^{T} \sigma r \downarrow_\epsilon S \cup \mathcal{DS}(d) \cup s \simeq d_1$ □

Also, sort rewriting above term rewriting is also confluent modulo sorts, for weak and upward strong matching.

Lemma 7.73. $\forall d \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$, if $d_1 \overset{T}{p}\!\leftarrow d \rightarrow^{S}_{q} d_2$, and $q \leq p$ then there are $d', d'' \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$ such that $d_1 \xrightarrow{*}^{D} d' \simeq d''\, {}^{D}\!\xleftarrow{*} d_2$, for weak or upward strong rewriting.

Proof (7.73). If $q < p$ then without loss of generality, let $q = \epsilon$. Then $d_1 = d[p \leftarrow \sigma r] \downarrow_p S \cup \mathcal{DS}(d|_p)$ for some rule $l \rightarrow_{\mathcal{S}} r$ and $d_2 = d \downarrow_\epsilon S' \cup \mathcal{DS}(d)$ for some rule $l' \rightarrow_{\mathcal{S}} r'$. Thus l still weak or upward strong matches with d_2, and $d_2 \rightarrow^{T} d[p \leftarrow \sigma r] \downarrow_p S \cup S' \cup \mathcal{DS}(d) \cup \mathcal{DS}(d|_p) \simeq d_1$. □

A further interesting result considers the application of a balancing step as an additional form of rewriting. This is not necessary for the completeness of rewriting. Nevertheless, balancing is a sound operation, and can be added as an efficiency gain in a rewriting engine.

Unfortunately, it does lead to non-confluence of rewriting.

Definition 7.74. A $d \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, unbalanced at paths $p, q \in O(d)$, *Balance Rewrites* to $B_{p,q}(d)$, written $d \rightarrow^B_{p.q} B_{p,q}(d)$.

Example 7.75. Consider the following specification.

Sorts:	$A\ B$	
Subsorts:	$A \leq B$	
Operators:	$a : \rightarrow A$	$b : \rightarrow B$
	$c : \rightarrow B$	$g : B \rightarrow B$
	$f : B\ B \rightarrow B$	
Rules:	$g(x : B) \bullet B \rightarrow_B c \bullet B$	

If the rewriting relation includes balancing, then there is the following non-confluent peak.

$$f(c\bullet B, b\bullet B)\bullet B \leftarrow f(g(b\bullet A)\bullet B, b\bullet B)\bullet B \rightarrow^B f(g(b\bullet A)\bullet B, b\bullet A)\bullet B \rightarrow f(c\bullet B, b\bullet A)\bullet B$$

□

However, this is not the case for confluence modulo sorts.

Lemma 7.76. $\forall d \in \mathcal{D}_\mathcal{S}(\mathcal{X})$, if $d_1 \overset{D}{_p\leftarrow} d \rightarrow^B_{q,q'} d_2$, then there are $d', d'' \in \mathcal{D}_\mathcal{S}(\mathcal{X})$ such that $d_1 \overset{*}{\longrightarrow}^D d' \simeq d''\ ^D\overset{*}{\longleftarrow} d_2$, for weak or upward strong rewriting.

Proof (7.76). If $p \bowtie q$ and $p \bowtie q$ then clearly the same proof as Lemma 7.64 applies.

If without loss of generality $q < p$ (balance above rewrite) then clearly the match still applies at p on d_2, and hence $d_2 \rightarrow^D_p d' \simeq d_1$.

If without loss of generality $p \leq q$ (rewrite above balance), as we are weakly or upward strongly matching, the match still applies at p and hence $d_2 \rightarrow^D_p d' \simeq d_1$. □

Chapter 8

Completion of Dynamic Systems

In this chapter we investigate the generation of Church-Rosser dynamic rewriting systems. This is done using a completion procedure which, if successful, generates a decision procedure which can generate automated rewrite proofs in the underlying equational theory. However, this completion procedure is contingent on the well-formedness of unifiers and the coherence of rewriting. We discuss alternative approaches to establishing these conditions.

8.1 Critical Pairs

To establish (local) confluence we examine terms which rewrite in alternative ways. Such terms are instances of the superposition of the terms in the rewrite rules involved and thus an instance of a *Critical Pair.*

8.1.1 Generating Critical Pairs

Critical pairs can be formed in nine different ways, depending on the combination of Term, Sort or Propagation rewriting used.

Definition 8.1. Term-Term Critical Pairs.

Given dynamic rewrite rules $\forall X.l_1 \rightarrow_{S_1} r_1$, $\forall Y.l_2 \rightarrow_{S_2} r_2$ in a set of dynamic rewrite rules R, if there is a path $p \in O(l_1)$, such that $l_1|_p \notin Vars(l_1)$, and a most general dynamic unifier σ such that $\sigma(l_1|_p) \simeq \sigma l_2$, then the terms $\sigma r_1 \downarrow_\epsilon S_1 \cup \mathcal{DS}(\sigma l_1)$ and $\sigma l_1[p \leftarrow r_2] \downarrow_p S_2 \cup \mathcal{DS}(\sigma l_1|_p)$ form a *Term-Term Critical Pair*, written:

$$\forall X \cup Y.\sigma r_1 \downarrow_\epsilon S_1 \cup \mathcal{DS}(\sigma l_1) = \sigma l_1[p \leftarrow r_2] \downarrow_p S_2 \cup \mathcal{DS}(\sigma l_1|_p)$$

The *Critical Term* of the pair is the term: $\sigma l_1[p \leftarrow \sigma l_1|_p \wedge \sigma l_2]$.

Critical pairs can also be generated when right-hand sides overlap and thus can be sort-rewritten in two different ways.

Definition 8.2. Sort-Sort Critical Pairs.

Given dynamic rewrite rules $\forall X.l_1 \rightarrow_{S_1} r_1$, $\forall Y.l_2 \rightarrow_{S_2} r_2$ in a set of dynamic rewrite rules R, if there is a path $p \in O(r_1)$, such that $r_1|_p \notin Vars(r_1)$, and a most general dynamic unifier σ such that $\sigma(r_1|_p) \simeq \sigma r_2$, then the terms $\sigma r_1 \downarrow_\epsilon S_1$ and $\sigma r_1 \downarrow_p S_2$ form a *Sort-Sort Critical Pair*, written:

$$\forall X \cup Y.\sigma r_1 \downarrow_\epsilon S_1 = \sigma r_1 \downarrow_p S_2$$

The *Critical Term* of the pair is the term: $\sigma r_1[p \leftarrow \sigma r_1|_p \wedge \sigma r_2]$.

Critical pairs can also be formed by a combination of term and sort rewriting, when either an instance of a left-hand side can be sort rewritten at a subterm, or an instance of a right-hand side can be term rewritten at a subterm.

Definition 8.3. Term-Sort Critical Pairs.

Term-Sort Critical Pairs can be generated in two ways. The dynamic rewrite rules $\forall X.l_1 \rightarrow_{S_1} r_1$, $\forall Y.l_2 \rightarrow_{S_2} r_2$ in a set of dynamic rewrite rules R form the following critical pairs.

1. *Term rewriting at root: sort rewriting at a subterm.* If there is a path $p \in O(l_1)$, such that $l_1|_p \notin Vars(l_1)$, and a most general dynamic unifier σ such that $\sigma(l_1|_p) \simeq \sigma r_2$, then the terms $\sigma r_1 \downarrow_\epsilon S_1 \cup \mathcal{DS}(\sigma l_1)$ and $\sigma l_1 \downarrow_p S_2$ form a *Term-Sort Critical Pair of Type 1,* written:

   $$\forall X \cup Y.\sigma r_1 \downarrow_\epsilon S_1 \cup \mathcal{DS}(\sigma l_1) = \sigma l_1 \downarrow_p S_2$$

The *Critical Term* of the pair is the term: $\sigma l_1[p \leftarrow \sigma l_1|_p \wedge \sigma r_2]$.

2. *Sort rewriting at root: term rewriting at a subterm.* If there is a path $q \in O(r_2)$, such that $r_2|_q \notin Vars(r_2)$, and a most general dynamic unifier σ such that $\sigma(r_2|_q) \simeq \sigma l_1$, then the terms $\sigma r_2 \downarrow_\epsilon S_2$ and $\sigma r_2[q \leftarrow \sigma r_1] \downarrow_q S_1 \cup \mathcal{DS}(\sigma r_2|_q)$ form a *Term-Sort Critical Pair of Type 2,* written:

   $$\forall X \cup Y. \sigma r_2 \downarrow_\epsilon S_2 = \sigma r_2[q \leftarrow \sigma r_1] \downarrow_q S_1 \cup \mathcal{DS}(\sigma r_2|_q)$$

   The *Critical Term* of the pair is the term: $\sigma r_2[p \leftarrow \sigma r_2|_q \wedge \sigma l_1]$.

From Lemmas 7.63, 7.67 and 7.68 it can be seen that certain critical pairs involving sort propagation are unnecessary. Consequently, we define only the following critical pairs.

Definition 8.4. Propagation Critical Pairs.
Propagation critical pairs are generated when a propagation step can be applied to a rule. Given a dynamic rewrite rule $\forall X. l \rightarrow_S r$, and a rank for operator $f \in \mathcal{F}_\Sigma$, $f : s_1 ... s_n \rightarrow s$, we can form the following critical pairs.

1. *Term rewriting at root: propagation at a subterm.* If there is a path $p \in O(l)$, such that there is a unifier σ such that $\sigma l|_p \simeq \sigma f(z_1 : s_1, \ldots, z_n : s_n) \bullet s$, for new variables $z_1, \ldots, z_n$, then the terms $\sigma l \downarrow_p s$ and σr form a *Term Propagation Critical Pair* written:

   $$\forall X. \sigma l \downarrow_p s = \sigma r$$

   The *Critical Term* of the pair is the term: $\sigma l \downarrow_p s$.

2. *Sort rewriting at root: propagation at a subterm.* If there is a path $q \in O(r)$, such that there is a unifier σ such that $\sigma r|_p \simeq f(z_1 : s_1, \ldots, z_n : s_n) \bullet s$, for new variables $z_1, \ldots, z_n$, then the terms $\sigma r \downarrow_p S$ and σl form a *Sort Propagation Critical Pair* written:

   $$\forall X. \sigma r \downarrow_p s = \sigma r \downarrow_\epsilon S$$

   The *Critical Term* of the pair is the term: $\sigma r \downarrow_p s$

Any of these critical pairs can be trivial.

Definition 8.5. A critical pair (d_1, d_2) is *trivial* if $\exists d' \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ such that

$$d_1 \xrightarrow{*}_R d' \; {}_R\!\xleftarrow{*} d_2.$$

A critical pair (d_1, d_2) is *trivial modulo sorts* if $\exists d_1', d_2' \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ such that

$$d_1 \xrightarrow{*}_R d_1' \simeq d_2' \; {}_R\!\xleftarrow{*} d_2.$$

and *normalised trivial modulo sorts* if d_1', d_2' are in normal form.

Previous lemmas show that some of the critical pairs generated will always be trivial. In particular, all sort-sort critical pairs and all term-sort critical pairs of type 1 are trivial, for weak and upward strong matching.

Lemma 8.6. All Sort-Sort critical pairs are trivial for weak and upward strong matching.

Proof (8.6). Immediate from Lemma 7.70, the local confluence of sort-rewriting. □

Lemma 8.7. All term-sort critical pairs of type 1 are trivial for weak and upward strong matching.

Proof (8.7). Immediate from Lemma 7.71. □

Propagation can also be excluded in nearly all cases.

Lemma 8.8. All propagation critical pairs are trivial for weak and upward strong matching, except for the case $c_1 \; {}^{\Sigma}_{p}\!\leftarrow c \rightarrow^{T}_{p} c_2$ for some path p.

Proof (8.8). Immediate from Lemmas 7.67 and 7.68, which excludes the given case. □

Consequently, sort-sort critical pairs and term-sort critical pairs of type 1, can be disregarded, as can all propagation pairs except those where a term rewriting and propagation occur at the same path. Without confusion, by sort critical pairs we mean term-sort critical pairs of type 2, and by propagation pairs those where term and propagation rewriting occur at the same path, unless explicitly stated otherwise.

Definition 8.9. Given a set of rules R, the set of all term, sort and propagation critical

pairs is denoted $CP(R)$.

If we consider confluence modulo sorts, then we can make further observations concerning critical pairs.

Lemma 8.10. All term-sort critical pairs of type 2 are trivial modulo sorts for weak and upward strong matching.

Proof (8.10). Immediate from Lemma 7.73. □

Lemma 8.11. All propagation critical pairs are trivial modulo sorts for weak and upward strong matching.

Proof (8.11). Immediate from Lemmas 8.8 and 7.72. □

Thus for confluence modulo sorts we need only consider Term-Term critical pairs.

Definition 8.12. Given a set of rules R, the set of all term-term critical pairs is denoted $CP_{\simeq}(R)$.

8.1.2 The Critical Pair Lemma

Given the definitions of critical pairs, we can formulate a critical pair lemma. However, we first need to clarify the notion of an 'instance' of a critical pair.

Lemma 8.13. If a term d rewrites in two ways such that $d \rightarrow^{D}_{p_1} d_1$ and $d \rightarrow^{D}_{p_2} d_2$, and w.l.o.g. $p_1 \leq p_2$ then there is a critical pair (c_1, c_2) such that there are $\pi \in DSubst_{\mathcal{S}}$ such that, $d_1|_{p_1} \simeq \pi c_1$ and $d_2|_{p_1} \simeq \pi c_2$, and in the case of upward strong rewriting $d_1|_{p_1} \trianglelefteq \pi c_1$ and $d_2|_{p_1} \trianglelefteq \pi c_2$.

Proof (8.13). We only consider the case of a term which term rewrites in two ways. Other cases are similar.

If a term d rewrites in two ways such that $d \rightarrow^T_{l_1 \rightarrow_{s_1} r_1, \sigma_1, p_1} d_1$ and $d \rightarrow^T_{l_2 \rightarrow_{s_2} r_2, \sigma_2, p_2} d_2$, then if $p_1.q = p_2$ for some path q then $\sigma_1 l_1|_q \simeq d|_{p_2} \simeq \sigma_2 l_2$ and as we can assume that variables in the two rules are disjoint, let $\sigma = \sigma_2 \circ \sigma_1$ and $\sigma l_1|_q \simeq d|_{p_2} \sigma l_2$ and σ_1, σ_2 are well-sorted substitutions, and thus these terms dynamically unify. Thus there is a most general unifier θ of $l_1|_q$ and l_2 such that $\theta \preceq^{\mathcal{W}} \sigma$, and there is a term-term critical pair between $l_1 \rightarrow_{s_1} r_1$ and $l_2 \rightarrow_{s_2} r_2$, (c_1, c_2) and there is a π such that $\sigma \trianglelefteq \pi \circ \theta$, and so $d_1|_{p_1} \trianglelefteq \pi c_1$ and $d_2|_{p_1} \trianglelefteq \pi c_2$.

Also in strong rewriting, $d_1|_{p_1} \trianglelefteq \sigma r_1 \trianglelefteq \pi c_1$ and $d_2|_{p_2} \trianglelefteq \sigma r_2 \trianglelefteq \pi c_2|_q$. We also know that from the canonical match $B_\Sigma(d_2|_{p_2}) \trianglelefteq \pi c_2$ so from Lemma 7.24 the lemma holds. □

The overlapping rewriting $d_1 \leftarrow d \rightarrow d_2$ is known as an *instance* of the critical pair (c_1, c_2).

Thus if there is a peak $d_1 \leftarrow d \rightarrow d_2$ then this is an instance of a critical pair $c_1 \leftarrow c \rightarrow c_2$, for some unified term c, under some substitution θ, where $\theta c_1 \simeq d_1$, and $\theta c_2 \simeq d_2$. For the peak (d_1, d_2) to be convergent a series of rewrites $d_1 \xrightarrow{*} d' \xleftarrow{*} d_2$ is required. In semantic matching this would be immediate if all critical pairs were trivial. However, this is not the case with weak matching, since if $c_1 \xrightarrow{*} c_1'$, it is not necessarily the case that $d_1 \rightarrow d_1'$.

We can however, give sufficient criteria for the existence of such convergent derivations if we restrict ourselves to *upward strong matching*, which we without confusion call strong matching, as downward strong matching is not used here. Thus we use strong matching and strong rewriting, which has the desired property.

To show that local confluence results from the elimination of critical pairs, it would be desirable to prove the following conjecture.

Theorem 8.14. Critical Pair Lemma.
Given a set of dynamic rewrite rules R, where all the critical pairs of R are trivial, then the strong rewriting relation $\rightarrow^D_R$ is locally confluent.

This theorem holds for *strict* rewriting; however, this requires many critical pairs to be considered and results in a completion algorithm similar to that of [HKK93], which is omitted here. Unfortunately, this is not proven for strong or weak rewriting. The proof of this lemma breaks down in the proof that the confluence of term-term critical pairs

necessarily gives the confluence of term-term overlaps. However, it can be shown that in these circumstances the overlap necessarily converges to terms equivalent modulo sorts. This gives rise to the following lemma.

Theorem 8.15. Critical Pair Lemma modulo Sorts.
Given a set of dynamic rewrite rules R, where all the critical pairs of R are trivial modulo sorts, then the strong rewriting relation is locally confluent modulo sorts.

Proof (8.15). This proof is performed by case analysis on overlaps, that is terms which can be rewritten in two different ways. Since there are three varieties of rewriting, we must include the cases of where these different kinds of rewriting interact. However, much of the proof of local confluence has been already carried out in Lemmas 7.63, 7.67, and 7.68, for confluence of sort-propagation in combination with other forms of rewriting; Lemma 7.70 for terms which sort-rewrite in two ways; Lemma 7.64 for all terms which rewrite at disjoint paths; Lemma 7.66 for variable overlaps; and Lemma 7.71 for the combination of term rewriting at a position above a sort-rewriting step. Further, Lemmas 7.72 and 7.73 demonstrate that the root overlap of propagation and term rewriting, and overlaps of sort-rewriting above term-rewriting are locally confluence modulo sorts, and can be discounted also. Thus there is only one case to consider: a non-variable overlap of term-rewriting steps. Assume a set of dynamic rewrite rules, R.

Term-Term Rewriting. Given a dynamic term d, there exist paths p, q, dynamic substitutions σ_1, σ_2 and dynamic rewrite rules $l_1 \rightarrow_{S_1} r_1$, $l_2 \rightarrow_{S_2} r_2 \in R$, such that $d \rightarrow^T_{l_1 \rightarrow_{S_1} r_1, \sigma_1, p} d_1$, and $d \rightarrow^T_{l_2 \rightarrow_{S_2} r_2, \sigma_2, q} d_2$. We need to show: $\exists d_1', d_2' \,.\, d_2 \xrightarrow{*}_R d_1' \simeq d_2' \,{}_R\!\xleftarrow{*} d_1$

Assume, without loss of generality, $p \leq q$, $q = p.p'$, and $l_1|_{p'} \in O(l_1)$ such that $l_1|_{p'} \notin Vars(l_1)$. In this case we have that

$$\begin{aligned} d_1 &= d[p \leftarrow \sigma_1 r_1] \downarrow_p S_1 \cup \mathcal{DS}(d|_p) \\ d_2 &= d[q \leftarrow \sigma_2 r_2] \downarrow_q S_2 \cup \mathcal{DS}(d|_q) \end{aligned}$$

Assuming the variable sets in the rules are disjoint, let $\sigma = \sigma_1 \circ \sigma_2$ and $\sigma l_1|_{p'} \simeq d_q \simeq \sigma l_2$. Thus $l_1|_{p'}$ and l_2 are unifiable and by Lemma 8.13 it is an instance of a Term-Term critical pair, $\theta r_1 \downarrow_\epsilon S_1 \cup \mathcal{DS}(\theta l_1) = \theta l_1[p' \leftarrow r_2] \downarrow_{p'} S_2 \cup \mathcal{DS}(\theta l_1|_{p'})$, where θ is a most general dynamic

unifier of $l_1|_{p'}$ and l_2. As this critical pair is, by assumption, trivial modulo sorts, then there are terms d_1', d_2' such that

$$\theta r_1 \downarrow_\epsilon S_1 \cup \mathcal{DS}(\theta l_1) \xrightarrow{*}{}^{D}_{R} d_1' \simeq d_2' \overset{D}{R}{}^{*}\!\!\leftarrow \theta l_1[p' \leftarrow r_2] \downarrow_{p'} S_2 \cup \mathcal{DS}(\theta l_1|_{p'})$$

As θ is most general there is a λ such that $\sigma \trianglelefteq \lambda \circ \theta$. Then by stability of rewriting,

$$\lambda\theta r_1 \downarrow_\epsilon S_1 \cup \mathcal{DS}(\theta l_1) \xrightarrow{*}{}^{D}_{R} \lambda d_1' \simeq \lambda d_2' \overset{D}{R}{}^{*}\!\!\leftarrow \lambda\theta l_1[p' \leftarrow r_2] \downarrow_{p'} S_2 \cup \mathcal{DS}(\theta l_2|_{p'})$$

and thus by Lemma 7.22 there are $d_1'' \trianglelefteq \lambda d_1'$ and $d_2'' \trianglelefteq \lambda d_2'$ such that

$$\sigma r_1 \downarrow_\epsilon S_1 \cup \mathcal{DS}(\theta l_1) \xrightarrow{*}{}^{D}_{R} d_1'' \simeq d_2'' \overset{D}{R}{}^{*}\!\!\leftarrow \sigma l_1[p' \leftarrow r_2] \downarrow_{p'} S_2 \cup \mathcal{DS}(\theta l_2|_{p'})$$

Further, since we are using strong rewriting, $\sigma l_1 \trianglerighteq B_\Sigma(d|_p)$ so again via Lemma 7.24 there are $d_1''' \trianglelefteq d_1''$, $d_2''' \trianglelefteq d_2''$ such that:

$$d|_p[\epsilon \leftarrow \sigma_1 r_1] \downarrow_\epsilon S_1 \cup \mathcal{DS}(d|_p) \xrightarrow{*}{}^{D}_{R} d_1''' \simeq d_2''' \overset{D}{R}{}^{*}\!\!\leftarrow d|_p[p' \leftarrow \sigma_2 r_2] \downarrow_{p'} S_2 \cup \mathcal{DS}(d|_{p.p'})$$

and by monotonicity of rewriting

$$\begin{array}{ccccc} d_1 = & d[p \leftarrow \sigma_1 r_1] \downarrow_p S_1 \cup \mathcal{DS}(d|_p) & & d[q \leftarrow \sigma_2 r_2] \downarrow_q S_2 \cup \mathcal{DS}(d|_q) & = d_2 \\ & \searrow^{*}_{D\,R} & & \overset{*}{D\,R}\swarrow & \\ & & d[p \leftarrow d_1'''] \simeq d[p \leftarrow d_2'''] & & \end{array}$$

and the theorem holds.

Hence in all cases the theorem holds and we are done. □

A crucial point to note is that when we use critical overlaps using upward strong rewriting, then the critical term of the critical pair involved is the Maximal Upward Strong Unified Term generated by the most general unifier, and is thus the *minimal term* in the subsumption ordering $\preceq^{\mathcal{W}}$. That is, any other term is an instance of this term, or is less than it (contains more sort information) in the approximation ordering $\trianglelefteq$.

8.2 Completion Modulo Sorts

The completion algorithm for dynamic order-sorted term-rewriting systems modulo sorts is similar to that for ordinary term-rewriting systems. It should not be forgotten that the operations used are the *dynamic* operations: (upward strong) dynamic matching; dynamic rewriting, term, sort and propagation rewriting all being used; and dynamic unification. A good deal of the calculation in the completion procedure is hidden in these operations. We obtain the set of rules given in Figure 8.1, which we call Basic Dynamic Completion (BDC). These rules are applied to a pair consisting of a set of dynamic equations E and a set of dynamic rules R. We also assume the existence of a well-founded sort-coherent termination ordering on dynamic terms, $>_d$.

(1) **Delete**

$$\frac{(E \cup \{d =_S d\}, R)}{(E, R)}$$

(2) **Reduce**

$$\frac{(E \cup \{d =_S e\}, R)}{(E \cup \{d =_{S \cup \{\mathcal{DS}(e')\}} e'\}, R)} \quad \text{if } e \rightarrow^D_R e'$$

(3) **Orient**

$$\frac{(E \cup \{d =_S e\}, R)}{(E, R \cup \{d \rightarrow_S e\})} \quad \text{if } d >_d e, \text{ or } d \simeq e \text{ and } d \neq e$$

(4) **Deduce**

$$\frac{(E, R)}{(E \cup \{d =_S e\}, R)} \quad \text{if } (d, e) \in CP_{\simeq}(R) \text{ and } S = \mathcal{DS}(d) \cup \mathcal{DS}(e).$$

Figure 8.1: BDC: Base Rules for Dynamic Completion.

In addition to the rules for basic dynamic completion, we can augment the procedure with additional rules which reduce rules as well as the equations. Note that $\overset{e}{\triangleright}$ is the well-founded strict encompassment ordering: $d_1 \overset{e}{\triangleright} d_2$ if $\exists \sigma$ and path $p \in O(d_1)$ such that $d_1|_p \simeq \sigma d_2$, and σ is not a variable renaming.

We call the set of rules BDC augmented with these additional rules the set of rules for Dynamic Completion (DC). In rule (5), Simplify Right, note that we can maintain the

(5) **Simplify Right**

$$\frac{(E, R \cup \{l \to_S r\})}{(E, R \cup \{l \to_{S \cup \{\mathcal{DS}(r')\}} r'\})} \quad \text{if } r \to_R^D r'$$

(6) **Simplify Left**

$$\frac{(E, R \cup \{l \to_S r\})}{(E \cup \{l' =_{S \cup \{\mathcal{DS}(l')\}} r\}, R)} \quad \text{if } (l \to^T_{g \to_T d} l' \text{ and } l \overset{e}{\triangleright} g) \text{ or } l \to^S_R l' \text{ or } l \to^\Sigma_R l'$$

Figure 8.2: Additional Rules for Interreduction.

equation as a rewrite rule, since we know that new right hand-side r' must be lower in the term-ordering than r. Note also that in Simplify Left we cannot rewrite a rule on the left by itself.

As a further refinement we note that if a rule in R is sort-rewritten on the left, then if the term ordering is an extension of a standard simplification ordering, as discussed in Section 7.3 above, the resulting equation will still be in the term-ordering and can immediately be reintroduced as a rule.

8.2.1 Proofs

We prove the correctness of these rules by considering all possible proofs, as in Bachmair [Bac87, Bac91] and other papers in the literature including [GKK90] and [HKK93].

Each inference rule transforms a pair of an Equation-set and Rule-set, (E_i, R_i) into a pair (E_{i+1}, R_{i+1}). We start from a pair $(E_0, \emptyset)$, where $E_0 = \Phi(E)$ and E are the non-dynamic equations of the specification. Assuming that the application of these rules do not result in failure, we generate the set of *persisting equations and rules*: $E^\infty = \bigcup_i \bigcap_{j>i} E_j$, $R^\infty = \bigcup_i \bigcap_{j>i} R_j$. It is our intention to show that, given a fair completion strategy, every proof of the equality of terms can be written as a dynamic rewrite proof. First we give a notion of fairness: every critical pair is considered for orientation into a rule, every critical pair is generated and each rule specialisation is added to the rule set.

Definition 8.16. The completion procedure is *fair* if:

1. $E^\infty = \emptyset$. That is every critical pair is oriented.
2. $CP(R^\infty)$ is generated.

We give a notion of a proof using a pair (E, R).

Definition 8.17. Given a set of dynamic equations E and a set of dynamic rewrite rules R, a *proof step* $d_1 \rightleftharpoons d_2$ is a pair of terms where either $d_1 \rightarrow_R^D d_2$, $d_1 \overset{D}{_R\leftarrow} d_2$, or $d_1 \simeq d_2$, or $d_1 \longleftrightarrow_E d_2$.

Definition 8.18. A *proof* of $d = d'$ using a set of dynamic equations E and a set of dynamic rules R is a sequence of proof steps written

$$d = d_1 \rightleftharpoons d_2 \rightleftharpoons \cdots \rightleftharpoons d_{n-1} \rightleftharpoons d_n = d'$$

A proof P is a *rewrite proof* if it is of the form

$$d = d_1 \rightarrow d_2 \rightarrow \cdots \rightarrow d_i \simeq d_{i+1} \leftarrow \cdots d_{n-1} \leftarrow d_n = d'$$

Given a proof P, and some dynamic substitution σ, then the proof σP is

$$\sigma d = \sigma d_1 \rightleftharpoons \sigma d_2 \rightleftharpoons \cdots \rightleftharpoons \sigma d_{n-1} \rightleftharpoons \sigma d_n = \sigma d'$$

and given a position p in some term u, the proof $u[p \leftarrow P]$ is given by:

$$u[p \leftarrow d] = u[p \leftarrow d_1] \rightleftharpoons u[p \leftarrow d_2] \rightleftharpoons \cdots \rightleftharpoons u[p \leftarrow d_{n-1}] \rightleftharpoons du[p \leftarrow_n] = u[p \leftarrow d']$$

By $P[Q]$ we denote the proof P with the subproof Q, ie $Q = d_i \rightleftharpoons \cdots \rightleftharpoons d_j$ for some $1 \leq i < j \leq n$.

From the soundness and completeness of rewriting, there is a proof for every equation $E \vdash_\Sigma t = t'$. We define a set of transformation rules which apply to proofs of this form, and show that the set of inference rules DC above simulates these transformations on proofs. We then show that every proof under the pair $(\emptyset, R^\infty)$ is a rewrite proof.

Definition 8.19. We give the following transformation rules $\mathcal{T}$ on proofs, to generate a transformation relation on proofs $P \longmapsto P'$. For brevity, some obvious symmetric counterparts of rules are omitted.

1. Deleting a Pair:

$$\forall d, d' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d \rightleftharpoons d') \longmapsto (d = d)$$
$$\text{if} \quad d \longleftrightarrow_{d=d,\iota,\epsilon} d'$$

2. Reducing a Pair:

$$\forall d, d' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d \rightleftharpoons d') \longmapsto (d \rightarrow d'' \rightleftharpoons d')$$
$$\text{if} \quad d \longleftrightarrow_{c_1=c_2,\sigma,p} d',$$
$$c_1 \rightarrow_{l \rightarrow_S r,\tau,q} c_1',$$
$$\text{and } d'' = d[p \leftarrow \sigma c_1']$$

$$\forall d, d' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d \rightleftharpoons d') \longmapsto (d \rightleftharpoons d'' \leftarrow d')$$
$$\text{if} \quad d \longleftrightarrow_{c_1=c_2,\sigma,p} d',$$
$$c_2 \rightarrow_{l \rightarrow_S r,\tau,q} c_2',$$
$$\text{and } d'' = d'[p \leftarrow \sigma c_2'].$$

3. Orienting a Pair:

$$\forall d, d' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d \rightleftharpoons d') \longmapsto (d \rightarrow^T d')$$
$$\text{if} \quad d \longleftrightarrow_{c_1=c_2,\sigma,p} d',$$
$$c_1 >_d c_2$$

$$\forall d, d' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d \rightleftharpoons d') \longmapsto (d \rightarrow^S d')$$
$$\text{if} \quad d \longleftrightarrow_{c_1=c_2,\sigma,p} d',$$
$$c_1 \simeq c_2 \text{ and } c_1 \neq c_2.$$

4. Adding a Critical Pair:

$$\forall d, d', d'' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d' \leftarrow d \rightarrow d'') \longmapsto (d' \longleftrightarrow d''' \simeq d'')$$
$$\text{if} \quad d \rightarrow^T_{l_1 \rightarrow_{S_1} r_1,\sigma_1,p} d'$$
$$d \rightarrow^T_{l_2 \rightarrow_{S_2} r_2,\sigma_2,q} d''$$
$$p \leq q$$
$$d'|_p \trianglelefteq \tau c_1$$
$$d''|_p \trianglelefteq \tau c_2$$
$$d''' = d'[p \leftarrow \tau c_2] \downarrow_p S \cup \mathcal{DS}(d'|_p)$$
$$c_1 =_S c_2 \in CP_{\simeq}(\{l_1 \rightarrow_{S_1} r_1, l_2 \rightarrow_{S_2} r_2\})$$

5. Rewriting by Term above Sort:

$$\forall d, d', d'' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d' \leftarrow d \rightarrow d'') \longmapsto (d' \leftarrow d'')$$
$$\text{if} \quad d \rightarrow^{T}_{l_1 \rightarrow_{s_1} r_1, \sigma_1, p} d'$$
$$d \rightarrow^{S}_{l_2 \rightarrow_{s_2} r_2, \sigma_2, q} d''$$
$$p \leq q$$

6. Rewriting by Sort above Term:

$$\forall d, d', d'' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d' \leftarrow d \rightarrow d'') \longmapsto (d' \rightarrow d''' \simeq d'')$$
$$\text{if} \quad d \rightarrow^{S}_{l_2 \rightarrow_{s_2} r_2, \sigma_2, q} d'$$
$$d \rightarrow^{T}_{l_1 \rightarrow_{s_1} r_1, \sigma_1, p} d''$$
$$q \leq p$$
$$d' \rightarrow^{T}_{l_1 \rightarrow_{s_1} r_1, \sigma_1, p} d'''$$

7. Reducing a Peak without Overlap:

$$\forall d, d', d'' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d' \leftarrow d \rightarrow d'') \longmapsto (d' \rightarrow d_1 \leftarrow d'')$$
$$\text{if} \quad d \rightarrow^{D}_{l_1 \rightarrow_{s_1} r_1, \sigma_1, p} d'$$
$$d \rightarrow^{D}_{l_2 \rightarrow_{s_2} r_2, \sigma_2, q} d''$$
$$p \bowtie q$$
$$d' \rightarrow^{D}_{l_2 \rightarrow_{s_2} r_2, \sigma_2, q} d_1$$
$$d'' \rightarrow^{D}_{l_1 \rightarrow_{s_1} r_1, \sigma_1, p} d_1$$

8. Reducing a Peak with a variable Overlap:

$$\forall d, d', d'' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d' \leftarrow d \rightarrow d'') \longmapsto (d' \xrightarrow{*} d_1 \xleftarrow{*} d'')$$
$$\text{if} \quad d \rightarrow^{T}_{l_1 \rightarrow_{s_1} r_1, \sigma_1, p} d'$$
$$d \rightarrow^{D}_{l_2 \rightarrow_{s_2} r_2, \sigma_2, q} d''$$
$$\exists p' \text{ such that } p.p' \leq q \text{ and } l_1|_{p'} \in \mathcal{X}$$
$$d' \xrightarrow{*} d_1$$
$$d'' \xrightarrow{*} d_1$$

$$\forall d, d', d'' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d' \leftarrow d \rightarrow d'') \longmapsto (d' \xrightarrow{*} d_1 \xleftarrow{*} d'')$$
$$\text{if} \quad d \rightarrow^{S}_{l_1 \rightarrow_{s_1} r_1, \sigma_1, p} d'$$
$$d \rightarrow^{D}_{l_2 \rightarrow_{s_2} r_2, \sigma_2, q} d''$$
$$\exists p' \text{ such that } p.p' \leq q \text{ and } r_1|_{p'} \in \mathcal{X}$$
$$d' \xrightarrow{*} d_1$$
$$d'' \xrightarrow{*} d_1$$

9. Reducing a Sort-Sort Overlap:

$$\forall d, d', d'' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d' \leftarrow d \rightarrow d'') \longmapsto (d' \rightarrow d_1 \leftarrow d'')$$

$$\text{if} \quad d \rightarrow^{S}_{l_1 \rightarrow_{S_1} r_1, \sigma_1, p} d'$$

$$d \rightarrow^{S}_{l_2 \rightarrow_{S_2} r_2, \sigma_2, q} d''$$

$$d' \rightarrow^{S}_{l_2 \rightarrow_{S_2} r_2, \sigma_2, q} d_1$$

$$d'' \rightarrow^{S}_{l_1 \rightarrow_{S_1} r_1, \sigma_1', p} d_1$$

10. Reducing a Σ Overlap:

$$\forall d, d', d'' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d' \leftarrow d \rightarrow d'') \longmapsto (d' \rightarrow d_1 \leftarrow d'')$$

$$\text{if} \quad d \rightarrow^{\Sigma}_{p} d'$$

$$d \rightarrow^{D}_{q} d''$$

$$p < q$$

$$d' \rightarrow^{D}_{q} d_1$$

$$d'' \rightarrow^{\Sigma}_{p} d_1$$

$$\forall d, d', d'' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d' \leftarrow d \rightarrow d'') \longmapsto (d' \rightarrow d_1 \leftarrow d'')$$

$$\text{if} \quad d \rightarrow^{\Sigma}_{p} d'$$

$$d \rightarrow^{\Sigma}_{q} d''$$

$$p = q$$

$$d' \rightarrow^{\Sigma}_{q} d_1$$

$$d'' \rightarrow^{\Sigma}_{p} d_1$$

$$\forall d, d', d'' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d' \leftarrow d \rightarrow d'') \longmapsto (d' \rightarrow d_1 \leftarrow d'')$$

$$\text{if} \quad d \rightarrow^{\Sigma}_{p} d'$$

$$d \rightarrow^{S}_{q} d''$$

$$p = q$$

$$d' \rightarrow^{S}_{q} d_1$$

$$d'' \rightarrow^{\Sigma}_{p} d_1$$

$$\forall d, d', d'' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d' \leftarrow d \rightarrow d'') \longmapsto (d' \rightarrow d'')$$

$$\text{if} \quad d \rightarrow^{\Sigma}_{p} d'$$

$$d \rightarrow^{T}_{q} d''$$

$$p > q$$

$$d' \rightarrow^{T}_{q} d''$$

$$\forall d, d', d'' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d' \leftarrow d \rightarrow d'') \longmapsto (d' \rightarrow d_1 \simeq d'')$$
$$\text{if} \quad d \rightarrow_p^{\Sigma} d'$$
$$d \rightarrow_q^{T} d''$$
$$p = q$$
$$d' \rightarrow_q^{T} d_1$$

11. Simplifying the Right-Hand Side of a Rule:

$$\forall d, d' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d \rightarrow d') \longmapsto (d \rightarrow d'' \leftarrow d')$$
$$\text{if} \quad d \rightarrow_{l_1 \rightarrow_{S_1} r_1, \sigma_1, p}^{D} d'$$
$$r_1 \rightarrow_{l_2 \rightarrow_{S_2} r_2, \sigma_2, q}^{T} r_1'$$
$$d'' = d'[p \leftarrow \sigma_1 r_1'] \downarrow_p \mathcal{DS}(d'|_p)$$

$$\forall d, d' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d \rightarrow d') \longmapsto (d \rightarrow d'' \leftarrow d')$$
$$\text{if} \quad d \rightarrow_{l_1 \rightarrow_{S_1} r_1, \sigma_1, p}^{D} d'$$
$$r_1 \rightarrow_{l_2 \rightarrow_{S_2} r_2, \sigma_2, q}^{S} r_1'$$
$$d'' = d' \downarrow_p \mathcal{DS}(r_1')$$

12. Term Simplifying the Left-Hand Side of a Rule:

$$\forall d, d' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d \rightarrow d') \longmapsto (d \rightarrow d'' \longleftarrow d')$$
$$\text{if} \quad d \rightarrow_{l_1 \rightarrow_{S_1} r_1, \sigma_1, p}^{D} d'$$
$$l_1 \rightarrow_{l_2 \rightarrow_{S_2} r_2, \sigma_2, q}^{T} l'$$
$$l_1 \overset{e}{\rhd} l_2$$
$$d'' = d[p \leftarrow \sigma_1 l'] \downarrow_p S_2 \cup \mathcal{DS}(d|_p)$$

Sort Simplifying the Left-Hand Side of a Rule:

$$\forall d, d' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d \rightarrow d') \longmapsto (d \rightarrow d'' \longleftarrow d')$$
$$\text{if} \quad d \rightarrow_{l_1 \rightarrow_{S_1} r_1, \sigma_1, p}^{D} d'$$
$$l_1 \rightarrow_{l_2 \rightarrow_{S_2} r_2, \sigma_2, q}^{S} l'$$
$$d'' = d \downarrow_p \mathcal{DS}(l')$$

13. Cliff reduction:

$$\forall d, d' : \mathcal{D}_{\mathcal{S}}(\mathcal{X}).(d'' \leftarrow d \simeq d') \longmapsto (d'' \overset{*}{\longrightarrow} d_1''' \simeq d_2''' \overset{*}{\longleftarrow} d')$$

Definition 8.20. If proof P is transformed into proof Q by repeated application of rules T, then we write $P \Rightarrow_T^+ Q$. If there is no proof Q such that proof $N \Rightarrow_T^+ Q$, then N is a normal form proof.

Definition 8.21. (Bachmair 91). A Proof transformation system $\mathcal{T}$ is said to *reflect* an inference system $\mathcal{I}$ on sets of equations and rules (E, R), if $(E, R) \vdash_{\mathcal{I}} (E', R')$ implies that every proof P in (E, R) can be transformed by $\mathcal{T}$ to a proof P' in (E', R').

We show that the above set of transformation rules on proofs reflects the set of rules DC.

Lemma 8.22. The set of transformation rules on proofs $\mathcal{T}$, reflects the set of rules DC.

Proof (8.22). Deduce is reflected by transformation rule 2, since if $(E, R) \vdash_{DC} (E', R')$ by Deduce, then $E' = E \cup c_1 = c_2$, where $(c_1, c_2) \in CP_{\simeq}(R)$, with critical term c, and $R' = R$. If P is a proof in (E, R), for every peak in P $d_1 \ {}^{T}\!\leftarrow d \rightarrow^{T} d_2$ such that there exist paths $p \leq q \in O(d)$, and substitution σ such that $d|p \trianglelefteq \sigma c$, and $d_1 = d[p \leftarrow \sigma c_1]\downarrow_p \mathcal{DS}(d|_p)$, and $d_2 \trianglelefteq d[p \leftarrow \sigma c_2]\downarrow_p \mathcal{DS}(d|_p)\downarrow_q \mathcal{DS}(d|_q) = d_2'$, and thus $d_1 \overset{*}{\longleftrightarrow}_{\{(c_1,c_2)\}} d_2' \simeq d_2$.

Similarly, Delete is reflected by transformation rule 1, Reduce by transformation rule 2, Orient by transformation rule 3, Simplify Right is reflected by transformation rule 11, and Simplify Left by rule 12. Proof transformations rule 13 transform proofs in (E, R) if R is (reachably) locally coherent. Other rules transform proofs within (E, R) by the confluence lemmas of Chapter 7. □

Lemma 8.23. Given a coherent well-founded dynamic term ordering $>_d$, the set of transformation rules $\mathcal{T}$ terminate.

Proof (8.23). Given a proof P, we define $M(P)$ to be the multiset of triples defined as follows. For each $d_i, d_{i+1} \in P$:

$$\text{if } d_i \overset{*}{\longleftrightarrow} d_{i+1} \text{ then } (\{d_i, d_{i+1}\}, \bot, \bot) \in M(P)$$

$$\text{if } d_i \rightarrow_{l \rightarrow_{sr}} d_{i+1} \text{ then } (\{d_i\}, l, r) \in M(P)$$

$$\text{if } d_i \ {}_{l \rightarrow_{sr}}\!\leftarrow d_{i+1} \text{ then } (\{d_{i+1}\}, l, r) \in M(P)$$

$$\text{if } d_i \simeq d_{i+1} \text{ then } (\{d_i, d_{i+1}\}, \bot, \bot) \in M(P)$$

$$\text{if } d_i = d_{i+1} \text{ then } (\{d_i\}, \bot, \bot) \in M(P)$$

where $\bot$ is an arbitrary element such that $\forall d \in \overline{\mathcal{D}}_{\Sigma}(\mathcal{X}) \cdot d >_d \bot$, and $\forall d \in \overline{\mathcal{D}}_{\Sigma}(\mathcal{X}) \cdot d \overset{e}{\rhd} \bot$. Clearly if $>_d$ is a well-founded then its multiset extension $>_{mul}$ is also well-founded. Let

$\succ_M$ be the lexicographic combination of $>_{mul}$, the strict encompassment ordering $\overset{e}{\rhd}$ on dynamic terms and $>_d$, and then let $\succ\!\!\!\succ_M$ be the multiset extension of $\succ_M$. We show that $\mathcal{T}$ is reducing in $\succ\!\!\!\succ_M$. Consider each rule in turn.

In each case, $\mathcal{T}$ transforms proof P to P'. For rules 1-10, we note the following reduction in the ordering $>_{mul}$:

1. $\{d, d\} >_{mul} \{d\}$
2. $\{d, d'\} >_{mul} \{d\}, \{d'', d'\}$ if $d'' <_d d$.
3. $\{d, d'\} >_{mul} \{d\}$, if $d'' <_d d$.
4. $\{d\}, \{d\} >_{mul} \{d'\}, \{d''', d''\}$, if $d', d'' <_d d$.
5. $\{d\}, \{d\} >_{mul} \{d''\}$, if $d', d'' <_d d$.
6. $\{d\}, \{d\} >_{mul} \{d'\}, \{d''', d''\}$, if $d', d'' <_d d$.
7. $\{d\}, \{d\} >_{mul} \{d'\}, \{d''\}$, if $d', d'' <_d d$.
8. $\{d\}, \{d\} >_{mul} \{d'\}, \{d''\}$, if $d', d'' <_d d$.
9. $\{d\}, \{d\} >_{mul} \{d'\}, \{d''\}$, if $d', d'' <_d d$.
10. $\{d\}, \{d\} >_{mul} \{d'\}, \{d''\}$, if $d', d'' <_d d$.

so in each of these cases $M(P) \succ\!\!\!\succ_M M(P')$. For rule 11, we note that:

11. $\{(\{d\}, l_1, r_1)\} \succ\!\!\!\succ_M \{(\{d\}, l_1, r_1'), (\{d'\}, l_2, r_2)\}$ as $d >_d d'$ *and* $r_1 >_d r_1'$

For Rule 12, we have:

12. $\{(\{d\}, l_1, r_1)\} \succ\!\!\!\succ_M \{(\{d\}, l_2, r_2), (\{d'', d'\}, \bot, \bot)\}$ as $d >_d d', d''$ and $l_1 \overset{e}{\rhd} l_2$.

For rule 13, since $>_d$ is a coherent ordering, $d, d' >_d d_1''', d_2'''$, so

13. $\{(\{d\}, l_1, r_1), (\{d, d'\}, \bot, \bot)\} \succ\!\!\!\succ_M \{(\{d''\}, l_2, r_2), \ldots, (\{d'', d'\}, \bot, \bot), \ldots, (\{d\}, l_n, r_n)\}$.

Thus $\mathcal{T}$ is terminating. □

Lemma 8.24. If proof $P \Rightarrow_{\mathcal{T}}^{+} N$ and N is a normal form proof, then N is a rewriting

proof, of the form $d_1 \xrightarrow{*} d_1' \simeq d_n' \xleftarrow{*} d_n$.

Proof (8.24). Assume that the following subproofs are in N.

1. If $d \xleftrightarrow{*} d' \in N$, then rule 1, 2, or 3 applies, so N is not in normal form.

2. If peak $d_1 \leftarrow d \rightarrow d_2 \in N$ then rule 4, 5, 6, 7, 8, 9 or 10 applies, so N is not in normal form.

3. If cliff $d_1 \leftarrow d \simeq d_2 \in N$, then rule 13 applies, so N is not in normal form.

Hence, there are no peaks, cliffs, or $=$ or $\xleftrightarrow{*}$ steps in N, so N is a rewrite proof. □

Theorem 8.25. If $(E_0, \emptyset)$ is transformed to $(\emptyset, R_\infty)$ by DC using a fair strategy, then any proof P in E_0 can be transformed into a rewrite proof N in R_∞.

Proof (8.25). By Lemma 8.22, $\mathcal{T}$ reflects DC, so proof P in E_0 is transformed into a proof Q in R_∞. $Q \Rightarrow_{\mathcal{T}}^{+} N$ where N is in normal form, and by Lemma 8.24, N is a rewrite proof. However, since there are no non-trivial critical pairs remaining in R_∞, and all rules have been oriented, there can be no steps using rules 1,2,3, or 4 in transforming Q to N, and hence N is a rewrite proof in R_∞. □

Thus we have a completion procedure modulo sorts. However, this completion is dependent on two conditions.

1. The dynamic rewriting relation is locally coherent.

2. Well-sorted unifiers can be identified.

These conditions are not in general met by dynamic rewriting systems. We go on to discuss ways that sufficient criteria can be given to meet these conditions. First, a simple example of how this completion algorithm can complete rule sets upon which standard completion fails.

Example 8.26. Given the specification of Example 3.5, we now have the dynamic rules:

Subsorts: $s' < s''$, $s < s''$

Rules: $h(x : s'')\bullet s'' \rightarrow_s f(x : s'')\bullet s$

$h(x : s'')\bullet s'' \rightarrow_{s'} g(x : s'')\bullet s'$

This set of rules can be seen to be coherent by inspection of terms. Completion generates the critical pair $f(x)\bullet s =_{\{s,s'\}} g(x)\bullet s'$, from the well-sorted unified term $h(x : s'')\bullet s''$, which given a well-founded term ordering, can be oriented into a rule (say left to right), resulting in the complete set of (interreduced) rules, under upward strong rewriting:

Rules: $h(x : s'')\bullet s'' \rightarrow_{\{s,s'\}} g(x : s'')\bullet\{s, s'\}$

$f(x : s'')\bullet s \rightarrow_{\{s,s'\}} g(x : s'')\bullet s'$

□

8.2.2 Proof Terms

Recall that the motivation of the completion procedure is to generate rewrite proofs of equations of the form $L(t) = L(t')$, where $t, t' \in T_\Sigma(\mathcal{X})$. The following set of terms represents a minimum set of terms which are needed to give proofs of such equations, which from the completeness of rewriting, are those terms used in a strict matching replacement proof, $L(t) \overset{*}{\longleftrightarrow}^{St} L(t')$, and also those proofs which can be derived from such proofs by transforming such proofs into rewrite proofs. It suffices to show the Church-Rosser property for this set of terms for such equations to be provable via rewrite proofs.

Definition 8.27. Given a specification $\mathcal{S} = (\Sigma, R)$, the set of *Proof Terms*, $\mathcal{P}_\mathcal{S}(\mathcal{X})$ is defined as follows.

The the set of *Kernel Proof Terms* is the set of terms:

$$Ker_\mathcal{S}(\mathcal{X}) = \{d \mid \exists t \in T_\Sigma(\mathcal{X}) \cdot L(t) \overset{*}{\longleftrightarrow}_R^{St} d\}$$

Then the proof terms, with respect to matching algorithm M are the set of terms:

$$\mathcal{P}_\mathcal{S}(\mathcal{X}) = Ker_\mathcal{S}(\mathcal{X}) \cup \{d \mid \exists d' \in Ker_\mathcal{S}(\mathcal{X}) \cdot d' \overset{*}{\longrightarrow}_R^M d\}$$

Clearly, the strict matching proof terms are exactly $Ker_{\mathcal{S}}(\mathcal{X})$. Unless otherwise stated, we shall assume the Upward Strong Proof terms. Note also that the set of proof terms is rewrite complete.

Lemma 8.28. If $R \vdash_{\Sigma} t_1 =_s t_2$, then there exists a proof in $Ker_{\mathcal{S}}(\mathcal{X})$ of $L(t_1) \overset{*}{\longleftrightarrow}_R L(t_2)$.

Proof (8.28). The completeness of rewriting demonstrates that there is a proof $L(t_1) \overset{*}{\underset{St}{\longleftrightarrow}}_R L(t_2)$. □

Thus we can restrict the Church-Rosser property, global and local confluence and global and local coherence to proof terms, as defined in Section 7.4.1. Also, we have the following property on the transformation system $\mathcal{T}$.

Lemma 8.29. The Transformation system $\mathcal{T}$ in Section 8.2 preserves proofs in $\mathcal{P}_{\mathcal{S}}(\mathcal{X})$.

Proof (8.29). Immediate from the forms of the rules. □

Note that the proof terms are not all the well-sorted terms, not even the well-sorted terms which are less than their syntactic-least form.

Example 8.30. Consider the specification:

Sorts:	$A\ B$
Subsorts:	$A \leq B$
Operators:	$a : \rightarrow A \qquad b : \rightarrow B$
	$f : B \rightarrow B$
Rules:	$b \bullet B \rightarrow_A a \bullet A$

Then $f(b \bullet A) \bullet B \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$, but $f(b \bullet A) \bullet B \notin \mathcal{P}_{\mathcal{S}}(\mathcal{X})$. □

8.3 Criteria for Completion

In the previous section, we established a completion procedure modulo sorts, which was dependent on a coherence rewriting relation, and well-sorted unification. In subsection 6.2.3, we discuss some criteria for identifying well-sorted unifiers. In this section we discuss further criteria, and also criteria for showing the coherence of dynamic rewriting.

8.3.1 Sort-Decreasing Rules

The restriction placed on the standard theory of rewriting is that rewrite rules should be sort-decreasing. We consider how this restriction is reflected in the Dynamic rewriting framework. The definition of sort-decreasing dynamic rules is as follows.

Definition 8.31. A dynamic rewrite rule $l \rightarrow_S r$ is *sort-decreasing* if:

1. $l \in \mathcal{L}_\Sigma(\mathcal{X})$ and $r \in \mathcal{L}_\Sigma(\mathcal{X})$.
2. $S = \mathcal{DS}(r) = \mathcal{LS}(r)$.
3. For all specialisations of variables ρ, $\rho l \xrightarrow{*}^{\Sigma} P_\Sigma(\rho l) = L(\rho l)$, $\rho r \xrightarrow{*}^{\Sigma} P_\Sigma(\rho r) = L(\rho r)$, and $\mathcal{DS}(P_\Sigma(\rho l)) \geq \mathcal{DS}(P_\Sigma(\rho r))$.

A set of dynamic rewrite rules R is *sort-decreasing* if all rules $l \rightarrow_S r \in R$ are sort-decreasing.

Note that there is no upward strong sort rewriting using sort-decreasing systems.

Lemma 8.32. If a set of dynamic rewrite rules R is *sort-decreasing* and $d \in \mathcal{D}_S(\mathcal{X})$, then there is no $d' \in \mathcal{D}_S(\mathcal{X})$ such that $d \rightarrow^S d'$ using upward strong (or strict) matching.

Proof (8.32). Let $l \rightarrow_S r$ be a rule in R. If $d \trianglerighteq L(d)$, then as $r \in \mathcal{L}_\Sigma(\mathcal{X})$, then r will not match on d. If $d \trianglelefteq L(d)$, then if $r \sqsubseteq^U d|_p$, then $\mathcal{DS}(r) \geq \mathcal{DS}(d|_p)$ and so $S \geq \mathcal{DS}(d|_p)$, and no sort rewrite will occur. □

If we restrict rewriting to proof terms, we can give the following result.

Lemma 8.33. If a set of dynamic rewrite rules R is *sort-decreasing*, then $\forall d \in \mathcal{P}_{\mathcal{S}}(\mathcal{X})$, $d \xrightarrow{*}^{\Sigma} L(d)$.

Proof (8.33). Consider terms in $Ker_{\mathcal{S}}(\mathcal{X})$. Clearly if $\exists t \in T_{\Sigma}(\mathcal{X})$ such that $d = L(t)$, then $d \in \mathcal{L}_{\Sigma}(\mathcal{X})$.

Assume that $d \xrightarrow{*}^{\Sigma} L(d)$, and $d \rightarrow_R^U d'$. Then as there are no sort rewrites, this must either be term or propagation rewriting. We prove the more general result.

If $d \rightarrow_{l \longrightarrow_{s} r,\sigma,p}^{U,T} d'$, then:

$$d' = d[p \leftarrow \sigma r] \downarrow_p s \cup \mathcal{DS}(d|_p)$$

As $d' \xrightarrow{*}^{\Sigma} L(d')$, then for all $x \in Dom(\sigma)$, $\sigma x \xrightarrow{*}^{\Sigma} L(\sigma x)$, and as $r \in \mathcal{L}_{\Sigma}(\mathcal{X})$ $\sigma r \xrightarrow{*}^{\Sigma} P_{\Sigma}(\sigma r) = L(\sigma r)$. Also with upward strong rewriting, $\mathcal{DS}(d|_p) \leq \mathcal{DS}(\sigma r) = s$, so:

$$\begin{aligned} d' &= d[p \leftarrow \sigma r] \downarrow_p \mathcal{DS}(d|_p) \\ &\xrightarrow{*}^{\Sigma} d[p \leftarrow L(\sigma r)] \downarrow_p \mathcal{DS}(d|_p) \end{aligned}$$

By matching $\mathcal{DS}(d|_p) \leq \mathcal{DS}(P_{\Sigma}(\sigma l))$, but as $d|_p \xrightarrow{*}^{\Sigma} L(d|_p)$ we must have $\mathcal{DS}(d|_p) \geq \mathcal{DS}(P_{\Sigma}(\sigma l))$ as both terms are in syntax normal forms. So, by sort decreasingness, $\mathcal{DS}(d|_p) \geq \mathcal{DS}(P_{\Sigma}(\sigma r))$, so

$$\begin{aligned} d' &\xrightarrow{*}^{\Sigma} d[p \leftarrow L(\sigma r)] \\ &\xrightarrow{*}^{\Sigma} P_{\Sigma}(d[p \leftarrow L(\sigma r)]) \\ &= L(d[p \leftarrow \sigma r]) \end{aligned}$$

For $d \in Ker_{\mathcal{S}}(\mathcal{X})$, Assume that $d \xrightarrow{*}^{\Sigma} L(d)$, and $d \leftarrow_{l \longrightarrow_{s} r,\sigma,p}^{ST,T} d'$. Then:

$$d' = d[p \leftarrow \sigma l] \downarrow_p \mathcal{DS}(d|_p)$$

but as strict rewriting is used, $\mathcal{DS}(d|_p) = \mathcal{DS}(l)$, so:

$$\begin{aligned} d' &= d[p \leftarrow \sigma l] \\ &\xrightarrow{*}^{\Sigma} d[p \leftarrow P_{\Sigma}(\sigma l)] \\ &\xrightarrow{*}^{\Sigma} d[p \leftarrow L(\sigma l)] \end{aligned}$$

as $d' \xrightarrow{*}^{\Sigma} L(d')$, which also gives, for any path $q \not/ p \in O(d')$ and $p \not\leq q$, $d'|_q \xrightarrow{*}^{\Sigma} L(d'|_q)$ and as least sorts are preserved $d|_q \xrightarrow{*}^{\Sigma} L(d|_q)$, so:

$$\begin{aligned} d' &\xrightarrow{*}^{\Sigma} P_\Sigma(d[p \leftarrow \sigma l]) \\ &= L(d[p \leftarrow \sigma l]) \end{aligned}$$

and we are done.

For terms in $\mathcal{P}_\mathcal{S}(\mathcal{X})$, the result holds from either, $d \in Ker_\mathcal{S}(\mathcal{X})$. which we have shown, or else we can assume that $d \in \mathcal{L}_\Sigma(\mathcal{X})$ and by the more general result given above. $d \rightarrow^U, Td' \xrightarrow{*}^{\Sigma} L(d)$. □

Theorem 8.34. If a set of dynamic rewrite rules R is *sort-decreasing*, then it is locally coherent on proof terms.

Proof (8.34). By the previous lemma $\forall d_1, d_2 \in \mathcal{P}_\mathcal{S}(\mathcal{X})$, such that $d_1 \simeq d_2$, $d \xrightarrow{*}^{\Sigma} L(d) \xleftarrow{*}^{\Sigma} d_2$, so R is resolvent confluent on $\mathcal{P}_\mathcal{S}(\mathcal{X})$. By Lemma 7.60, R is coherent on $\mathcal{P}_\mathcal{S}(\mathcal{X})$. □

Similarly, we only need generate critical terms which are proof terms. That is we only need to generate unifiers through proof terms. We can give the following fact about critical terms of proof terms.

Lemma 8.35. If R is a set of sort-decreasing rewrite rules, and if $d_1, d, d_2 \in \mathcal{P}_\mathcal{S}(\mathcal{X})$, such that $d_1 \leftarrow d \rightarrow d_2$, is a critical overlap, the critical term of the peak, $c \trianglerighteq L(c)$.

Proof (8.35). Assume a term-term peak, $d_1 \; {}_{l_1 \rightarrow_{S_1} r_1, \sigma_1} \leftarrow d \rightarrow_{l_2 \rightarrow_{S_2} r_2, \sigma_2} d_2$, formed by sort-decreasing rules. Then we know from Lemma 8.33 that $d \trianglerighteq L(d)$, so σ_1, σ_2 are substitutions which approximate to syntactic least terms, l_1, l_2 are syntactic least terms and so the critical peak term $c = \sigma l_1[p \leftarrow \sigma l_1|_p \wedge \sigma l_2] \trianglerighteq L(c)$. □

Hence the critical term approximates the syntactic least form. Thus for *peak terms* we satisfy the conditions of Lemma 6.55, and we can use the modified unification algorithm of Figure 6.6 to generate well-sorted unifiers, and well-sorted peak terms.

8.3.2 Sort-Convergent Rules

As noted in Section 3.5, in [CH91b] the notion of sort convergence is shown to be sufficient to ensure that the confluence of syntactically well-sorted term rewriting satisfies the confluence of $\mathcal{W}$-rewriting, or all well-sorted terms rewrites. We can reproduce these results in a dynamic framework.

Definition 8.36. A dynamic term-rewriting system R is *sort convergent* if for any $l \rightarrow_S r \in R$ and any Σ-substitution σ, there is a L-term u such that $\sigma(l) \xrightarrow{*}{}^{L} u \; {}^{L}\xleftarrow{*} \sigma(r)$.

This condition is decidable by considering all specialisations of variable in the rules.

Lemma 8.37. If R is a terminating, sort convergent and confluent modulo sorts set of dynamic rewrite rules, and if $u \in \mathcal{L}_\Sigma(\mathcal{X})$, then $u \xrightarrow{*}{}^{D} w \in \mathcal{L}_\Sigma(\mathcal{X})$ is a normal form if and only if $u \xrightarrow{*}{}^{L} w \in \mathcal{L}_\Sigma(\mathcal{X})$ is a normal form.

Proof (8.37). Only if direction. By induction on termination ordering $>_d$. If $d \in \mathcal{L}_\Sigma(\mathcal{X})$, and $d \rightarrow^{D}_{l \rightarrow_S r, \sigma, p} d'$, then as R is sort convergent $\exists u \in \mathcal{L}_\Sigma(\mathcal{X}) \cdot \sigma(l) \xrightarrow{*}{}^{L} u \; {}^{L}\xleftarrow{*} \sigma(r)$, and hence $d \xrightarrow{*}{}^{L} d[p \leftarrow u]$ and $d[p \leftarrow u] \in \mathcal{L}_\Sigma(\mathcal{X})$. By the induction hypothesis, $d[p \leftarrow u] \xrightarrow{*}{}^{L} d[p \leftarrow u] \downarrow_R \in \mathcal{L}_\Sigma(\mathcal{X})$.

If direction. Clearly, $d \xrightarrow{*}{}^{L} d'$ implies $d \xrightarrow{*}{}^{D} d'$. We prove by contradiction. Assume that d' is a normal form for $\rightarrow^L$, but not for $\rightarrow^D$, then there is some d'' such that $d' \rightarrow^{D}_{l \rightarrow_S r, \sigma, p} d''$. R is sort convergent so $\exists u \in \mathcal{L}_\Sigma(\mathcal{X}) \cdot \sigma(l) \xrightarrow{*}{}^{L} u \; {}^{L}\xleftarrow{*} \sigma(r)$, and hence $d' \xrightarrow{*}{}^{L} d'[p \leftarrow u]$ and $d'[p \leftarrow u] \in \mathcal{L}_\Sigma(\mathcal{X})$, which contradicts the d' being a normal form for $\rightarrow^L$. □

Theorem 8.38. Given a set of terminating and sort convergent dynamic rewrite rules R, $\rightarrow^D$ is confluent modulo sorts if and only $\rightarrow^L_R$ is confluent modulo sorts.

Proof (8.38). A corollary of the previous lemma. □

As a consequence of this theorem, to establish confluence modulo sorts, and hence the Church-Rosser property modulo sorts, for proofs of the form $t_1, t_2 \in T_\Sigma(\mathcal{X})$ for sort-

convergent sets of rules, we need only establish confluence modulo sorts, and hence the Church-Rosser property modulo sorts, for the relation $\rightarrow^L$. Thus all proofs are using $\rightarrow^L$, and by Lemma 7.62, this relation is coherent. Also, we need only generate critical pairs from unifying Σ-substitutions, and hence can use the Standard Abstract rule in generating well-sorted unifiers.

8.4 Towards a Conditional Method

The above completion procedure is contingent upon the dynamic unifier used to generate critical pairs being well-sorted. However, we have seen that in general the well-sortedness of unifiers is not decidable, and the unitary dynamic unification algorithm given in Chapter 6 in general generates ill-sorted unifiers, and ill-sorted unified terms. In this section we propose a modified version of dynamic rewriting given in Chapter 7, which allows ill-sorted unified terms.

The following example demonstrates the problems which can occur with ill-sorted unifiers.

Example 8.39. Consider the following specification.

Sorts:	$A\ B$	
Operators:	$a : \rightarrow A$	$f : A \rightarrow A$
	$b : \rightarrow B$	$f : B \rightarrow B$
Rules:	$f(x : A)\bullet A \rightarrow_A a\bullet A$	
	$f(y : A)\bullet B \rightarrow_B b\bullet B$	

Dynamically unifying $f(x : A)\bullet A \stackrel{?}{=} f(y : B)\bullet B$ generates the dynamic critical pair $(a\bullet\{A,B\}, b\bullet\{A,B\})$. However, $a = b$ is not a valid equational consequence of the specification, as the meet sort $A \wedge B$ is uninhabited. □

The problem that occurs in Example 8.39 can be captured by noting that the *maximal upward strong unified term* of the overlap, $f(z : \{A,B\})\bullet\{A,B\}$, is ill-formed and has no well-formed instances. Another example is as follows.

Example 8.40. Extend the specification given in Example 6.33 with the following operators and rules.

Operators:	$f : Nat \rightarrow Nat$
	$N : \rightarrow Nat$
Rules:	$f(y_{pos})\bullet Nat \rightarrow_{Pos} y_{pos}$
	$f(mod_2(z : Nat)\bullet Nat)\bullet Nat \rightarrow_{Nat} N\bullet Nat$

This is not confluent since for any $n \in \mathbb{N}$:

$$\begin{array}{ccc} & f(mod_2(s^{2n+1}(0\bullet Nat)\bullet^{2n+1} Pos)\bullet Pos)\bullet Nat & \\ \swarrow & & \searrow \\ mod_2(s^{2n+1}(0\bullet Nat)\bullet^{2n+1} Pos)\bullet Pos & & N\bullet Nat \end{array}$$

forms a well-sorted critical peak. However, the most-general dynamic unifier gives an ill-sorted critical peak:

$$\begin{array}{ccc} & f(mod_2(z : Nat)\bullet Pos)\bullet Nat & \\ \swarrow & & \searrow \\ mod_2(z : Nat)\bullet Pos & & N\bullet Nat \end{array}$$

This peak, equates a well-sorted term $N\bullet Nat$ with an ill-sorted term. □

Thus this critical peak is ill-sorted, and we should not allow this as a valid dynamic equation to be added to the current axioms. However, in general we cannot tell whether a unified term, or any of its instances, are well-sorted or not.

Our solution is to keep the critical peak term in the resulting equality. Thus we generate *critical triples* rather than critical pairs. When these equalities are oriented into rewrite rules, the rule can only apply if the instance of the critical peak term is well-sorted.

To formalise these notions, first we define a constrained equality.

Definition 8.41. Dynamic Constrained Equation.

A Dynamic Constrained Equation is defined to be a quintuple, written $\forall X.C : l =_S r$, where $X \in \mathcal{X}$, $C \subseteq \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, $l, r \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, and $S \in S_\Sigma$ and $X \supseteq Vars(C) \cup Vars(\{l, r\})$ and

$S \supseteq \mathcal{DS}(l) \cup \mathcal{DS}(r)$.

We define Dynamic Constrained Rewrite Rules in a similar fashion, and use them instead of normal rewrite rules. The check that we have to make when we apply a dynamic rewrite rule is that all the constraints are well-sorted in the instance the rule is applied. Thus dynamic rewriting with constrained rewrite rules becomes similar to conditional rewriting. For details on conventional conditional rewriting see for example [Kap87, Gan91a]. We only define constrained rewriting for upward strong matching.

Definition 8.42. Given $d \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$ and a constrained dynamic rule $\forall X.C : l \rightarrow_S r$:

1. *d constrained term rewrites* to d' if $\phi(l) \neq \phi(r)$, if there is a dynamic substitution $\sigma : Y \rightarrow \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, and path $p \in O(d)$, such that $l \sqsubseteq^U_\sigma d|_p$, and $\forall c \in C \cdot \sigma c \in \mathcal{D}_\mathcal{S}(\mathcal{X})$, and $d' = d[p \leftarrow \sigma r]\downarrow_p S \cup \mathcal{DS}(d|_p)$.

2. *d constrained sort rewrites* to d' if there is a dynamic substitution $\sigma : Y \rightarrow \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, $p \in O(d)$ such that $r \sqsubseteq^U_\sigma d|_p$, $\forall c \in C \cdot \sigma c \in \mathcal{D}_\mathcal{S}(\mathcal{X})$, and $\mathcal{DS}(d) \not\leq S$, and $d' = d\downarrow_p S$.

Sort propagation rewriting is as before. The union of these three types of rewriting is known as *Constrained Dynamic Rewriting*, written $d \rightarrow^C d'$.

Clearly, if for all rules in a set of rules R, the set of constraints is empty, then we revert to unconstrained dynamic rewriting. We define Critical Triples as follows.

Definition 8.43. Critical Triples.
If constrained dynamic rules $\forall X.C_1 : l \rightarrow_{S_1} r$, and $\forall Y.C_2 : g \rightarrow_{S_2} d$, regarded as unconstrained rules, generate a critical pair: (c_1, c_2), with most general dynamic unifier θ, and critical term c, then the *Critical Triple* is defined to be $\forall X \cup Y \cdot \theta C_1 \cup \theta C_2 \cup \{c\} : (c_1, c_2)$. The constraint c and all subterms of c are said to be *derived* from rules $\forall X.C_1 : l \rightarrow_{S_1} r$, and $\forall Y.C_2 : g \rightarrow_{S_2} d$.

The denotation of constrained dynamic rules is similar to that of conditional rewrite rules. In this case an instance of a constrained dynamic equation $\forall X.C : l =_S r$ under substitution σ holds as a dynamic equation $\sigma l =_S \sigma r$ if and only if $\sigma C \subseteq \mathcal{D}_\mathcal{S}(\mathcal{X})$, that is, it is well-sorted.

This interpretation can be seen to be analogous to the situation of empty sorts, sketched in Section 2.4. Given an equation $\forall X.l = r$, this holds in all models where the sorts of the variables in X are inhabited, that there is a well-sorted ground term of the form $t : s$ for each $x_s \in X$, so in dynamic terms $t \bullet s \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$. The concept of constrained equations extends this concept. Given a constrained equation $\forall X.C : l = r$, it (or an instance) is valid if the set of well-sorted terms which are instances of the terms in C is non-empty.

Example 8.44. In Example 8.40, the ill-sorted peak can generate the constrained equation:

$$f(mod_2(z_{Nat}) \bullet Pos) \bullet Nat : mod_2(z_{Nat} \bullet Pos) \;=\; N \bullet Nat$$

and for each substitution, $\theta_n = \{z : Nat \mapsto s^{2n+1}(0 \bullet Nat) \bullet^{2n+1} Pos\}$, the peak term:

$$\theta_n f(mod_2(z_{Nat}) \bullet Pos) \bullet Nat \;=\; f(mod_2(s^{2n+1}(0 \bullet Nat) \bullet^{2n+1} Pos) \bullet Pos) \bullet Nat$$

is well-sorted, and thus so is the corresponding instance of the equation. □

We can use constrained dynamic equation and rules to reproduce the results of Chapter 7, ignoring the condition. Birkhoff's Theorem trivially holds for such rules as in its proof we assume only well-sorted replacements are performed. We can also trivially extend the notion of completion modulo sorts to constrained dynamic rules.

Reducing Constrained Dynamic Equations

By using constrained dynamic rules, we have shifted the problem of deciding the well-sortedness of dynamic unification to that of deciding the well-sortedness of terms, which is equally undecidable. Criteria for deciding the well-sortedness of terms is the subject of further research. However, we can reduce and possibly eliminate constraints with a (non-exclusive) set of reduction rules on constrained equations which use the sort information already deduced. The reduction rules in Figure 8.3 apply to constrained equations and rules.

Lemma 8.45. If a constrained dynamic equation of the form $\forall X.C : l =_{\mathcal{S}} r$ is transformed to $\forall X.C' : l =_{\mathcal{S}} r$ by the rules of Figure 8.3, then $C \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$ if or only if $C' \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$.

(1) Approximate.
$$\frac{\forall X.\{c_1, c_2\} \cup C : l =_S r}{\forall X.\{c_1\} \cup C : l =_S r} \text{ if } c_2 \trianglerighteq c_1$$

(2) Merge.
$$\frac{\forall X.\{c_1, c_2\} \cup C : l =_S r}{\forall X.\{c_1 \wedge c_2\} \cup C : l =_S r} \text{ if } \phi(c_1) = \phi(c_2)$$

(3) Decompose.
$$\frac{\forall X.\{f(c_1, \ldots, c_n)\bullet T\} \cup C : l =_S r}{\forall X.\{c_1, \ldots, c_n\} \cup C : l =_S r} \text{ if } T \geq \mathcal{LS}(f(c_1, \ldots, c_n)\bullet T)$$

(4) Weaken.
$$\frac{\forall X.\{f(c_1, \ldots, c_n)\bullet T \cup \{s\}\} \cup C : l =_S r}{\forall X.\{f(c_1, \ldots, c_n)\bullet T\} \cup C : l =_S r} \text{ if } s \geq \mathcal{LS}(f(c_1, \ldots, c_n)\bullet T)$$

(5) Variable.
$$\frac{\forall X.\{x : s\} \cup C : l =_S r}{\forall X.C : l =_S r} \text{ if } x \in \mathcal{X}_s, \text{ and } s \text{ inhabited.}$$

Figure 8.3: Rules for Reducing Constraints.

Proof (8.45). Consider each rule in turn.

Approximate. If $c_1 \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$ then every d such that $c_1 \trianglelefteq d$, $d \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$.

Merge. $c_1, c_2 \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$ if and only if $c_1 \wedge c_2 \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$.

Decompose. Clearly, $L(f(c_1, \ldots, c_n)) \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$, so if $c_1, \ldots, c_n \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$, then $f(c_1, \ldots, c_n)\bullet\mathcal{LS}(f(c_1, \ldots, c_n)) \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$, so $f(c_1, \ldots, c_n)\bullet T \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$.

Weaken. A similar argument as for Decompose.

Variable. Trivial. Note that s can be an intersection sort. □

By repeated application of the Decompose rule, we can remove conditions c such that $c \trianglerighteq L(c)$.

The above reduction rules have been of a syntactic nature. We can use the (constrained) well-sortedness of the rules generating the constraint to reduce it further. We use the

concept of a constraint being derived from a rule. The following applies to constraints arising from term-term critical triples; similar lemmas and rules can be given for other critical triples.

Lemma 8.46. If $\forall X.\{f(c_1,\ldots,c_n)\bullet T\} \cup C : d =_S e$ is a constrained equation such that $c_1,\ldots,c_n \in \mathcal{DS}(\mathcal{X})$, and $f(c_1,\ldots,c_n)\bullet T$ is derived from $\forall X.C_1 : l \rightarrow_{S_1} r$, and $\exists p \in O(l)$ such that $l|_p \notin \mathcal{X}$ and $\sigma \in \overline{DSubst}_\Sigma \cdot \sigma l|_p \trianglerighteq f(c_1,\ldots,c_n)\bullet T$, and $\mathcal{DS}(l|_p) \cup S_1 \leq T$, then for any $\theta \in \overline{DSubst}_\Sigma$ such that $\theta C_1 \subseteq \mathcal{DS}(\mathcal{X})$, $\theta f(c_1,\ldots,c_n)\bullet T \in \mathcal{DS}(\mathcal{X})$.

Proof (8.46). As $f(c_1,\ldots,c_n)\bullet T$ is derived from $\forall X.C_1 : l \rightarrow_{S_1} r$ from a term-term critical triple, then there is a critical peak term c such that for some $q \in O(l)$ and $\sigma \in \overline{DSubst}_\Sigma$, $\sigma l_q \trianglerighteq c$, and there is a path $q' \in O(c)$ such that $c|_{q'} \trianglelefteq f(c_1,\ldots,c_n)\bullet T$. Let $p = q.q'$, and assume that $l|_p \notin \mathcal{X}$, so $\sigma l|_p \simeq f(c_1,\ldots,c_n)\bullet T$.

If $\theta C_1 \in \mathcal{DS}(\mathcal{X})$, then $\theta l|_p \in \mathcal{DS}(\mathcal{X})$, so $T \cup S_1 \in \mathcal{SS}(\theta l|_p)$, by definition of dynamic equation, so as $l|_p \notin \mathcal{X}$, $T \cup S_1 \geq \mathcal{SS}(\theta f(c_1,\ldots,c_n)\bullet T)$ so if $c_1,\ldots,c_n \in \mathcal{DS}(\mathcal{X})$, then $\theta f(c_1,\ldots,c_n)\bullet T \in \mathcal{DS}(\mathcal{X})$. □

We can use this lemma to justify the rule in Figure 8.4.

(6) Derived.

$$\frac{\forall X.\{f(c_1,\ldots,c_n)\bullet T\} \cup C : d =_S e}{\forall X.\{c_1,\ldots,c_n\} \cup C : d =_S e}$$

if $f(c_1,\ldots,c_n)\bullet T$ is derived from $\forall X.C_1 : l \rightarrow_{S_1} r$ and $\exists p \in O(l) \cdot l|_p \preceq f(c_1,\ldots,c_n)\bullet T$, $l|_p \notin \mathcal{X}$ and $\mathcal{DS}(l|_p) \leq T$

Figure 8.4: Rule for Reducing a Derived Constraint of Critical Triple.

In this case σC_1, or terms which have been reduced from C_1, is contained within C. Thus by the lemma, the term $f(c_1,\ldots,c_n)\bullet T$ is well-sorted in all contexts that C is, and can thus be eliminated.

As the well-sortedness of dynamic terms is dependent upon the equational theory, the well-sortedness of a term can be determined by rewriting. However, we cannot naively rewrite a term to see if it rewrites to a well-sorted term; as we have seen, ill-sorted terms can rewrite to a well-sorted term. Thus, we have to take an equivalent well-sorted term and see if a

term rewrites to a term of the appropriate sort.

Lemma 8.47. Given a set of Constrained dynamic rewrite rules R, and $d \in \overline{\mathcal{D}}_\Sigma(\mathcal{X})$, such that $d = f(c_1, \ldots, c_n)\bullet T$ and $c_1, \ldots, c_n \in \mathcal{D}_\mathcal{S}(\mathcal{X})$, if $L(d) \xrightarrow{*}{}^{C} d'$ and $\mathcal{DS}(d') \leq T$, then $d \in \mathcal{D}_\mathcal{S}(\mathcal{X})$.

Proof (8.47). Clearly, $L(d) \in \mathcal{D}_\mathcal{S}(\mathcal{X})$. By Lemma 7.17 extended to constrained rewriting, that if $L(d) \rightarrow^C d'$ then $d' \in \mathcal{D}_\mathcal{S}(\mathcal{X})$. Hence, by soundness of dynamic rewriting, $L(d)\bullet T \trianglerighteq L(d)\bullet LS(d') \in \mathcal{D}_\mathcal{S}(\mathcal{X})$. As $c_1, \ldots, c_n \in \mathcal{D}_\mathcal{S}(\mathcal{X})$, so $f(c_1, \ldots, c_n)\bullet T \in \mathcal{D}_\mathcal{S}(\mathcal{X})$. □

This lemma can be used in a constraint reduction rule given in Figure 8.5. Alternatively, instances of constraints can be tested for well-sortedness in this way, during rewriting. We give an example in the next chapter.

(7) Rewrite.

$$\frac{\forall X.\{f(c_1, \ldots, c_n)\bullet T\} \cup C : l =_\mathcal{S} r}{\forall X.\{c_1, \ldots, c_n\} \cup C : l =_\mathcal{S} r} \text{ if } L(f(c_1, \ldots, c_n)\bullet T) \xrightarrow{*}{}^{C} d \text{ and } \mathcal{DS}(d) \leq T$$

Figure 8.5: Rule for Reducing Constraint via Rewriting.

Conditional term rewriting introduces the notion of a *reductive conditional rewrite rule* [Kap87], which is a conditional rewrite rule where all terms in conditions are less than the left-hand term of the rule. This condition ensures termination of the rewrite relation and also that local confluence is equal to global confluence. It can be included within constrained dynamic rewriting, so the Orient rule in the completion algorithm is replaced by the stronger rule in Figure 8.6.

(3a) Constrained Orient.

$$\frac{\forall X.C : d =_\mathcal{S} e}{\forall X.C : d \rightarrow_\mathcal{S} e} \text{ if } d >_d e, \text{ or } d \simeq e \text{ and } d \neq e, \text{ and also } \forall c \in C.l >_t c$$

Figure 8.6: Rule for Orienting Critical Triples.

This condition may seem hard to establish, as by the rewriting relation, if $c_1 \leftarrow c \rightarrow c_2$ is a critical peak, then $c >_d c_1, c_2$. However, the constraint reduction rules 1-7 above may reduce the constraint sufficiently to establish this condition.

The constrained dynamic rewriting method offers a different path of analysis to the standard dynamic rewriting, pushing the boundaries of dynamic order-sorted equational rewriting, and allowing us to make further connections with work in other areas. Note for example, that the constraints are similar to the definedness condition in partial algebras [BW82]. There are also similarities to the work on membership constraints of for example [Toy87] and [Com90, Com92]. This method is a candidate for further research.

8.5 Comments on Dynamic Rewriting

The formulation of dynamic rewriting gives rise to a choice of rewriting relations, depending on which matching algorithm is used. Which rewriting relation is most appropriate in any given situation, is dependent on the use of the rewriting. If rewriting only is required, to simplify terms for example, without regard to the completeness of the rewriting. then weak rewriting provides the largest relation and may result in shorter, more efficient rewriting sequences. However, as we have seen, there is no completion established for this relation. Upward strong rewriting provides completion modulo sorts, if the coherence and well-sorted dynamic unification conditions can be met, either implicitly in the completion algorithm, or explicitly in constrained rules. Strict rewriting is the most inefficient rewriting relation, but there is a completion procedure associated with it, which used strict matching and unification. This completion algorithm also requires many more critical pairs than completion modulo sorts using upward strong matching. Currently, no special properties have been identified for downward strong rewriting, and it is included for completeness.

Chapter 9

Review and Conclusions

In the last chapter of this thesis, we give a roundup of the results presented in the previous chapters. We first summarise the contents of each chapter, emphasising the new results. Some examples are then revisited illustrating the use of the dynamic order-sorted logic and rewriting theory upon some well-known problem cases are then given to support these results. Finally, we discuss the conclusions of this thesis and give some possible directions for future research in both theory and in how the dynamic rewriting theory could be realised within an implementation, extending the existing MERILL system.

9.1 A Summary of the work of this Thesis

We review the work of this thesis and give a summary of the major results.

The context of specification is presented in Chapter 1 where the motivations for chosing to study order-sorted algebraic specification are presented. The area of order-sorted specification has been a well studied area in the past ten years, a body of work summarised in the first part of Chapter 2. Nevertheless, there are few extant implementations of this work; the MERILL system developed as part of this thesis, described in the sec-

ond part of Chapter 2, is the first implementation of order-sorted completion modulo the associative-commutative axioms. This system thus handles a wider range of problems than its predecessor ERIL [Dic85, Dic87].

However, as demonstrated in Chapter 3 there is a wide class of problems which are not handled by MERILL, not merely due to implementation restrictions within MERILL, but because of deep theoretical problems within the standard order-sorted theories themselves, whether using the non-overloaded semantics of [SNGM88], or the overloaded semantics of [GM89]. The causes of these problems are discussed at length in the same chapter, and the existing literature is reviewed in some detail to consider approaches to overcoming these difficulties. The conclusions of this chapter are that although there are many interesting attempts to accommodate these difficulties, which in themselves reveal interesting problems, none are entirely satisfactory.

As a consequence, it was decided that a reworking of order-sorted theory was required, and in Chapter 4 a new approach to order-sorted algebras is presented. This captures the intuition of the specifier more closely as it allows unsorted terms as well as sorted, by constructing *two-tier dynamic algebras.* A semantics of sorting which takes into account equality is introduced via the notion of *sort judgement*, and a new dynamic equational logic is defined. This logic is notable for its simplicity and power; it is shown to be sound and complete with respect to the two-tier semantics. This dynamic algebra and logic capture precisely what is intuitively meant by order-sorted specification, a claim supported by comparing it with Equational Typed Logic [MSS90]. The dynamic logic is the specialisation of ETL to order-sorted logic rather than the standard semantics.

To produce a computational analogue to the dynamic semantics, the notion of terms is modified to record explicitly the sorts of each subterm. This notion of dynamic terms is explored in Chapter 5 where the relationship of a lattice of dynamic terms, ordered by the approximation ordering, to each standard term is discussed. Dynamic substitutions are introduced, the syntactic sort propagations operations of *percolation* and *balancing* are described, and dynamic equations are introduced.

In Chapter 6, dynamic terms are discussed further with the introduction of the concepts

of dynamic matching and unification. The differences in behaviour between dynamic and standard terms become more apparent here. It transpires that for these concepts there is more than one possible analogue corresponding to the standard definitions, each with their own properties. Where there may be a single match in the standard domain, there may be several in the dynamic; a canonical match is given representing the 'best' available. In dynamic unification, although a most general unifier can be defined, unlike in standard unification, the unified term is not unique; in some circumstances a best unified term can be given. A dynamic unification algorithm is given which is unitary, but may not generate well-sorted unifiers, although well-sorted unifiers have a relation to these unifiers. Some discussion follows on how to generate well-sorted unifiers, comparing with other approaches in the literature.

Chapter 7 defines dynamic term rewriting. Dynamic rewriting is similar to standard term rewriting except that it can also modify the sorts information in terms. However, this does not prove to give a sufficiently strong rewriting relation, and so two other forms of rewriting are introduced; sort propagation which performs a sort percolation step, and sort rewriting which uses the right-hand side of rules to modify sorts. This combination is shown to be the computational analogue of dynamic order-sorted equational logic by the statement and proof of a Birkhoff Theorem for dynamic rewriting which shows the soundness and completeness of dynamic rewriting, without using the compatibility property, required for standard rewriting. Termination of dynamic rewriting is briefly discussed and it is shown that it is sufficient to use standard methods to establish termination.

In order to perform automated rewriting proofs, definitions of the Church-Rosser and confluence properties are given, and in addition, the weaker properties of Church-Rosser and confluence modulo sorts are introduced, which will give automated rewriting proofs up to identity of underlying terms. This reminds us that it is not proofs in the dynamic system which are the prime motivation of this work, but proofs in the underlying order-sorted algebra. Local confluence proves sufficient to demonstrate confluence in a terminating rewrite system; however, the extra condition of *local coherence* is required for local confluence modulo sorts to establish confluence modulo sorts. The chapter concludes with the demonstration of some local confluence lemmas for special cases of rewriting.

Work towards generating a decision procedure for dynamic rewriting proofs continues in Chapter 8, where we define critical pairs and note that using the local confluence results of Chapter 7, many critical pairs become redundant. We then discuss the critical pair lemma for both local confluence and local confluence modulo sorts and prove the latter. We then define a completion algorithm and sketch a proof of its correctness. However, this completion algorithm is dependent on two conditions: local coherence, and also the well-sortedness of unifiers. Criteria can be given in special cases to establish these properties. As a more general approach to handling this problem, the unified term is placed in the critical pair explicitly as a *critical triple.* This leads to a modified rewriting method similar to conditional or constrained rewriting, where rewriting can only take place if the condition terms are well-sorted. Methods of simplifying the conditions are considered at the end of the chapter.

9.2 Illustrative Examples

In order to illustrate the dynamic rewriting method, we give some examples of dynamic rewriting. This demonstrates that it can be used to solve some of the well known problems associated with order-sorted specifications. Also some further examples show how it may be possible to extend the method.

Example 9.1. It is worth briefly going over Smolka's Example [SNGM88], discussed in Example 2.66. If with the same signature as before, we have dynamic rules:

Rules: $a \bullet A \rightarrow_A b \bullet B$

$a' \bullet A \rightarrow_A b \bullet B$

Then there is the rewriting proof:

$$f(a \bullet A) \bullet A \rightarrow f(b \bullet A) \bullet A \leftarrow f(a' \bullet A) \bullet A$$

and rewriting is complete. □

Example 9.2. A motivation for dynamic rewriting was the problem of the restriction of

sort-decreasing sets of rules. We re-examine Example 3.2 and can now orient the square equation in the natural way:

Rules: *(rules for _ * _ omitted)*

$$square(x)\bullet Nat \rightarrow_{Nat} (x * x)\bullet Int$$

Thus there is the rewrite sequence:

$$\begin{aligned} square(succ(succ(0\bullet Nat)\bullet Nat)\bullet Nat)\bullet Nat \\ \rightarrow (succ(succ(0\bullet Nat)\bullet Nat)\bullet Nat * succ(succ(0\bullet Nat)\bullet Nat)\bullet Nat)\bullet Nat \\ \xrightarrow{*} succ(succ(succ(succ(0\bullet Nat)\bullet Nat)\bullet Nat)\bullet Nat)\bullet Nat \end{aligned}$$

□

Example 9.3. No treatment of algebraic specifications is complete without a discussion of the Stack example. The stack example is given in dynamic specification as follows:

Sorts: $Elem,\ NeStack,\ Stack$

Subsorts: $NeStack \leq Stack$

Operators: $empty : \rightarrow Stack$

$push : Elem\ \ Stack \rightarrow NeStack$

$pop : NeStack \rightarrow Stack$

$top : NeStack \rightarrow Elem$

Variables: $e, e_1, e_2 : Elem\ s : Stack$

Rules: $pop(push(e, s)\bullet NeStack)\bullet Stack \rightarrow_{Stack} s$

$top(push(e, s)\bullet NeStack)\bullet Elem \rightarrow_{Elem} e$

A well known problem with the standard order-sorted stack example is that the term:

$$t = pop(pop(push(e_1, push(e_2, s))))$$

is ill-formed syntactically. In Section 3.2 in the similar example of sequences, this is resolved using retracts as auxiliary operators to make ill-sorted terms well-sorted. Using dynamic rewriting, we give a more elegant solution, by rewriting the least syntactic dynamic term for t.

$$L(t) \quad = \quad pop(pop(push(e_1, push(e_2, s)\bullet NeStack)\bullet NeStack)\bullet Stack)\bullet\{\}$$

$$\begin{aligned} &\rightarrow^{T} \quad pop(push(e_2, s)\bullet NeStack)\bullet\{\} \\ &\rightarrow^{\Sigma} \quad pop(push(e_2, s)\bullet NeStack)\bullet Stack \\ &\rightarrow^{T} \quad s \end{aligned}$$

Thus t is a well-sorted term of sort *Stack*. Note also that the terms:

$$L(pop(empty)) = pop(empty\bullet Stack)\bullet\{\}$$
$$L(top(empty)) = top(empty\bullet Stack)\bullet\{\}$$

are valid dynamic terms, but are not rewritable, and are thus ill-sorted. □

Example 9.4. Order-sorted algebras were originally intended to handle partial functions and the specification of errors. In this example, we show how such functions can be presented in dynamic order-sorted specification. We repeat the stack example, but this time we do not make *pop* and *top* partial, but introduce error supersorts:

Sorts:	*Elem, ErrElem, Stack, ErrStack*
Subsorts:	$Elem \leq ErrElem$
	$Stack \leq ErrStack$
Operators:	$empty : \rightarrow Stack$
	$push : Elem\ \ Stack \rightarrow Stack$
	$pop : Stack \rightarrow ErrStack$
	$top : Stack \rightarrow ErrElem$
	$errorelem : \rightarrow ErrElem$
	$underflow : \rightarrow ErrStack$
Variables:	$e, e_1, e_2 : Elem\ s : Stack$
Rules:	$pop(push(e, s)\bullet NeStack)\bullet ErrStack \rightarrow_{Stack} s$
	$top(push(e, s)\bullet NeStack)\bullet ErrElem \rightarrow_{Elem} e$
	$pop(empty\bullet Stack)\bullet ErrStack \rightarrow_{ErrStack} underflow\bullet ErrStack$
	$top(empty\bullet Stack)\bullet ErrElem \rightarrow_{ErrElem} errorelem\bullet ErrElem$

The error sort elements can be used as error messages. This is an alternative to Example 9.3 and has much the same properties, and is equally well-behaved as a dynamic rewriting system. □

Example 9.5. In [CH91b], the following example is given, using the unsorted rules and membership constraint notation of [Com92].

$$y \in S \quad f(g(y)) \to b$$
$$h(x) \to l(x)$$

where $S = \{h^i(a)\}$. This is not confluent as an unsorted system as:

$$b \leftarrow f(g(h(a))) \to f(g(l(a)))$$

Chen and Hsiang point out Comon's inference rules generate an infinite set of rules involving higher-order variables and rightly suggest that this system can be more elegantly expressed as an order-sorted specification.

Sorts:	$S\ T$	
Subsorts:	$S \leq T$	
Operators:	$a : \to S$	$b : \to T$
	$f : T \to T$	$h : T \to T$
	$g : T \to T$	$h : S \to S$
	$l : T \to T$	
Variables:	$x : T\ ,\ y : S$	
Rules:	$f(g(y)\bullet T)\bullet T \to_T b\bullet T$	
	$h(x)\bullet T \to_T l(x)\bullet T$	

This set of rules is not sort-decreasing, as $\mathcal{LS}(h(y : S)\bullet T) = S \leq T = \mathcal{LS}(l(y : S)\bullet T)$. In standard rewriting the non-confluence remains, and Chen and Hsiang solve this by adding another rank : $l : S \to S$ using their completion procedure. However, using dynamic rewriting, this set of rules is complete. The non-confluent peak shown above is confluent using dynamic rewriting.

$$b\bullet T \leftarrow f(g(h(a\bullet S)\bullet S)\bullet T)\bullet T \to f(g(l(a\bullet S)\bullet S)\bullet T)\bullet T \to b\bullet T$$

Thus in this example, dynamic rewriting is more compact then either Comon's or Chen and Hsiang's approaches. □

Example 9.6. Dynamic rewriting can be used to capture Term-Sort Declarations, as given in Definition 3.20. Consider the Example 3.21 which describes the Even sort. We can reformulate this example as a set of dynamic rewrite rules as follows.

Sorts:	$Nat,\ Even$
Subsorts:	$Even \leq Nat$
Operators:	$0 : \rightarrow Even$
	$succ : Nat \rightarrow Nat$
Rules:	$succ(succ(x : Even) \bullet Nat) \bullet Nat \rightarrow_{Even} succ(succ(x : Even) \bullet Nat) \bullet Nat$

The Term-Sort Declaration $t : S$ has been transformed into a dynamic rule $L(t) =_S L(t)$. This rule cannot be used for term rewriting, by Definition 7.3, but it can be used for sort rewriting. Thus we can show that $succ^4(0)$ is even as follows:

$$\begin{aligned} L(succ(succ(succ(succ(0))))) &= succ(succ(succ(succ(0 \bullet Even) \bullet Nat) \bullet Nat) \bullet Nat) \bullet Nat \\ &\rightarrow^S succ(succ(succ(succ(0 \bullet Even) \bullet Nat) \bullet Even) \bullet Nat) \bullet Nat \\ &\rightarrow^S succ(succ(succ(succ(0 \bullet Even) \bullet Nat) \bullet Even) \bullet Nat) \bullet Even \end{aligned}$$

Such rewrite rules should be treated exactly as normal rewrite rules. □

Example 9.7. In Chapter 8, term-sort constraints on equations can be generated during completion, representing unified terms, the well-sortedness of which are undetermined. By allowing such conditions in specifications, we can extend the power of the method further. Consider the specification of the subtraction function over the naturals. This is a problematic specification as subtraction in the naturals is only well defined if the first argument is greater than the second. However, consider the following dynamic specification. inspired by a similar example in [MSS90].

Sorts:	$NzNat,\ Nat,\ ErrNat$
Subsorts:	$NzNat \leq Nat \leq ErrNat$
Operators:	$0 : \ \rightarrow Nat$
	$succ : Nat \rightarrow NzNat$
	$pred : NzNat \rightarrow Nat$
	$\_ - \_ : ErrNat\ \ ErrNat \rightarrow ErrNat$[1]
Variables:	$m, n : Nat$
Rules:	$pred(succ(n) \bullet NzNat) \bullet Nat \ \rightarrow_{Nat}\ n$
	$(n - 0 \bullet Nat) \bullet ErrNat \ \rightarrow_{Nat}\ n$
	$(m - n) \bullet NzNat : (m - succ(n)) \bullet ErrNat \ \rightarrow_{Nat}\ pred((m - n) \bullet NzNat) \bullet Nat$

The last rule has a condition that $(m - n) \bullet NzNat \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$. This condition can be used in the following proof, which uses the result of Lemma 8.47 to show that the constraint term is well-sorted.

$$L(succ(succ(0)) - succ(0)) \rightarrow pred(succ(succ(0 \bullet Nat) \bullet NzNat) \bullet NzNat - 0 \bullet Nat)$$
$$\text{if } (succ(succ(0 \bullet Nat) \bullet NzNat) \bullet NzNat - 0 \bullet Nat) \bullet NzNat \in \mathcal{D}_{\mathcal{S}}(\mathcal{X})$$

and since we can rewrite the condition to a well-sorted term of sort $NzNat$:

$$succ(succ(0 \bullet Nat) \bullet NzNat) \bullet NzNat - 0 \bullet Nat \ \rightarrow\ succ(succ(0 \bullet Nat) \bullet NzNat) \bullet NzNat$$

the condition holds and we can rewrite further:

$$pred((succ(succ(0 \bullet Nat) \bullet NzNat) \bullet NzNat - 0 \bullet Nat) \bullet NzNat) \bullet Nat$$
$$\rightarrow pred(succ(succ(0 \bullet Nat) \bullet NzNat) \bullet NzNat) \bullet Nat$$
$$\rightarrow succ(0 \bullet Nat) \bullet NzNat$$

This is an elegant solution to the example, and it is speculated that this method could be exploited further. □

Example 9.8. A further extension could be the addition of conditions on the lattice of sorts. For example, we could declare that certain subsorts are uninhabited. This could be used as a semi-decision procedure for refutational theorem proving.

[1] This operator declaration for $\_ - \_$ is somewhat unsatisfactory. It would be preferable to have the least syntactic sort of terms involving $\_ - \_$ as $\{\}$ and leave their sorting entirely to the equations. However, our definition of signature does not allow this.

Sorts:	$Bool,\ True,\ False$
Subsorts:	$True \leq Bool,\ False \leq Bool$
Sort Conditions:	$True \bowtie False$
Operators:	$tt: \rightarrow True$
	$ff: \rightarrow False$

The condition $True \bowtie False$ declares that the intersection between these two sorts is empty, that is we will accept no models which have common elements of both sorts. We can thus use this for refutational theorem proving. If during completion the (unconditional) equality $d_1 =_{\{True,False\}} d_2$ is generated, this equality contradicts the hypothesis that $True \bowtie False$, and thus the initial set of equations is inconsistent. In [DK88] and [Mat88], using the example of Henkin models given in [Pau85], a simple boolean calculus is used to show that in these models, the binary operator LE is transitive, using axioms which we omit for brevity. Adding the dynamic equalities:

$$\begin{aligned} LE(A,B)\bullet Bool &=_{True} tt\bullet True \\ LE(B,C)\bullet Bool &=_{True} tt\bullet True \\ not(LE(A,C)\bullet Bool)\bullet Bool &=_{True} tt\bullet True \end{aligned}$$

for some Skolem constants A, B, C, completion (only standard unification is required), results in the generation of the equality:

$$tt\bullet True =_{\{True,False\}} ff\bullet False$$

In ERIL, an exceptional rule is built into the system to recognise this contradiction. By allowing the condition to be placed on the sort in dynamic sorting, we can give a formal justification for this and also generalise it to sorts other than Boolean. □

Further extensions to the theory could be developed by allowing greater expressiveness in the definition of sort structures. For example, higher-order and dependent sorts could be defined. However, these extensions are outside the domain of this thesis and within the general domain of extensions to Universal Algebra such as Equational Typed Logic [MSS90]. Nevertheless, we can speculate that via such extensions, dynamic rewriting could be used as a computational method for ETL, perhaps a more satisfactory method than using conditional rewriting as described in [MSS90].

9.3 Future Research Directions

There are many possible areas for future research on dynamic order-sorted rewriting. We suggest below several which immediately present themselves as candidates.

9.3.1 Outstanding Issues

Some issues remain outstanding in the theoretical work presented in this thesis.

Establishing Local Coherence

The establishment of local coherence in more general cases is an outstanding problem. This is entirely reasonable property for a dynamic rewriting system to have. Indeed, it is in many ways the purpose of dynamic rewriting that such the rewriting relation has this property: in Examples 7.4 and 7.7, static and dynamic sort rewriting are introduced as a result of a lack of local coherence. However, local coherence has proved an elusive property to demonstrate, and further analysis of the dynamic rewriting relation is required to find further sufficient conditions for it.

Dynamic Unification

The dynamic unification algorithm which is given in Section 6.2 generates a unifier which may not be well-sorted. It is important that we can distinguish well-sorted unifiers, although in general this is undecidable. The dynamic unifier can be used as a basis to analyse particular classes of specifications with a view to developing sufficient conditions for well-sortedness.

The analysis would have to concentrate on two rules of the unification algorithm, Intersect and Abstract, which are the two rules that can generate ill-sorted unifiers. The Intersect rule is closely related to Sort Inhabitedness, and sufficient conditions need to be developed

to establish that property. The Abstract rule is more difficult to analyse. This is dependent on whether specific terms, or instances of those terms, are of certain sorts.

Constrained Dynamic Completion

The problem of deciding the well-sortedness of unification is closely related to that of completion in the presence of term constraints, as described in Chapter 8, and this method needs more work to find its strengths and limitations. Constraint simplification can also provide an interesting area for future research.

An area closely related to unification is *Narrowing* [Fay79, Hul80] and other generalised unification methods [GS89, Sny91]. A direction of future work would be to explore these methods in the dynamic context to solve equations, and may well shed light on the general problem of dynamic unification, using the constraint simplification rules as a starting point.

9.3.2 Extensions to Dynamic Rewriting

The theoretical work presented in this thesis could be extended in several ways.

Termination of Dynamic Rewriting

As mentioned in Section 7.3, it may be possible to exploit the added expressive power of dynamic terms to establish the termination of sets of dynamic rewrite rules. The paper by Gnaedig [Gna92b] makes a start in this area in the context of the standard order-sorted rewriting theories, and this work could straightforwardly be expressed in terms of the dynamic order-sorted method. However, Gnaedig's method is only defined in the context of sets of sort-decreasing rules; in the case of dynamic term-rewriting we are not bound by this restriction. It would be interesting to explore new methods of establishing termination for dynamic rewriting.

Dynamic Rewriting modulo Equational Theories

The standard rewriting theory given and implemented in Chapter 2 covers rewriting modulo an equational theory, and the associative-commutative theory in particular as a useful special case. The dynamic rewriting theory does not consider rewriting modulo equations for simplicity in its presentation, and it would be interesting to consider what implications working modulo equations would have for dynamic rewriting. As seen in Section 2.6, unification modulo equational theories is not straightforward in standard order-sorted logic requiring the property of *sort compatibility.* It is speculated that some of these restrictions can be ignored in dynamic unification, similar to the removal of the compatibility condition on rewrite rules. However, it is likely that extra complications will arise, and this area is one for future research.

Further Extensions to the Method

In Section 9.2 some examples are given which extend the dynamic order-sorted method. Adding term-sort declarations is the simplest of these, and indeed they can be included without extending dynamic rewriting at all. Nevertheless, it is necessary to reconcile existing work in this area with this proposed method of handling such declarations.

The further extensions are perhaps more interesting. Adding a term well-sortedness condition into the specification as in Example 9.7 is an small extension to the method, and clearly this would be covered in further research into the conditional rewriting method sketched in Chapter 8. Adding more sophisticated conditions moves the method further into generalised universal algebras such as ETL. As mentioned earlier, it would be interesting to see how far the dynamic rewriting method can be extended into these areas to provide a computational analogue to ETL.

Closely related is the idea of extending the notion of sort conditions beyond simple ordering conditions, as in Example 9.8. This needs to be worked through to demonstrate that the method does work and the theoretical underpinnings need to be verified. Sort disjointness constraints have been investigated in the related automated reasoning technique of

resolution [BP92]: this could form the basis of the work in dynamic rewriting.

9.3.3 Future Developments to MERILL

The MERILL system is still under development and at present provides a basic system for equational reasoning using completion. As a base of experience using the system grows, changes in adaptability and use will no doubt arise. The system is designed to be flexible and easy to modify for experimenting with new tools in equational reasoning. Future developments for MERILL include:

- **Tactic Language.** Several existing provers, notably ERIL and ORME have a flexible system of tactics and configurations to allow the user to adapt the use of the system to their own requirements, and it would enhance the usability and applicability of MERILL to give it such a tactic language.

- **Orderings.** The orderings available in the MERILL system are inflexible at present. They are tedious to use as they demand the user to set up the precedence and weights in advance, and only the limited Associative Knuth-Bendix ordering is applicable in the presence of AC operators. Several enhancements could improve this.

  1. Further improvements to the implementation of a Incremental Knuth-Bendix Ordering. Nick Cropper at the University of St Andrews has been investigating this area.
  2. Implementation of an automatic Recursive Path Ordering.
  3. Investigation and implementation of further AC compatible orderings.
  4. Investigation and implementation of further Order-sorted compatible orderings.

- **Induction.** Inductive proof is a central method of reasoning with equations and it would be useful to provide structural induction in the style of the Larch Prover.

- **User Interface.** The teletype user interface is unsophisticated and can be tedious to use. It would be desirable to design and implement a graphical user interface for MERILL.

- **Module Support.** A longer term plan for the system is to allow for module and theory store to be integrated into the system for larger scale program specification and refinement.

The major development which awaits MERILL is an implementation of the dynamic rewriting method outlined in this thesis. Some aspects of this are discussed in Section 9.3.4.

9.3.4 Implementing Dynamic Order-Sorted Rewriting

In this section some of the changes required to implement dynamic equational reasoning are discussed.

Changes to the Signature

The main change in the signature's implementation is to allow sets of sorts to be manipulated as sorts. We give the following data type in Standard ML.

abstype *Sort* = *sort* **of** *string* | *M* **of** *Sort list*

A sort is which has two alternative constructors, either a named sort or a list of sorts, minimised using the function *meetSort*, which calculates $M_\Sigma(S_1 \cup S_2)$ and has the following type.

val *meetSort : (Sort * Sort → bool) → Sort → Sort → Sort*

This function takes the current sort ordering and forms the meet of the two argument sorts. The rest of the functions for handling sorts are unchanged. These changes are propagated into the functions handling operators and variables, so for instance a variable can be of an intersection sort.

Changes to Terms

To represent dynamic terms, the sort of a term is explicitly stored with the term and may change with the history of the term. Thus we have a data type for terms as follows.

```
abstype Term = VarTerm of Variable
             | OpTerm of OpId * (Term list) * Sort
```

Terms have two constructors, one for variable considered as terms, and the other for compound terms, with an operator symbol and a list of subterms. This constructor also has a sort argument, representing the current dynamic sort of the term. We give functions which directly manipulate this sort.

```
(* dynamic functions *)
val dynamic_sort : Term → Sort
val assert_sort : Signature → Term → Sort → Term
```

dynamic_sort directly returns the current dynamic sort of a term. *assert_sort* allows us to change the sort of a term by taking the meet sort of the current dynamic sort and the sort that is being asserted for the term.

We also give the static sorting functions which are described in detail in Chapter 7.

```
(* sort propagation functions *)
val percolate : Signature → Term → Term
val balance : Signature → Term → Term
```

Algorithms are required for these. For sort percolation, the standard least-sort algorithm to sort terms suffices, using the current dynamic sorts of subterms and percolating that information up the tree. For balancing, as it is a global property of the term, this strategy is not applicable. To perform balancing efficiently, it is necessary to scan the term from its leaves, considering subterms of equal height at each level, and balancing those.

The Matching Algorithms

In a naive implementation the algorithms for upward strong and weak matching are straightforward modifications of a standard order-sorted matching algorithm which implements the rules in Figures 6.1 and 6.2 directly via case analysis of the forms of the pattern and target terms. For weak matching, the variable-term case checks that the current dynamic sort of the term to be matched by the variable is less than the sort of the variable. This is similar to the syntactic case where the least sort of the target term is checked to be less than the sort of the variable. Upward strong matching has the extra condition in the term-term case that the current dynamic sort of the pattern term has to be greater than or equal to the current dynamic sort of the balanced target term.

Dynamic Unification

The unification algorithm is in many ways simpler than that for ordinary syntactic order-sorted unification. It is unitary so we can follow the same strategy for unification as unsorted unification. The implementation currently in MERILL is a naive one with a straightforward implementation of the given rules. It should be possible to derive a linear algorithm in the style of Paterson and Wegman [PW78] or a quasi-linear one in the style of Martelli and Montenari [MM82].

The major difference between these approaches and dynamic unification is that dynamic unification requires us to derive the unified term as well as the substitution. This also is straightforward as we can build the (maximal upward strong) unified term as we are unifying, building the term from the unificands from the root towards the leaves.

Dynamic Rewriting

Dynamic term rewriting is exactly as ordinary term rewriting, except that we use either the strong or the weak matching algorithms. The major difference is the introduction of sort propagation and rewriting. These steps are carried out in an analogous way to term

rewriting.

In conventional rewriting, there is a question of which is the best strategy to adopt in rewriting, outermost and innermost rewriting being the most well-known strategies, each with its advantages and disadvantages. As dynamic rewriting is a combination of three rewriting techniques, we have at our disposal a much wider variety of strategies, given that the order of application of the rules is arbitrary.

For example, one possibility would be to interleave term rewriting and sort normalisation, with propagation taking place. After each term-rewriting step, the sorts of resulting terms are normalised. A term may also need to be sort-normalised initially to allow a term-rewriting step.

Definition 9.9. Alternation Strategy. Given a set of dynamic rewrite rules, R the Dynamic Alternation Rewriting Relation $\rightarrow^A$, is given by $\xrightarrow{1}{}^{T} \circ (\xrightarrow{*}{}^{S} \cup \xrightarrow{*}{}^{\Sigma})$.

Other strategies can be devised, and which is most appropriate requires more analysis.

9.4 Conclusions

This thesis has given an account of the current state of order-sorted specification, and reports the first successful implementation of order-sorted associative-commutative term-rewriting and completion in a practical equational theorem proving system.

However, investigation of the standard order-sorted theory soon exposes its difficulties and the thesis gives an alternative approach of *dynamic* order-sorted algebra, logic and rewriting. This work has been largely successful; it gives a clean approach to order-sorted algebra, with an elegant equational logic, without the extra complication of the more ambitious approaches of reformulating the entire paradigm of universal algebra.

This logic is realised for automated theorem proving using dynamic rewriting, a sound and complete, but practical approach to the computational aspects of the algebra. The use of

rewrite rules in two ways to modify terms and sorts is more elegant than using different rules for these purposes. It has been demonstrated that completion algorithms can be defined for dynamic rewriting, and criteria can be given to generate decision procedures for order-sorted equations. Two conditions are identified for completion modulo sorts: coherence of the rewriting relation and well-sorted unification. This lays the basis for sufficient conditions in special classes of rewrite system. We give sufficient conditions for larger class of specifications than the standard theory.

A more general approach is offered by constrained dynamic rewriting; again, we have identified that sufficient conditions for well-sorted constraints are required, and we give some rules to reduce constraints

Dynamic order-sorted algebra, logic and rewriting solve the immediate problems of the standard method. However, this work is not complete. There remain unsolved problems in the area of completion and unification. Further criteria need to be defined to extend the class of specifications upon which the dynamic completion algorithms are applicable.

This work makes a contribution to automated theorem proving by laying the theoretical foundations for dynamic order-sorted reasoning. Future work can use this framework to analyse the intriguing nature of order-sorted specification and more generally typed algebraic systems. It is inevitable that solutions will be at best partial as we are working on a narrow edge between decidability and undecidability.

Bibliography

[Abr96] J-R. Abrial. *Deriving Programs from Meaning.* Prentice Hall International, 1996. To appear.

[AK90] M. Adi and C. Kirchner. The AC-unification race: the system solving approach. In A. Miola, editor, *Proceedings of DISCO'90*, volume 429 of *Lecture Notes in Computer Science*, pages 174–183. Springer-Verlag, 1990.

[Bac87] L. Bachmair. *Proof Methods for Equational Theories.* PhD thesis, University of Illinois, USA, 1987.

[Bac91] L. Bachmair. *Canonical Equational Proofs.* Birkhauser, 1991.

[BCD90] A. Boudet, E. Contejean, and H. Devie. A new AC unification algorithm with an algorithm for solving systems of diophantine equations. In *IEEE Symposium on Logic in Computer Science, 1990*, pages 289–299, 1990.

[BG80] R. M. Burstall and J. A. Goguen. The semantics of CLEAR, a specification language. In *Proceedings of Advanced Course on Abstract Software Specification*, volume 86 of *Lecture Notes in Computer Science*, pages 292–332. Springer-Verlag, 1980.

[BL79] A. M. Ballantyne and D. S. Lankford. New decision algorithms for finitely presented commutative semigroups. Technical Report ATP-59, University of Texas, Austin, Department of Mathematics and Computer Science, 1979.

[BL87] A. Ben Cherifa and P. Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9(2):137–159, 1987.

[Blo91] S. Blott. *An Approach to Overloading with Polymorphism.* PhD thesis, University of Glasgow, 1991. Deptartment of Computing Science, Technical Report FP-1992-1.

[Bou92] A. Boudet. Unification in order-sorted algebras with overloading. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Computer Science*, pages 193–207. Springer-Verlag, 1992.

[BP92] T. Bollinger and U. Pletat. An order-sorted logic with sort literals and disjointness constraints. In B. Nebel, C. Rich, and W. Swartout, editors, *KR'92: Proceedings of the 3rd International Conference on the Principles of Knowledge Representation and Reasoning*, pages 413–424. Morgan Kaufmann Publishers, 1992.

[BS93] F. Baader and J.H. Siekmann. Unification theory. In D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming.* Oxford University Press, Oxford, UK, 1993.

[Bün93] R. Bündgen. Reduce the Redex → ReDuX. In C. Kirchner, editor, *Proceedings of the 5th International Conference on Rewriting Techniques and Applications*, volume 690 of *Lecture Notes in Computer Science*, pages 446–450. Springer-Verlag, 1993.

[BW82] M. Broy and M. Wirsing. Partial abstract types. *Acta Informatica*, 18:47–64, 1982.

[BW88] R. Bird and P. Wadler. *Introduction to Functional Programming.* Prentice Hall International, 1988.

[CD85] R. J. Cunningham and A. J. J. Dick. Rewrite systems on a lattice of types. *Acta Informatica*, 22:149–169, 1985.

[CF89] M. Clausen and A. Fortenbacher. Efficient solution of linear diophantine equations. *Journal of Symbolic Computation*, 8:201–216, 1989.

[CH91a] Hong Chen and Jieh Hsiang. Logic programming with recurrence domains. In B. Monien J. Leach Albert and M. Rodriguez Artalejo, editors, *Proceedings of the 18th ICALP, Madrid (Spain)*, number 510 in Lecture Notes in Computer Science. Springer-Verlag, 1991.

[CH91b] Hong Chen and Jieh Hsiang. Order-sorted equational specification and completion. State University of New York, Stony Brook. Unpublished Draft, 1991.

[Com90] H. Comon. Equational formulas in order-sorted algebras. In M.S. Paterson, editor, *Proceedings of the International Conference on Algebra and Logic Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 674–688. Springer-Verlag, 1990.

[Com92] H. Comon. Completion of rewrite systems with membership constraints. In W. Kuich, editor, *Proceedings of the 19th International Conference on Algebra and Logic Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 392–403. Springer-Verlag, 1992.

[Cro92] N. Cropper. Implementing the automated Knuth-Bendix ordering. 4th year project report, Department of Computing Science, University of Glasgow, 1992.

[Der87] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3:69–116, 1987.

[Dia91] R. Diaconescu. The order-sorted paradigm. University of Oxford, Unpublished Note, 1991.

[Dic85] A. J. J. Dick. ERIL. Equational Reasoning: an Interactive Laboratory. In B. Buchberger, editor, *Proceedings of the EUROCAL conference.* Springer-Verlag, 1985.

[Dic87] A. J. J. Dick. *Order-Sorted Equational Reasoning and Rewrite Systems.* PhD thesis, Imperial College, University of London, 1987.

[DK88] A. J. J. Dick and J. R. Kalmus. ERIL (Equational Reasoning: an Interactive Laboratory) user's manual. Technical Report RAL-88-055, Rutherford Appleton Laboratory, 1988.

[DKM90] A. J. J. Dick, J. R. Kalmus, and U. H. Martin. Automating the Knuth-Bendix ordering. *Acta Informatica*, 28:95–119, 1990.

[DM79] N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the Association for Computing Machinery*, 22:465–476, 1979.

[Dom92] E. Domenjoud. A technical note on AC-unification. The number of minimal unifiers of the equation $\alpha x_1 + \cdots + \alpha x_p =_{AC} \beta y_1 + \cdots + \beta y_q$. *Journal of Automated Reasoning*, 8(1):39–44, 1992.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1985.

[EMCO92] H. Ehrig, B Mahr, I Classen, and F Orejas. Introduction to algebraic specification. Part 1 : Formal methods for software development. *The Computer Journal*, 35(5):460–467, 1992.

[EMO92] H. Ehrig, B Mahr, and F Orejas. Introduction to algebraic specification. Part 2 : From classical view to foundations of system specifications. *The Computer Journal*, 35(5):468–477, 1992.

[Fag87] F. Fages. Associative-commutative unification. *Journal of Symbolic Computation*, 3:257–275, 1987.

[Fay79] M. Fay. First-order unification in an equational theory. In *Proceedings of the 4th workshop on Automated Deduction*, pages 161–167, Austin, Texas, 1979.

[FH83] F. Fages and G. Huet. Complete sets of unifiers and matchers in equational theories. In *Proceedings of CAAP'83*, volume 159 of *Lecture Notes in Computer Science*, pages 205–220. Springer-Verlag, 1983.

[For84] R. Forgaard. A program for generating and analyzing term rewriting systems. Master's thesis, Massachusetts Institute of Technology, 1984.

[For87] A. Fortenbacher. An algebraic approach to unification under associativity and commutativity. *Journal of Symbolic Computation*, 3:217–229, 1987.

[Gan89] H. Ganzinger. Order-sorted completion: the many-sorted way. In M. Diaz and F. Orejas, editors, *Proceedings of TAPSOFT'89, Volume 1*, volume 351 of *Lecture Notes in Computer Science*, pages 244–258. Springer-Verlag, 1989.

[Gan91a] H. Ganzinger. A Completion Procedure for Conditional Equations. *Journal of Symbolic Computation*, 11:51–81, January 1991.

[Gan91b] H. Ganzinger. Order-sorted completion: the many-sorted way. *Theoretical Computer Science*, 89(1):3–32, 1991.

[GD92] J. Goguen and R. Diaconescu. A survey of order-sorted algebra. University of Oxford, Unpublished Note, 1992.

[GDLE84] M. Gogolla, K. Drosten, U. Lipeck, and H-D. Ehrich. Algebraic and Operational Semantics of Specifications Allowing Exceptions and Errors. *Theoretical Computer Science*, 34:289–313, 1984.

[GG89] S. J. Garland and J. V. Guttag. An overview of LP, the Larch Prover. In N. Dershowitz, editor, *Proceedings of the 3rd Conference on Rewriting Techniques and Applications*, volume 355 of *Lecture Notes in Computer Science*, pages 137–151. Springer-Verlag, 1989.

[GH78] J. V. Guttag and J. J. Horning. The algebraic specification of abstract data types. *Acta Informatica*, 10:27–52, 1978.

[GHW85] J. Guttag, J. J. Horning, and J. Wing. Larch in five easy pieces. Technical Report 5, DEC Systems Research Centre, Palo Alto, CA., 1985.

[GI88] J. H. Gallier and T. Isakowitz. Rewriting in order-sorted equational logic. In *Proceedings of the 5th International Conference and Symposium on Logic Programming*, volume 1, pages 280–294. MIT Press, 1988.

[GJM85] J. A. Goguen, J-P. Jouannaud, and J. Meseguer. Operational semantics for order-sorted algebra. In *Proceedings of the International Conference on Automata, Languages and Programming*, volume 194 of *Lecture Notes in Computer Science*, pages 221–231. Springer-Verlag, 1985.

[GKK90] I. Gnaedig, C. Kirchner, and H. Kirchner. Equational completion in order-sorted algebras. *Theoretical Computer Science*, 72:169–202, 1990.

[GM87] J. A. Goguen and J. Meseguer. Models and equality for logical programming. In *Proceedings of TAPSOFT'87*, volume 250 of *Lecture Notes in Computer Science*, pages 1–22. Springer-Verlag, 1987.

[GM88] J. A. Goguen and J. Meseguer. Unifying Functional Object-Oriented and Relational Programming with Logical Semantics. Technical Report SRI-CSL-87-7, SRI International Computer Science Laboratory, 1988.

[GM89] J. A. Goguen and J. Meseguer. Order-sorted algebra i: Equational deduction for multiple inheritence overloading, exceptions and partial operations. Technical Report PRG-80, University of Oxford, 1989.

[Gna92a] I. Gnaedig. ELIOS-OBJ. Theorem proving in a specification language. In B. Kreig-Bruckner, editor, *Proceedings of ESOP'92*, volume 582 of *Lecture Notes in Computer Science*, pages 182–199. Springer-Verlag, 1992.

[Gna92b] I. Gnaedig. Termination of order-sorted rewriting. In Kirchner and Levy, editors, *Proceedings of the 3rd Conference on Algebraic and Logic Programming*, volume 632 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.

[Gog78a] J. A. Goguen. Abstract errors for abstract data types. In J. Dennis, editor, *Proceedings of the IFIP Working Conference on Formal Descriptions of Programming Concepts*. North-Holland, 1978.

[Gog78b] J. A. Goguen. Order-sorted algebra. Technical Report 14, UCLA Computer Science Department, 1978.

[Gog84] M. Gogolla. Partially ordered sorts in algebraic specifications. In *Proceedings of the 9th CAAP: Colloq. on Trees in Algebra and Programming*, pages 139–153. Cambridge University Press, 1984.

[Gog87] M. Gogolla. On parametric algebraic specifications with clean error handling. In *Proceedings of the Joint Conference on Theory and Practice of Software Development*, volume 249 of *Lecture Notes in Computer Science*, pages 81–95. Springer-Verlag, 1987.

[Gog88] J. A. Goguen. Higher-order functions considered unnecessary for higher-order programming. Technical Report SRI-CSL-88-1, SRI International Computer Science Laboratory, 1988.

[GS89] J. H. Gallier and W. Snyder. Complete sets of transformations for general E-unification. *Theoretical Computer Science*, 67(2):203–260, 1989.

[GSHH92] J. A. Goguen, A. Stevens, H. Hilberdink, and K. Hobley. 2OBJ: a metalogical framework theorem prover based on equational logic. *Philosophical Transactions of the Royal Society London, Series A.*, 339:69–86, 1992.

[GTW79] J. A. Goguen, J. W. Thatcher, and E. G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In R. Yeh, editor, *Current Trends in Programming Methodology*, volume IV, pages 80–149. Prentice-Hall, 1979.

[GW88] J. A. Goguen and T. Winkler. Introducing OBJ3. Technical Report SRI-CSL-88-9, SRI International Computer Science Laboratory, 1988.

[Her30] J. Herbrand. *Recherches sur la Théorie de la Démonstration.* PhD thesis, Sorbonne, Paris, 1930. Reprinted in Logical Writings, Ed. W.D.Goldfarb, Reidel, 1971.

[Her67] J. Herbrand. Investigations in proof theory: The properties of true propositions. In J. van Heijenoort, editor, *From Frege to Gödel: A Source book in Mathematical Logic, 1879-1931*, pages 525–581. Harvard University Press, 1967.

[HKK93] C. Hintermeier, C. Kirchner, and H. Kirchner. Dynamically-typed computations for order-sorted equational presentations. Technical Report CRIN 93-R-309, Centre de Recherche en Informatique de Nancy, 1993.

[HKK94] C. Hintermeier, C. Kirchner, and H. Kirchner. Dynamically-typed computations for order-sorted equational presentations (extended abstract). In *Proceedings of the 21st International Conference on Automata, Languages and Programming*, volume 820 of *Lecture Notes in Computer Science*, pages 450–461. Springer-Verlag, 1994.

[HMT88] R. Harper, R. Milner, and M. Tofte. The definition of Standard ML, version 2. Technical Report ECS-LFCS-88-62, LFCS, University of Edinburgh, 1988.

[HO80] G. Huet and D. C. Oppen. Equations and rewrite rules: A survey. In R. V. Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 349–393. Academic Press, 1980.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall International, 1985.

[HPJE92] P. Hudak, S. Peyton-Jones, and P. Wadler (Editors). Report on the Programming Language Haskell A Non-strict, Purely Functional Programming Language Version 1.2. *ACM SIGPLAN Notices*, 27(5), 1992.

[HR87] Jieh Hsiang and M. Rusinowitch. On word problems in equational theories. In T. Ottmann, editor, *Proceedings of the 14th International Colloquium on Automata, Languages, and Programming*, volume 267 of *Lecture Notes in Computer Science*, pages 54–71. Springer-Verlag, 1987.

[HS86] J. R. Hindley and J. P. Seldin. *Introduction to Combinators and λ-calculus.* Cambridge University Press, 1986.

[Hue80] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the Association for Computing Machinery*, 27(4):797–821, 1980.

[Hue81] G. Huet. A complete proof of the correctness of the Knuth-Bendix completion algorithm. *Journal of Computer and Systems Sciences*, 23(1):11–21, 1981.

[Hul80] J-M. Hullot. Canonical forms and unification. In W. Bibel and R. Kowalski, editors, *Proceedings of the 5th International Conference on Automated Deduc-*

tion, volume 87 of *Lecture Notes in Computer Science*, pages 318–334, Les Arcs, France, 1980. Springer-Verlag.

[IG88] T. Isakowitz and J. H. Gallier. Congruence closure in order-sorted algebra. Technical report, Department of Computer and Information Sciences, University of Pennsylvania, Philadelphia, PA, 1988.

[ISO88] International Standards Organisation, ISO IS 8807. *LOTOS - A Formal Description Technique based on the Temporal Ordering of Observed Behaviour*, 1988.

[JD90] J-P. Jouannaud and N. Dershowitz. Rewrite systems. In *Handbook of Theoretical Computer Science*, volume 2. Elsevier Science Publishers B.V., 1990.

[JK86] J-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal of Computing*, 15(4):1155–1194, 1986.

[JK91] J-P. Jouannaud and C. Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In J-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson.* MIT Press, 1991.

[JL82] J-P. Jouannaud and P. Lescanne. On multiset orderings. *Information Processing Letters*, 15(2):57–63, 1982.

[Jon90] C. B. Jones. *Systematic Softare Development using VDM.* Prentice Hall International, 2nd edition, 1990.

[Jou93] J-P. Jouannaud. Rewrite Proofs and Computations. In H. Schwichtenberg, editor, *Proof and Computation*, volume 139 of *NATO ASI Series. Series F: Computer and Systems Sciences*, pages 271–316. Springer-Verlag, 1993.

[Kap87] S. Kaplan. Simplifying Conditional Term Rewriting Systems: Unification, Termination and Confluence. *Journal of Symbolic Computation*, 4(3):295–334, 1987.

[KB70] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263 – 297. Pergamon Press, 1970.

[Kir87] C. Kirchner. Methods and tools for equational unification. Technical Report 87-R-008, Centre de Recherche en Informatique de Nancy, 1987.

[Kir88] C. Kirchner. Order-sorted equational unification. In *5th International Conference on Logic Programming, Seattle*, 1988.

[Kir89] C. Kirchner. From unification in combination of equational theories to a new AC-unification algorithm. In *Resolution of Equations in Algebraic Structures*, pages 171–210. Academic Press, 1989.

[KKM88] C. Kirchner, H. Kirchner, and J. Meseguer. Operational semantics of OBJ-3. In *Proceedings of the 9th International Conference on Automata Languages and Programming*, volume 241 of *Lecture Notes in Computer Science*. Springer-Verlag, 1988.

[Klo90] J. W. Klop. Term rewriting systems. Technical Report CS-R9073, Centrum voor Wiskunde en Informatica, 1990.

[KST96] S. Kahrs, D. Sannella, and A. Tarlecki. The definition of Extended ML: a gentle introduction. *Theoretical Computer Science*, 1996. To appear. A minor revision of LFCS, University of Edinburgh Report ECS-LFCS-95-322, 1995.

[KZ89] D. Kapur and Hantao Zhang. An overview of the Rewrite Rule Laboratory (RRL). In N. Dershowitz, editor, *Proceedings of the 3rd Conference on Rewriting Techniques and Applications*, volume 355 of *Lecture Notes in Computer Science*, pages 559–563. Springer-Verlag, 1989.

[LB77] D. S. Lankford and A. M. Ballantyne. Decision procedures for simple equational theories with permutative axioms: Complete sets of permutative reductions. Technical Report ATP-37, University of Texas, Austin, Department of Mathematics and Computer Science, 1977.

[LC89] P. Lincoln and J. Christian. Adventures in associative-commutative unification. *Journal of Symbolic Computation*, 8:217–240, 1989.

[Les89] P. Lescanne. Completion procedures as transition rules + control. In M. Diaz and F. Orejas, editors, *Proceedings of TAPSOFT'89*, volume 351 of *Lecture Notes in Computer Science*, pages 28–41. Springer-Verlag, 1989.

[Les90] P. Lescanne. Implementation of completion by transition rules + control: ORME. In *Proceedings of the 10th ACM Symposium on Principles of Programming Languages*, volume 463 of *Lecture Notes in Computer Science*, pages 262–269. Springer-Verlag, 1990.

[LZ74] B. H. Liskov and S. N. Zilles. Programming with abstract data types. In *Proceedings of the ACM Symposium on Very High Level Programming Languages*, volume 9 of *ACM SIGPLAN Notices*, pages 50–59, 1974.

[Mat88] B. M. Matthews. Strategies for theorem proving in an equational reasoning system. Master's thesis, Imperial College, University of London, 1988.

[Mat92] B. M. Matthews. Reusing functional code using type classes for library search. Presented at ERCIM workshop on Software Reuse, Heraklion, Crete, ERCIM Workshop Report ERCIM-92-W006, October 1992.

[Mat93a] B. M. Matthews. MERILL: An equational reasoning system in standard ML. In C. Kirchner, editor, *Proceedings of the 5th International Conference on Rewriting Techniques and Applications*, volume 690 of *Lecture Notes in Computer Science*, pages 441–445. Springer-Verlag, 1993.

[Mat93b] B. M. Matthews. MERILL: An equational reasoning system in standard ML - A user guide. Technical Report RAL-93-026, Rutherford Appleton Laboratory, 1993.

[Még92] A. Mégrelis. Partial algebra + order sorted algebra = galactic algebra. In A. Nerode and M. Taitslin, editors, *Logical Foundations of Computer Science - Tver'92*, volume 620 of *Lecture Notes in Computer Science*, pages 314–325. Springer-Verlag, 1992.

[MGS89] J. Meseguer, J. A. Goguen, and G. Smolka. Order-sorted unification. *Journal of Symbolic Computation*, 8:383–413, 1989.

[Mil78] R. Milner. A theory of type polymorphism in programming. *Journal of Computer and Systems Sciences*, 17:348–375, 1978.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall International, 1989.

[MM82] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, 1982.

[MN90a] U. Martin and T. Nipkow. Automating Squiggol. In M. Broy and C.B. Jones, editors, *Programming Concepts and Methods*, chapter 24, pages 233–246. Elsevier Science Publishers, 1990.

[MN90b] U. Martin and T. Nipkow. Ordered rewriting and confluence. In *Proceedings of the 10th International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Computer Science*, pages 366–380. Springer-Verlag, 1990.

[Mos89a] P. D. Mosses. Unified algebras and action semantics. In *STACS'89, Proceedings of the Symposium on Theoretical Aspects of Computer Science*, volume 349 of *Lecture Notes in Computer Science*. Springer-Verlag, 1989.

[Mos89b] P. D. Mosses. Unified algebras and institutions. In *LICS'89, Proceedings of the 4th Annual Symposium on Logic in Computer Science*, pages 304–312. IEEE Computer Society Press, 1989.

[Mos89c] P. D. Mosses. Unified algebras and modules. In *POPL'89, Proceedings of the 16th Annual ACM Symposium on Principles of Programming Languages*, pages 329–343. ACM Press, 1989.

[Mos92] P. D. Mosses. The use of sorts in algebraic specifications. In M. Bidoit and C. Choppy, editors, *Recent Trends in Data Type Specification: Selected Papers from the 8th Workshop on Abstract Data Type joint with the 3rd COMPASS Workshop*, volume 655 of *Lecture Notes in Computer Science*, pages 66–92. Springer-Verlag, 1992.

[MSS89a] V. Manca, A. Salibra, and G. Scollo. DELTA: a deduction system integrating equational logic and type assignment. In *AMAST'89, International Conference on Algebraic Methodology and Software Technology, Iowa City, Iowa, USA*, 1989.

[MSS89b] V. Manca, A. Salibra, and G. Scollo. On the nature of TELLUS. In A. Kreczmar and G. Mirkowska, editors, *MFCS 90: 15th Symposium on Mathematical*

Foundations of Computer Science, volume 379 of *Lecture Notes in Computer Science*, pages 338–349. Springer-Verlag, 1989.

[MSS89c] V. Manca, A. Salibra, and G. Scollo. Reasoning with equations and type assignments. In *3rd Italian Conference on Theoretical Computer Science, Mantova, Italy*, 1989.

[MSS90] V. Manca, A. Salibra, and G. Scollo. Equational typed logic. *Theoretical Computer Science*, 77(1-2):131–159, 1990. Also University of Twente Memorandum INF-89-43, 1989.

[New42] M. H. A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2):223–243, 1942.

[NS91] T. Nipkow and G. Snelting. Type classes and overloading resolution via order-sorted unification. In *Functional Programming Languages and Computer Architecture*, 1991.

[Pau85] E. Paul. On Solving the Equality Problems in Theories defined by Horn Clauses. In *Proceedings of EUROCAL'85, Linz, Austria*, volume 203 of *Lecture Notes in Computer Science*. Springer-Verlag, 1985.

[Pau96] L. C. Paulson. *ML for the Working Programmer*. Cambridge University Press, 2nd edition, 1996.

[Pla93] D. A. Plaisted. Equational reasoning and term rewriting systems. In D. Gabbay and J. Siekmann, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*. Oxford University Press, 1993.

[Plo72] G. Plotkin. Building-in equational theories. *Machine Intelligence*, 7:73–90, 1972.

[Poi87] A. Poigné. Partial algebras, subsorting and dependent types. In D. Sannella and A. Tarlecki, editors, *Recent Trends in Data Type Spcification: 5th Workshop on Specification of Abstract Data Types*, volume 332 of *Lecture Notes in Computer Science*, pages 208–234. Springer-Verlag, 1987.

[Poi90] A. Poigné. Parameterization for order-sorted algebra. *Journal of Computer and Systems Sciences*, 40:229–268, 1990.

[Poi91] A. Poigné. Once more on order-sorted algebras. In A. Tarlecki, editor, *Proceedings of the 16th International Symposium on Mathematical Foundations of Computer Science*, volume 520 of *Lecture Notes in Computer Science*, pages 397–405. Springer-Verlag, 1991.

[PS81] G. E. Peterson and M. E. Stickel. Complete sets of reductions for some equational theories. *Journal of the Association for Computing Machinery*, 28(2):233–264, 1981.

[PW78] M. S. Paterson and M. N. Wegman. Linear unification. *Journal of Computer and Systems Sciences*, 16:158–167, 1978.

[Rea89] C. M. P. Reade. *Elements of Functional Programming.* Addison Wesley, 1989.

[Rit89] M. Rittri. Using types as search keys in function libraries. In *Functional Programming Languages and Computer Architecture.* ACM Press, 1989. Revised version in Journal of Functional Programming, 1(1):71-89, 1991.

[Rit93] M. Rittri. Retrieving library functions by unifying types modulo linear isomorphism. *Theoretical Informatics and Applications*, 27(6):523–540, 1993.

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12:23–41, 1965.

[RS86] V. J. Rayward-Smith. *A First Course in Computability.* Blackwell Scientific Publications, 1986.

[SA93] R. Socher-Ambrosius. Unification in order-sorted logic with term declarations. In A. Voronkov, editor, *LPAR '93: 4th International Conference on Logic Programming and Automated Reasoning*, volume 698 of *Lecture Notes in Computer Science*, pages 301–308. Springer-Verlag, 1993.

[Sie89] J. Siekmann. Unification theory. *Journal of Symbolic Computation*, 7:207–274, 1989.

[Smo86] G. Smolka. Order-sorted horn logic: Semantics and deduction. Technical Report SEKI Report, SR-86-17, Universität Kaiserslautern, 1986.

[Smo88] G. Smolka. TEL (Version 0.9) Report and User Manual. Technical Report SEKI Report, SR-87-11, Universität Kaiserslautern, 1988.

[SNGM88] G. Smolka, W. Nutt, J. Goguen, and J. Meseguer. Order-sorted equational completion. In M. Nivat and H. Aït-Kaci, editors, *Resolution of Equations in Algebraic Structures.* Academic Press, 1988.

[Sny91] W. Snyder. *A Proof Theory for General Unification.* Birkhauser, 1991.

[Spi92] M. Spivey. *The Z Notation: A Reference Manual.* Prentice Hall International, 2nd edition, 1992.

[SS85] M. Schmidt-Schauss. Unification in a many-sorted calculus with declarations. In H. Stoyan, editor, *Proceedings of the 9th German Workshop on Artificial Intelligence*, pages 118–132, 1985.

[SS86] M. Schmidt-Schauss. Unification in many-sorted equational theories. In J.H. Siekmann, editor, *CADE'8: Proceedings of the 8th Conference on Automated Deduction*, volume 230 of *Lecture Notes in Computer Science*, pages 538–552. Springer-Verlag, 1986.

[SS89] M. Schmidt-Schauss. *Computational Aspects of an Order-Sorted Logic with Term Declarations.*, volume 395 of *Lecture Notes in Computer Science.* Springer-Verlag, 1989. Doctoral Dissertation, University of Kaiserslautern, 1988.

[ST86] D. Sannella and A. Tarlecki. Extended ML: an institution-independent framework for formal program development. In *Proceedings of the Workshop on Category Theory and Computer Programming*, volume 240 of *Lecture Notes in Computer Science.* Springer-Verlag, 1986. Also LFCS, University of Edinburgh, Report ECS-LFCS-86-16.

[Ste90] J. Steinbach. AC-termination of rewrite systems: A modified Knuth-Bendix ordering. In H. Kirchner and W. Wechler, editors, *Proceedings of the 2nd*

International Conference on Algebraic and Logic Programming, volume 463 of *Lecture Notes in Computer Science*, pages 372–386. Springer-Verlag, 1990.

[Sti81] M. E. Stickel. A unification algorithm for associative-commutative functions. *Journal of the Association for Computing Machinery*, 28(3):423–434, 1981.

[Toy87] Y. Toyama. Confluence term rewriting systems with membership constraints. In S. Kaplan and J-P. Jouannaud, editors, *Proceedings of the 1st International Workshop on Conditional Term Rewriting Systems*, volume 308 of *Lecture Notes in Computer Science*, pages 228–244. Springer-Verlag, 1987.

[Voi86] F. Voisin. CIGALE: A tool for interactive grammar construction and expression parsing. *Science of Computer Programming*, 7:61–86, 1986.

[Wal83] C. Walther. A many-sorted calculus based on resolution and paramodulation. In J. Kaufmann, editor, *IJCAI'83: Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 882–891, 1983.

[Wal89] U. Waldmann. Unification in order-sorted signatures. Technical Report 298, Universität Dortmund, 1989.

[Wal90] U. Waldmann. Compatibility of order-sored rewrite rules. In S. Kaplan and M. Okada, editors, *Proceedings of the 2nd International Workshop on Conditional Term Rewriting Systems*, volume 516 of *Lecture Notes in Computer Science*, pages 407–416. Springer-Verlag, 1990.

[Wal92] U. Waldmann. Semantics of order-sorted specifiations. *Theoretical Computer Science*, 94(1):1–35, 1992.

[WB89] P. Wadler and S. Blott. How to make *ad-hoc* polymorphism less ad hoc. In *Proceedings of the 16th Annual ACM Symposium on Principles of Programming Languages, Austin, Texas*, 1989.

[WB95] J. Walter and R. Bündgen. *The ReDuX User Guide (Version 1.5).* Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Germany, 1995.

[WD89] P. Watson and A. J. J. Dick. Least sorts in order-sorted term rewriting. Technical Report CSD-TR-606, Royal Holloway and Bedford New College, University of London., 1989.

[Wer93] A. Werner. A semantic approach to order-sorted rewriting. In C. Kirchner, editor, *Proceedings of the 5th International Conference on Rewriting Techniques and Applications*, volume 690 of *Lecture Notes in Computer Science*, pages 47–61. Springer-Verlag, 1993.

[Wir90] M. Wirsing. Algebraic specifications. In *Handbook of Theoretical Computer Science*, volume 2. Elsevier Science Publishers B.V., 1990.

[Wit92] L. With. Completeness and confluence of order-sorted term rewriting. In M. Rusinowitch and J-L. Rémy, editors, *Proceedings of the 3rd International Workshop on Conditional Rewriting Systems*, volume 656 of *Lecture Notes in Computer Science*, pages 393–407. Springer-Verlag, 1992.

[Yel85] K. Yelick. Combining unification algorithms for confined regular equational theories. In J-P. Jouannaud, editor, *Proceedings of the 1st International Conference on Rewriting Techniques and Applications*, volume 202 of *Lecture Notes in Computer Science*, pages 365–380. Springer-Verlag, 1985.

Index

The section number and page number of Definitions are given in this index.