UNIVERSITY
of
GLASGOW

Computing Science
M.Sc Thesis

A Precise Semantics for Ultraloose Specifications

Alastair D. Reid

Submitted for the degree of

Master of Science

Thesis
9846
Copy 1

Abstract

All formal specifiers face the danger of overspecification: accidentally writing an overly restrictive specification. This problem is particularly acute for axiomatic specifications because it is so easy to write axioms which hold for some of the intended implementations but not for all of them (or, rather, it is so *hard* not to write overly strong axioms).

One of the best developed ways of recovering some of those implementations which do not *literally* satisfy the specification is to apply a "behavioural abstraction operator" to a specification: adding in those implementations which have the same "behaviour" as an implementation which *does* satisfy the specification.

In two recent papers Wirsing and Broy propose an alternative (and apparently simpler) approach which they call "ultraloose specification." This approach is based on a particular style of writing axioms which avoids certain forms of overspecification.

An important, unanswered question is "How does the ultraloose approach relate to the other solutions?" The major achievement of this thesis is a proof that the ultraloose approach is semantically equivalent to the use of the "behavioural abstraction operator." This result is rather surprising in the light of a result by Schoett which *seems* to say that such a result is impossible.

Acknowledgements

I would like to thank the following people for their help during the period of this research.

- Dr. Muffy Thomas for acting as my supervisor.
- My office-mates Kei Davis and Shahad Ahmed for their support and advice.
- My parents for financial and other support.
- The Computing Science Department for their patience and generosity in providing facilities.
- The Science and Engineering Research Council for funding (award ♯88303611).
- Aran Lunzer for caffeine fixes and questions for all my LaTeX answers.

Contents

List of Figures

List of Theorems and Definitions

Chapter 1

Introduction

Two important requirements of a framework for formal program development are that it should allow any "legitimate" *informal* program development; and that it should be straightforward to prove that each step in a program development is allowed.

All programmers know that replacing one module by another module with the same external behaviour has no effect on the overall behaviour of a program and so, to satisfy the first requirement, any framework for formal program development should support the replacement of "behaviourally equivalent" modules (i.e. modules with the same "external behaviour"). That is, program specifications should have the following closure property.

> If a program module implements a specification then so should all behaviourally equivalent program modules.[1]

This thesis is concerned with different ways in which axiomatic (aka algebraic) specification languages achieve this closure property and with how these different approaches affect the ease of proving properties of the resulting specifications. In

[1] We are being deliberately vague about what we mean by "implements" and "behavioural equivalence." These terms are defined in chapters 2 and 3 respectively.

particular, we look at two closely-related languages due to Wirsing, Sannella and Tarlecki and to Wirsing and Broy.

The best developed axiomatic specification language which addresses the issue of behavioural closure is ASL proposed by Wirsing and Sannella [34,40] and developed further by Sannella and Tarlecki [27–30]. ASL is a kernel specification language used to define the semantics of higher-level specification languages such as PLUSS [6] and Extended ML [26].

Not all specifications written in ASL are behaviourally closed: for example, the specification of stacks of natural numbers in figure 1.1 admits *some* stack-like implementations such as the obvious list-based implementation but rejects others with the same behaviour such as the "array and pointer" implementation in figure 1.2.[2]

```
enrich Nat
by sign  Stack: type
         empty: → Stack
         push: Nat × Stack → Stack
         pop: Stack → Stack
         top: Stack → Nat
         isEmpty: Stack → Bool
   axioms ∀s: Stack, x: Nat. top(push(x, s)) = x
         ∀s: Stack, x: Nat. pop(push(x, s)) = s
         isEmpty(empty) = True
         ∀s: Stack, x: Nat. isEmpty(push(x, s)) = False
end
```

Figure 1.1: Stacks in ASL

There are two reasons why the stack specification in figure 1.1 is too strong and so fails to have the desired closure property.

1. The use of equations between stacks is too strong.

   This can be seen by considering the array and pointer implementation's failure to satisfy the second axiom.

[2]We use an *ad-hoc* but, we hope, clear notation to define the "implementation." In chapter 2 we will see that implementations should be defined in the same language that is used to write specifications.

```
type Stack = Pair(Int, Array of Nat)

empty = ⟨0, emptyArray⟩
push(x, ⟨i, a⟩) = ⟨i + 1, a[i]:= x⟩
pop(⟨i, a⟩) = ⟨i − 1, a⟩
top(⟨i, a⟩) = a[i]
isEmpty(⟨i, a⟩) = i == 0
```

Figure 1.2: A Stack Implementation

$pop(push(x, empty))$
$=$ { definition of $empty$ }
$pop(push(x, \langle 0, emptyArray\rangle))$
$=$ { definition of $push$ }
$pop(\langle 1, emptyArray[0] := x\rangle)$
$=$ { definition of pop }
$\langle 0, emptyArray[0] := x\rangle$
$\neq$
$\langle 0, emptyArray\rangle$
$=$ { definition of $empty$ }
$empty$

Although the array and pointer implementation does not satisfy this axiom, there is no "real" problem (from the programmer's point of view) because it is not possible to distinguish *pop*(*push*(*x*, *empty*)) from *empty* using the operations provided (i.e. *empty*, *push*, *pop* and *top*).

The problem with the specification is that is that it requires two values to be *identical* when it is sufficient for them to be *indistinguishable* (with respect to the operations provided).

2. The use of universal quantification is too strong.

This can be seen for the array and pointer implementation by considering the fourth axiom and instantiating *s* with the "nonsense" value $\langle -1, emptyArray\rangle$.

$\forall s\colon Stack, x\colon Nat.\ isEmpty(push(x, s)) = False$
$\Rightarrow$
$\forall x\colon Nat.\ isEmpty(push(x, \langle -1, emptyArray\rangle)) = False$
$=$ { definition of $push$ }
$\forall x\colon Nat.\ isEmpty(\langle 0, emptyArray[-1] := x\rangle)) = False$

$=$ { definition of *isEmpty* }
$\forall x{:}\, \mathit{Nat}.\ (0 == 0) = \mathit{False}$
$=$ { arithmetic, predicate calculus }
false

Again, the non-satisfaction of this axiom is not a "real" problem because "nonsense" values such as $\langle -1, \mathit{emptyArray}\rangle$ cannot be constructed using the operations provided and so will never arise during the running of a program.

The problem with the specification is that it requires a condition to hold for all values of type *Stack* when it is sufficient for the condition to hold only for values that can be constructed using the operations provided.

The solution to these problems adopted in ASL is to provide a "behavioural abstraction operator" which modifies the meaning of a specification *SP* by allowing any implementation which is behaviourally equivalent to an implementation of *SP*.

Wirsing and Broy's "ultraloose framework" [2,42] takes the alternative approach of trying to fix the problems with equations and quantification directly. The language (which we shall call USL) used in this framework is closely related to ASL (it shares four of ASL's five basic specification building operations.) It lacks ASL's behavioural abstraction operator but achieves a similar effect by allowing the use of slightly different notions of equality and quantification. The specification in figure 1.3 is a USL specification of a stack. There are two important differences:

1. To avoid the above problems with universal quantification, the ultraloose specification uses "reachable quantification" ($\forall^r$) which only ranges over the values which can be constructed using the available operations.

2. To avoid the above problems with equality, the ultraloose specification uses a congruence $\equiv$ instead of equality. (Since congruences are not "built in" to the specification language as equality is, it is necessary to add the last eight axioms specifying the reflexivity, symmetry, transitivity and substitutivity of $\equiv$.)

Unlike ASL, USL has not been extensively studied. The major contribution of this thesis are answers to the following questions:

```
enrich Nat
by sign  Stack: type
         empty: → Stack
         push: Nat × Stack → Stack
         pop: Stack → Stack
         top: Stack → Nat
         isEmpty: Stack → Bool
         ≡: Stack × Stack → Bool
   axioms ∀ʳs: Stack, x: Nat. top(push(x, s)) = x
          ∀ʳs: Stack, x: Nat. pop(push(x, s)) ≡ s
          isEmpty(s) = True
          ∀ʳs: Stack, x: Nat. isEmpty(push(x, s)) = False

          ∀s: Stack. s ≡ s
          ∀s1, s2: Stack. s1 ≡ s2 ⇔ s2 ≡ s1
          ∀s1, s2, s3: Stack. s1 ≡ s2 ∧ s2 ≡ s3 ⇒ s1 ≡ s3

          empty ≡ empty
          ∀s1, s2: Stack, x: Nat. s1 ≡ s2 ⇒ push(x, s1) ≡ push(x, s2)
          ∀s1, s2: Stack. s1 ≡ s2 ⇒ pop(s1) ≡ pop(s2)
          ∀s1, s2: Stack. s1 ≡ s2 ⇒ top(s1) = top(s2)
          ∀s1, s2: Stack. s1 ≡ s2 ⇒ isEmpty(s1) = isEmpty(s2)
end
```

Figure 1.3: Stacks in USL

- **Under what circumstances are USL specifications behaviourally closed?**

  We tackle this question by defining a transformation (the "ultraloose transformation") from ASL specifications such as that in figure 1.1 to USL specifications such as that in figure 1.3 and identifying sufficient conditions under which the transformed specification is behaviourally closed.

- There are two obvious ways of making the specification in figure 1.1 behaviourally closed: apply ASL's behavioural abstraction operation; or apply the ultraloose transformation mentioned above.

  Under what circumstances do these two approaches give the same result?

- **For which approach is it easiest to prove properties of the resulting specifications?**

  Since the ASL specification is shorter than the corresponding USL specification, one might think that the ASL specification is simpler than the USL specification; but, Wirsing and Broy claim that the behavioural abstraction operator is "mathematically difficult" and that their approach avoids these difficulties [42 paragraphs 4–5]. It is not immediately obvious which argument is correct.

  We tackle the question by comparing proofs for ASL and USL specifications.

Our interest in these results is twofold: they provide a basis on which to compare the approaches taken in ASL and in USL; and they provide useful results for use in proving properties of specifications and of specification transformations.

Related Work

The notion of behavioural equivalence can be traced back to Hoare's paper "Proof of Correctness of Data Representation" [13] which uses abstraction functions to describe the relationship between two modules. The use of functions rather than relations resulted in an asymmetric relation —that is, Hoare defined a behavioural ordering. Later work in the area of model-based formal program development (for example, [18,19]) generalised the abstraction function to a representation relation yielding an equivalence like that discussed in this thesis.

Early work on axiomatic specifications (in partciular, that of the influential ADJ group [9 section 5.5]) adopted a notion of implementation like that of Hoare. This has been developed further by (amongst others) Ehrig et al. [4] and is discussed in detail by Wirsing in [41].

One of the earliest uses of behavioural equivalence in the semantics of a specification language is that of Sannella and Wirsing discussed earlier (notable previous moves in this direction are those of Giarratana et al. [7] and of Wand [39]). Making the use of behavioural equivalence by inclusion of the behavioural abstraction operator in ASL allowed Sannella and Wirsing to adopt a notion of implementation which was very much simpler than that of the ADJ group and Ehrig et al. (this is perhaps the major technical innovation in ASL). The ASL language has subsequently been refined in a series of papers including [27–30].

Instead of explicitly including the behavioural abstraction operator in the language, several workers [8,11,12,17,20] have defined notions of "behavioural satisfaction" of axioms. Roughly, a model *behaviourally satisfies* an axiom iff there is a behaviourally equivalent model which satisfies (in the usual sense) that axiom. This approach (potentially) suffers from a major problem: behavioural satisfaction leads to strange results if we allow arbitrary first-order axioms.

For example, under the usual semantics the specification in figure 1.4 would be inconsistent (unimplementable) because the second and third axioms conflict but under a behavioural semantics based on this notion of behavioural satisfaction, this specification is consistent. (For example, the usual list-based implementation satisfies the first two axioms directly and behaviourally satisfies the third axiom since the behaviourally equivalent array and pointer-based implementation satisfies the third axiom.)

```
enrich Nat
by sign  Stack: type
         empty: → Stack
         push: Nat × Stack → Stack
         pop: Stack → Stack
         top: Stack → Nat
   axioms  ∀s: Stack, x: Nat. top(push(x, s)) = x
           ∀s: Stack, x: Nat. pop(push(x, s)) = s
           ∀s: Stack, x: Nat. pop(push(x, s)) ≠ s
end
```

Figure 1.4: Inconsistent Stacks in ASL

To avoid this problem, this approach (severely) restricts the form of axioms allowed in specifications to being conditional equations. That is, axioms must be of the form

$$\forall xs\colon \tau s.\ l1 = r1 \wedge \ldots lm = rm \Rightarrow l = r.$$

An early attempt to avoid the need for a radically different semantics is that of Maibaum, Sadler and Veloso [14,15] who used a direct encoding of Hoare's abstraction function. At first glance, their approach seems very complex since it uses infinitary logic suggesting that it would be hard to carry out finite proofs. However, their use of infinitary logic could have been replaced by use of the quantifier $\forall^r$ used in figure 1.3 for which we need only structural induction. The importance of this work is that using essentially the same simple notion of implementation as in ASL and USL, this approach allows broadly the same implementations as under the more complex semantics of the ADJ group. (We shall not attempt to give a more precise characterisation of the semantics here.)

Schoett's impossibility theorems [36,37] show that neither the usual language of first order logic with equality (as used in figure 1.1) nor Wirsing and Broy's logic (with $\forall^r$ instead of $\forall$) is powerful enough to precisely characterise a simple behaviourally closed class of algebras. This seems to suggest that something like ASL's behavioural abstraction operator is essential. However, as a corollary he showed that proving simple properties of modules using specifications written using the behavioural abstraction operator can require infinite proofs if one uses the proof technique suggested by Sannella and Tarlecki in [27]. This suggests that the goal of a simple behaviourally closed axiomatic specification language is unattainable.

Most algebraic specification languages provide a way to control which sorts and operations are exported from a specification. We shall show that, for such languages, Wirsing and Broy's logic is powerful enough to precisely characterise the class of all stack-like algebras (this is a corollary to our discussion of the relation between ASL and USL in chapter 3). (Since 1977 it has been known that allowing operations to be "hidden" by not exporting them greatly increases the power of specification languages (see, for example, [16,17]) so our result is perhaps not overly surprising. Indeed, in [37 section 5] Schoett suggests that operation hiding *may be* one way of avoiding the problem but does not show how it could be done. Our contribution is to confirm that operation hiding *can* be used to solve the problem and to provide a systematic method for doing so.)

Finally, it is worth remarking that Schoett's thesis [35] is the only work we know of which relates (any of) the above *theoretical* notions of behavioural equivalence to the modularisation facilities found in programming languages. Schoett introduces a concept he calls "stability (for behavioural equivalence)" (discussed further in chapter 5) and shows that if a programming language only provides "stable" modularisation facilities, then traditional Abstract Data Type theory is valid. That is, it is valid to replace an implementation of a module by any behaviourally equivalent module. Schoett is primarily concerned with programming languages and so his ideas do not directly apply to this thesis. Sannella and Tarlecki [32 section 6] discuss how the notion of stability can be applied to specification languages — we give a brief outline in chapter 5.

Organisation of this Thesis

The remainder of this chapter discusses various pieces of notation used throughout this thesis.

Specifications in both ASL and USL denote a class of algebras. Chapter 2 defines both languages and a satisfaction relation between algebras and specifications. This is used to define the implementation and equivalence relations between specifications.

Chapter 3 defines the major tool used in exploring the semantics of USL specifications: behavioural equivalence.

Chapter 4 explores two of the main themes of this thesis: behavioural closure of USL specifications and the relationship between USL and ASL.

Having shown that the ASL and the USL approaches to behavioural closure have the same result, chapter 5 demonstrates an advantage of USL over ASL: it can be easier to prove that a USL specification satisfies a given axiom than to show that the corresponding ASL specification satisfies the same axiom.

Chapter 6 concludes.

Notation

Our notation for the predicate calculus closely follows that of the Eindhoven School. That is:

Logical Operators $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$ denote negation, conjunction, disjunction, implication and equivalence respectively as usual.

$\Leftarrow$ pronounced "follows from," is defined by $P \Leftarrow Q \overset{\text{def}}{=} Q \Rightarrow P$.

In decreasing binding power we have $\neg$; $\wedge$ and $\vee$; $\Rightarrow$ and $\Leftarrow$; and $\Leftrightarrow$.

Format of Proofs Many of our proofs have the shape

$$\begin{array}{ll}
P & \\
\Leftrightarrow & \{ \text{ hint why } P \Leftrightarrow Q \ \} \\
Q & \\
\Rightarrow & \{ \text{ hint why } Q \Rightarrow R \ \} \\
R & \\
\vdots &
\end{array}$$

This is used as a shorthand for $P \Leftrightarrow Q \wedge Q \Rightarrow R \wedge \ldots$.

Quantifiers The general pattern for a quantified expression is

$$(Q\ xs : P(xs) : F(xs))$$

with Q a quantifier, xs a list of variables, $P(xs)$ a predicate in terms of the variables (the range) and $F(xs)$ the term of the quantification. ($F(xs)$ should be defined for all xs that satisfy $P(xs)$.) (For sets, the notation $\{ x : P(x) : F(x) \}$ is used as an abbreviation for $(\cup x : P(x) : \{F(x)\})$.)

The following table gives a few examples in "conventional" notation and in the notation used in this report.

$\bigcup_{i \in I} A_i \cup B_i$	$(\cup i : i \in I : A_i \cup B_i)$
$\bigcap_{a \sqsubseteq x \sqsubseteq b} F(x)$	$(\cap x : a \sqsubseteq x \sqsubseteq b : F(x))$

$$\{\, y \mid \exists x \in \mathrm{dom}(f).\ f(x) = y \,\} \qquad \{\, x : x \in \mathrm{dom}(f) : f(x) \,\}$$

One advantage of this notation that it eliminates any ambiguity as to which variables are being quantified over (as shown in the second example).

In this thesis we make extensive use of the notion of the (downward) closure of a set.

Definition 1.1 (closure)

Let A be a set and $\leq : A \leftrightarrow A$ a reflexive transitive relation on A.

The *downward closure of a subset A' of A with respect to $\leq$* (written $\mathbf{Cl}_{\leq}(A')$) is defined by

$$\mathbf{Cl}_{\leq}(A') \overset{\text{def}}{=} \{a, a' \colon a \in A \wedge a' \in A' \wedge a \leq a' \colon a\}$$

A subset A' of A is said to be *downward closed with respect to* $\leq$ if $\mathrm{Cl}_{\leq}(A') = A'$.

In the common case that $\leq$ is an equivalence, we drop the word "downward" — that is we refer to $\mathbf{Cl}_{\leq}(A')$ as "the closure of A' with respect to $\leq$" and say that A' is "closed with respect to $\leq$ if $\mathbf{Cl}_{\leq}(A') = A'$."

End Definition.

Much of our other notation is taken from the Z specification language. (For example, the image of a set X under a function f is $f(\!|X|\!) \overset{\text{def}}{=} \{x \colon x \in X \colon f(x)\}$.)

Other notation will be introduced as the need arises.

Chapter 2

The Semantics of ASL and USL

This chapter defines the language and semantics of ASL and USL. Given the similarity between the two languages it is convenient to define the semantics of the "union" of the the languages and define ASL and USL as sublanguages.

In both languages, the simplest and most fundamental form of specification consists of a signature (which names the types and operations defined by the specification) and a set of axioms. For example,

$$
\begin{array}{lll}
\textbf{spec} & \textbf{sign} & \mathit{Bool}\ \textbf{:type} \\
 & & \text{True}, \text{False} :\rightarrow \mathit{Bool} \\
 & \textbf{axioms} & \text{True} \neq \text{False} \\
 & & \quad \forall x{:}\,\mathit{Bool}.\ x = \text{True} \vee x = \text{False} \\
\textbf{end} & &
\end{array}
$$

The semantics of such specifications is the class of algebras satisfying the axioms. Section 2.1 defines signatures, algebras, axioms and related concepts. Those familiar with the semantics of axiomatic specifications will be able to skim everything except the definition of axioms.

Section 2.2 defines the specification building operations used to construct large, structured specifications from these components and defines the notion of implementation used in ASL and USL. Again, those familiar with ASL will be able to skim this section.

Finally, section 2.3 defines the sublanguages corresponding to ASL and USL.

2.1 Signatures, Algebras and Axioms

This section defines the mathematical structures used to define two important aspects of ASL and USL: the syntax and the semantics.

The syntactic aspects of ASL and USL are signatures (which name the types and functions in a specification), signature morphisms (functions between signatures), terms (expressions) and axioms.

The semantics aspects of ASL and USL are algebras (which define interpretations of the types and functions in a signature). Algebras are used to give a meaning to terms (by defining a notion of evaluation of a term) and to axioms (by defining a satisfaction relation between algebras and axioms).

2.1.1 Signatures and Algebras

In essence a signature is a set of symbols with an additional (monomorphic, first-order) type structure. The definition of this "set with structure" is as follows.

Definition 2.1 (signatures)

A "signature" is a triple

$$\Sigma = \langle T, F, \alpha : F \to [T] \times T \rangle$$

where T and F are disjoint sets containing the "sort symbols" and the "function symbols" of Σ respectively.[1]

For $f \in F$, the "type" of f in Σ is $\alpha(f)$; and if $\alpha(f) = \langle [\tau 1, \ldots \tau m], \tau \rangle$, we write $f : \tau 1 \times \cdots \tau m \to \tau$ in Σ. We define $\mathbf{Tp}(\langle T, F, \alpha \rangle) \overset{\text{def}}{=} T$ and $\mathbf{Op}(\langle T, F, \alpha \rangle) \overset{\text{def}}{=} F$.

We write **Sign** to denote the class of all signatures and write Σ: **Sign** to indicate that Σ is a signature.

End Definition.

[1] Our notation for lists is based on the functional programming language Haskell: $[A]$ denotes the set of lists of A; $[a1, \ldots am]$ denotes the list of length m with elements $a1, \ldots am$; $as \mathbin{+\!\!+} bs$ denotes the concatenation of the lists as and bs; and $\sharp as$ denotes the length of the list as.

[Note: In the literature, F and α are often replaced by a $[T] \times T$-indexed set of function symbols which may be (and occasionally is) used to express "overloading" of function symbols. e.g. $+\colon Nat \times Nat \rightarrow Nat$ and $+\colon Int \times Int \rightarrow Int$ could appear in the same signature. Our definitions resemble those of Wirsing and Schoett: in [41], Wirsing defines a signature as a pair $\langle T, F\rangle$ but leaves α implicit; in [37], Schoett defines a signature as a pair $\langle T, \alpha\rangle$ but leaves F implicit.]

For example, the following is a typical signature for a stack.

$$
\begin{array}{lll}
\mathit{StackSig} \overset{\text{def}}{=} & \langle & \{\mathit{Nat}, \mathit{Stack}\}, \\
& & \{\mathit{0}, \mathit{succ}, \mathit{empty}, \mathit{push}, \mathit{pop}, \mathit{top}\}, \\
& & \lambda f. \begin{cases}
\langle [], \mathit{Nat}\rangle, & \text{if } f = \mathit{0}; \\
\langle [\mathit{Nat}], \mathit{Nat}\rangle, & \text{if } f = \mathit{succ}; \\
\langle [], \mathit{Stack}\rangle, & \text{if } f = \mathit{empty}; \\
\langle [\mathit{Nat}, \mathit{Stack}], \mathit{Stack}\rangle, & \text{if } f = \mathit{push}; \\
\langle [\mathit{Stack}], \mathit{Stack}\rangle, & \text{if } f = \mathit{pop}; \\
\langle [\mathit{Stack}], \mathit{Nat}\rangle, & \text{if } f = \mathit{top}.
\end{cases} \\
& \rangle &
\end{array}
$$

This notation is a bit unwieldy and so we usually use the following more readable notation instead.

$$
\begin{array}{lll}
\mathit{StackSig} \overset{\text{def}}{=} & \textbf{sign} & \mathit{Nat}, \mathit{Stack}\colon \textbf{type} \\
& & \mathit{0}\colon \rightarrow \mathit{Nat} \\
& & \mathit{succ}\colon \mathit{Nat} \rightarrow \mathit{Nat} \\
& & \mathit{empty}\colon \rightarrow \mathit{Stack} \\
& & \mathit{push}\colon \mathit{Nat} \times \mathit{Stack} \rightarrow \mathit{Stack} \\
& & \mathit{pop}\colon \mathit{Stack} \rightarrow \mathit{Stack} \\
& & \mathit{top}\colon \mathit{Stack} \rightarrow \mathit{Nat} \\
& \textbf{end} &
\end{array}
$$

Signature morphisms are functions between signatures which respect the type structure.

Definition 2.2 (signature morphisms)

Let $\Sigma = \langle T, F, \alpha\rangle$ and $\Sigma' = \langle T', F', \alpha'\rangle$ be signatures.

A signature morphism σ from Σ to Σ' (written $\sigma: \Sigma \to \Sigma'$) is a function of type $(T \cup F) \to (T' \cup F')$ such that $\sigma|_T: T \to T'$, $\sigma|_F: F \to F'$ and, for each $f: \tau 1 \times \cdots \tau m \to \tau$ in Σ, $\sigma(f): \sigma(\tau 1) \times \cdots \sigma(\tau m) \to \sigma(\tau)$ in Σ'.[2]

The signature Σ is a subsignature of Σ' (written $\Sigma \subseteq \Sigma'$) if $T \subseteq T'$, $F \subseteq F'$ and $\alpha = \alpha'|_F$.

A signature morphism $\sigma: \Sigma \to \Sigma'$ is said to be an inclusion (written $\sigma: \Sigma \hookrightarrow \Sigma'$) if $\Sigma \subseteq \Sigma'$ and $\sigma = id_{T \cup F}$.[3]

Where Σ is obvious from context and $\Sigma' \subseteq \Sigma$, we sometimes use the set $T' \cup F'$ to denote Σ'.

End Definition.

In essence an algebra is an abstraction of a program module: it is a function mapping symbols in a signature to their interpretation (either a set of values or a function). Algebras abstract away from details like the execution time or space of a function: this reflects the emphasis of formal methods on correctness rather than efficiency.

Definition 2.3 (algebras)

Let $\Sigma = \langle T, F, \alpha \rangle$ be a signature.

A Σ-algebra $\mathcal{A}$ is a $T \cup F$-indexed family such that for each $\tau \in T$, $\mathcal{A}_\tau$ is a set (the "carrier of τ") and for each $f: \tau 1 \times \cdots \tau m \to \tau$ in Σ, $\mathcal{A}_f$ is a (total) function of type

$$\mathcal{A}_f: \mathcal{A}_{\tau 1} \times \cdots \mathcal{A}_{\tau m} \to \mathcal{A}_\tau$$

If $\mathcal{A}$ and $\mathcal{B}$ are Σ-algebras, $\mathcal{A}$ is a *subalgebra* of $\mathcal{B}$ (written $\mathcal{A} \subseteq \mathcal{B}$) if, for each sort $\tau \in T$, $\mathcal{A}_\tau \subseteq \mathcal{B}_\tau$ and, for each function symbol $f: \tau 1 \times \tau m \to \tau$ in Σ and each $a1 \in \mathcal{A}_{\tau 1}, \ldots am \in \mathcal{A}_{\tau m}$, $\mathcal{A}_f(a1, \ldots am) = \mathcal{B}_f(a1, \ldots am)$.

[2]The notation $h|_{X'}$ denotes the restriction of a function $h: X \to Y$ to a subset X' of its domain X. That is, $h|_{X'}(x') \overset{\text{def}}{=} h(x')$ for $x' \in X'$.

[3]The notation id_X denotes the identity function over the set X defined by $id_X(x) \overset{\text{def}}{=} x$ for $x \in X$.

The class of all Σ-algebras is denoted by $\mathbf{Alg}(\Sigma)$.

End Definition.

[Our definition of an algebra is essentially the same as that of Schoett [37]. Other authors such as Ehrig and Mahr [5] use two functions $S_{\mathcal{A}}$ and $OP_{\mathcal{A}}$ (respectively) to assign interpretations to sort and function symbols (respectively) instead of a single family $\mathcal{A}$.]

For example, the following is a *StackSig*-algebra called *stack*.

$$\begin{array}{ll}
\mathit{stack}_{\mathit{Nat}} & \stackrel{\text{def}}{=} \{\mathit{0}, \mathit{1}, \mathit{2}, \ldots\} \\
\mathit{stack}_{\mathit{Stack}} & \stackrel{\text{def}}{=} [\{\mathit{0}, \mathit{1}, \mathit{2}, \ldots\}] \\
\mathit{stack}_{\mathit{0}} & \stackrel{\text{def}}{=} \mathit{0} \\
\mathit{stack}_{\mathit{succ}} & \stackrel{\text{def}}{=} \lambda x.\ x + \mathit{1} \\
\mathit{stack}_{\mathit{empty}} & \stackrel{\text{def}}{=} [\,] \\
\mathit{stack}_{\mathit{push}} & \stackrel{\text{def}}{=} \lambda x, s.\ [x] +\!\!+\ s \\
\mathit{stack}_{\mathit{pop}} & \stackrel{\text{def}}{=} \lambda s.\ \mathit{if}\ s = [\,]\ \mathit{then}\ [\,]\ \mathit{else}\ \mathit{tail}(s) \\
\mathit{stack}_{\mathit{top}} & \stackrel{\text{def}}{=} \lambda s.\ \mathit{if}\ s = [\,]\ \mathit{then}\ \mathit{0}\ \mathit{else}\ \mathit{head}(s)
\end{array}$$

This notation is a bit unwieldy and so we usually use the following more readable notation instead.

$$\begin{array}{lll}
\mathit{stack} \stackrel{\text{def}}{=} & \langle & \mathit{Nat} = \{\mathit{0}, \mathit{1}, \mathit{2}, \ldots\} \\
& & \mathit{Stack} = [\mathit{Nat}] \\
& & \mathit{0} = \mathit{0} \\
& & \mathit{succ}(x) = x + \mathit{1} \\
& & \mathit{empty} = [\,] \\
& & \mathit{push}(x, s) = [x] +\!\!+\ s \\
& & \mathit{pop}(s) = \mathit{if}\ s = [\,]\ \mathit{then}\ [\,]\ \mathit{else}\ \mathit{tail}(s) \\
& & \mathit{top}(s) = \mathit{if}\ s = [\,]\ \mathit{then}\ \mathit{0}\ \mathit{else}\ \mathit{head}(s) \\
& \rangle &
\end{array}$$

One of the most useful operations on an algebra is to compose it with a signature morphism and so rename, copy or hide some of the interpretations of the symbols in the algebra.

Definition 2.4 (reducts)

Let Σ and Σ' be signatures, $\sigma\colon \Sigma' \to \Sigma$ a signature morphism and $\mathcal{A}$ a Σ-algebra.

The "σ-reduct of $\mathcal{A}$" (written $\mathcal{A}|_\sigma$) is the Σ'-algebra defined by

$$\mathcal{A}|_\sigma \overset{\text{def}}{=} \mathcal{A} \cdot \sigma$$

If σ is an inclusion, and $\mathcal{B} = \mathcal{A}|_\sigma$, $\mathcal{A}$ is an *extension* of $\mathcal{B}$.

End Definition.

We note that if σ is an inclusion, then $\mathcal{A}|_\sigma$ is the algebra obtained by restricting the domain of $\mathcal{A}$ to the sort and function symbols named in Σ' (hence the choice of notation). Where σ is obvious from context and an inclusion, we write $\mathcal{A}|_{\Sigma'}$ instead of $\mathcal{A}|_\sigma$.

A homomorphism can be thought of as a "representation function" describing how values in one algebra may be represented by values in another algebra.

Definition 2.5 (homomorphisms and isomorphisms)

Let Σ be a signature with sorts T and let $\mathcal{A}$ and $\mathcal{B}$ be Σ-algebras.

A total T-indexed function $h : \mathcal{A}|_T \to \mathcal{B}|_T$ is a Σ-homomorphism if, for each $f\colon \tau 1 \times \cdots \tau m \to \tau$ in Σ and values $a1 \in \mathcal{A}_{\tau 1}, \ldots am \in \mathcal{A}_{\tau m}$,

$$h_\tau(A_f(a1, \ldots am)) = \mathcal{B}_f(h_{\tau 1}(a1), \ldots h_{\tau m}(am))$$

If $h\colon \mathcal{A} \to \mathcal{B}$ and $h'\colon \mathcal{B} \to \mathcal{A}$ are Σ-homomorphisms such that $h' \cdot h = id_{\mathcal{A}}$ and $h \cdot h' = id_{\mathcal{B}}$ then both h and h' are said to be Σ-isomorphisms (written $h\colon \mathcal{A} \cong \mathcal{B}$ or just $\mathcal{A} \cong \mathcal{B}$.)

End Definition.

[Notes: Since $\mathcal{A}$ is a family (i.e. a function), $\mathcal{A}|_T$ denotes the T-indexed set of "carriers" of the sorts T. Thus, a homomorphism relates the *values* in one algebra to the *values* in another.

The condition

$$h_\tau(\mathcal{A}_f(a1, \ldots am)) = \mathcal{B}_f(h_{\tau 1}(a1), \ldots h_{\tau m}(am))$$

is known as "the homomorphism condition."]

The following result is standard (see, for example, [5 section 3.1]):

Lemma 2.6 (bijectivity and uniqueness of isomorphisms)

Let Σ be a signature and $\mathcal{A}$ and $\mathcal{B}$ two Σ-algebras.

If $h{:}\,\mathcal{A} \rightarrow \mathcal{B}$ is a Σ-isomorphism, then h is bijective; and there is exactly one Σ-isomorphism $h'{:}\,\mathcal{B} \rightarrow \mathcal{A}$ such that $h' \cdot h = id_{\mathcal{A}}$ and $h \cdot h' = id_{\mathcal{B}}$.

End Lemma.

It is easily seen that reducts preserve isomorphisms (that is: $\mathcal{A} \cong \mathcal{B} \Rightarrow \mathcal{A}|_\sigma \cong \mathcal{B}|_\sigma$). It has been remarked (see, for example [28 section 5] that reducts need not reflect isomorphisms (that is: $\mathcal{A}|_\sigma \cong \mathcal{B}|_\sigma \not\Rightarrow \mathcal{A} \cong \mathcal{B}$).

Counterexample 2.7 ($\mathcal{A}|_{\Sigma'} \cong \mathcal{B}|_{\Sigma'} \not\Rightarrow \mathcal{A} \cong \mathcal{B}$)

Let $\Sigma \overset{\text{def}}{=}$ **sign** *Bool* :**type** , True, False :→ *Bool* **end**, $\Sigma' \overset{\text{def}}{=}$ **sign** *Bool* :**type** **end** and let the Σ-algebras $\mathcal{A}$ and $\mathcal{B}$ be defined by

$$\mathcal{A} \overset{\text{def}}{=} \langle \begin{array}{l} Bool \overset{\text{def}}{=} \{0, 1\} \\ \text{True} \overset{\text{def}}{=} 1 \\ \text{False} \overset{\text{def}}{=} 0 \end{array} \rangle \quad \text{and} \quad \mathcal{B} \overset{\text{def}}{=} \langle \begin{array}{l} Bool \overset{\text{def}}{=} \{0, 1\} \\ \text{True} \overset{\text{def}}{=} 1 \\ \text{False} \overset{\text{def}}{=} 1 \end{array} \rangle$$

It is clear that $\mathcal{A}|_{\Sigma'} \cong \mathcal{B}|_{\Sigma'}$ (since $\mathcal{A}|_{\Sigma'} = \mathcal{B}|_{\Sigma'}$) but $\mathcal{A} \ncong \mathcal{B}$. Hence,

$$\mathcal{A}|_{\Sigma'} \cong \mathcal{B}|_{\Sigma'} \not\Rightarrow \mathcal{A} \cong \mathcal{B}$$

End Counterexample.

Definition 2.8 (congruences and quotients)

Let Σ be a signature with sorts T, let $\mathcal{A}$ be a Σ-algebra.

If $\equiv$ is a T-indexed equivalence over $\mathcal{A}$ (that is, for each $\tau \in T$, $\equiv_\tau\colon \mathcal{A}_\tau \leftrightarrow \mathcal{A}_\tau$ is an equivalence) and, for each function symbol $f\colon \tau 1 \times \cdots \tau m \to \tau$ in Σ and elements $a1, a1' \in \mathcal{A}_{\tau 1}, \ldots am, am' \in \mathcal{A}_{\tau m}$,

$$a1 \equiv_{\tau 1} a1' \wedge \ldots am \equiv_{\tau m} am' \;\Rightarrow\; \mathcal{A}_f(a1, \ldots am) \equiv_\tau \mathcal{A}_f(a1', \ldots am')$$

then we say that $\equiv$ *is a Σ-congruence over $\mathcal{A}$.*

If $\equiv$ is a Σ-congruence relation over $\mathcal{A}$, the quotient algebra $\mathcal{A}/_\equiv$ is defined for each sort $\tau \in T$ by[4]

$$(\mathcal{A}/_\equiv)_\tau \overset{\text{def}}{=} \{a\colon a \in \mathcal{A}_\tau\colon [\![a]\!]_{\equiv_\tau}\}$$

and for each function symbol $f\colon \tau 1 \times \cdots \tau m \to \tau$ by

$$(\mathcal{A}/_\equiv)_f([\![a1]\!]_{\equiv_{\tau 1}}, \ldots [\![am]\!]_{\equiv_{\tau m}}) \overset{\text{def}}{=} [\![\mathcal{A}_f(a1, \ldots am)]\!]_{\equiv_\tau}$$

End Definition.

It is well known (see, for example, [5 section 3.13]) that there is a surjective homomorphism from an algebra $\mathcal{A}$ to any quotient of $\mathcal{A}$. (This fact is used in the discussion of behavioural equivalence in chapter 3.)

Lemma 2.9 (homomorphism to quotient algebras)

Let Σ be a signature with sorts T, $\mathcal{A}$ a Σ-algebra, and $\equiv$ a Σ-congruence over $\mathcal{A}$.

The T-indexed function $[\![\_]\!]_\equiv$ defined for each $\tau \in T$ by $([\![\_]\!]_\equiv)_\tau \overset{\text{def}}{=} [\![\_]\!]_{(\equiv_\tau)}$ is a surjective Σ-homomorphism from $\mathcal{A}$ to $\mathcal{A}/_\equiv$.

Proof The homomorphism condition follows immediately from the definition of $(\mathcal{A}/_\equiv)_f$. Surjectivity of $[\![\_]\!]_\equiv$ follows from the definition of $(\mathcal{A}/_\equiv)_\tau$.

End Lemma.

[4] For any equivalence relation $\equiv\colon A \leftrightarrow A$, the *equivalence class* $[\![a]\!]_\equiv$ of an element $a \in A$ is the set of all values equivalent to a. That is, $[\![a]\!]_\equiv \overset{\text{def}}{=} \{a'\colon a' \in A \wedge a \equiv a'\colon a'\}$.

2.1.2 Terms, Derived Operators and Reachability

This section defines terms, interpretations and derived operators.

In essence, a $\Sigma(X)$-term is an expression constructed using the function symbols in a signature Σ and a set of variable symbols X. Throughout this thesis, we use X to denote an infinite indexed set of variable symbols such that X_τ and $X_{\tau'}$ are disjoint if $\tau \neq \tau'$. We say x has sort τ (written $x{:}\,\tau$) if $x \in X_\tau$.

Definition 2.10 (terms)

Let Σ be a signature with sorts T and X a T-indexed set of variables.

The T-indexed set $W(\Sigma, X)$ of finite Σ-terms with variables X is the least T-indexed set (with respect to $\subseteq$) such that:[5]

$$\begin{array}{ll} x \in W(\Sigma,X)_\tau & \text{if } x \in X_\tau \\ f(ts) \in W(\Sigma,X)_\tau & \text{if } \tau s \in [T],\ f{:}\,\tau s \to \tau \text{ and } ts \in W(\Sigma,X)_{\tau s} \end{array}$$

We say that t is a "$\Sigma(X)$-term" (or just "Σ-term") if $t \in W(\Sigma, X)$.

The set of variables used in a term t (written $\mathbf{vars}(t)$) is defined by

$$\begin{array}{ll} \mathbf{vars}(x) & \stackrel{\text{def}}{=} \{x\} \\ \mathbf{vars}(f(t1, \ldots tm)) & \stackrel{\text{def}}{=} \mathbf{vars}(t1) \cup \ldots \mathbf{vars}(tm) \end{array}$$

We say that a term t is "ground" (or that t is a "ground term") if $\mathbf{vars}(t) = \emptyset$.

The T-indexed set of variables used in a term t (written $\mathbf{Vars}(t)$) is defined for each $\tau \in T$ by $\mathbf{Vars}_\tau(t) \stackrel{\text{def}}{=} X_\tau \cap \mathbf{vars}(t)$.

We say that a Σ-term t has sort τ (written $t{:}\,\tau$) if $t \in W(\Sigma, X)_\tau$. This is extended to lists and tuples of terms in the obvious way. That is,

$$t1, \ldots tm{:}\,\tau 1, \ldots \tau m \stackrel{\text{def}}{=} t1{:}\,\tau 1, \ldots tm{:}\,\tau m$$

End Definition.

[5]The notation $[a1, \ldots am] \in A_{[i1, \ldots in]}$ (where $i1, \ldots in \in I$ and A is an I-indexed set) is an abbreviation for $m = n \wedge a1 \in A_{i1} \wedge \ldots am \in A_{in}$.

For example, $\mathit{empty}()\colon \mathit{Stack}$ and $\mathit{top}(\mathit{push}(x, \mathit{empty}()))\colon \mathit{Nat}$. Where obvious from context, we drop the redundant "()" after constant operators. For example, we write *empty* and $\mathit{top}(\mathit{push}(x, \mathit{empty}))$ for the above terms.

Definition 2.11 (valuations and interpretation)

Let Σ be a signature with sorts T, $\mathcal{A}$ a Σ-algebra.

A valuation is any partial T-indexed function $v\colon X \nrightarrow \mathcal{A}$ (it "assigns" values to variables).[6] For any set of variables $x1 \in X_{\tau 1}, \ldots xm \in X_{\tau m}$ and values $a1 \in \mathcal{A}_{\tau 1}$, $\ldots am \in \mathcal{A}_{\tau m}$ we write $\{x1 := a1, \ldots xm := am\}$ to denote the least valuation which, for each $i \in \{1, \ldots m\}$ assigns the value ai to the variable xi. That is, for each $i \in \{1, \ldots m\}$,

$$\{x1 := a1, \ldots xm := am\}_{\tau i}(xi) \overset{\mathrm{def}}{=} ai$$

Let t be a $\Sigma(X)$-term and $v\colon X \nrightarrow \mathcal{A}$ a valuation such that the value of $v_\tau(x)$ is defined for each $x \in \mathbf{Vars}(t)_\tau$ (and possibly undefined otherwise). The value (or "interpretation") of t in $\mathcal{A}$ under v (written $t_{\mathcal{A}}(v)$) is inductively defined by:

$$\begin{aligned} x_{\mathcal{A}}(v) &\overset{\mathrm{def}}{=} v(x) \\ f(t1, \ldots tm)_{\mathcal{A}}(v) &\overset{\mathrm{def}}{=} \mathcal{A}_f(t1_{\mathcal{A}}(v), \ldots tm_{\mathcal{A}}(v)) \end{aligned}$$

If $\mathbf{vars}(t) = \emptyset$, the value of $t_{\mathcal{A}}(v)$ is independent of v and so we define

$$t_{\mathcal{A}} \overset{\mathrm{def}}{=} t_{\mathcal{A}}(\{\})$$

where $\{\}$ denotes the completely undefined valuation.

To let us emphasize that a function $v\colon X \nrightarrow \mathcal{A}$ is a valuation, we define the set $\mathbf{Val}(\mathcal{A})$ to be the set of all partial T-indexed functions $v\colon X \nrightarrow \mathcal{A}$ and the set $\mathbf{Val}(\mathcal{A}, t)$ to be the set of all partial T-indexed functions $v\colon X \nrightarrow \mathcal{A}$ such that $v_\tau(x)$ is defined for each $x \in \mathbf{Vars}(t)$.

End Definition.

[6] Partial functions are used to avoid the problem that, if any carrier of an algebra is empty, there is no total T-indexed function $v\colon X \to \mathcal{A}|_T$. This solution is based on that used by Schoett in [37].

The following property of homomorphisms is used in chapter 3:

Lemma 2.12 (representation of terms)

Let Σ be a signature with sorts T, X a T-indexed set of variables, $\mathcal{A}$ and $\mathcal{B}$, Σ-algebras and $h: \mathcal{A} \to \mathcal{B}$ a Σ-homomorphism.

Then, for any $\Sigma(X)$-term t and $v \in \mathbf{Val}(\mathcal{A}, t)$ a valuation.

$$h(t_{\mathcal{A}}(v)) = t_{\mathcal{B}}(h \cdot v)$$

Proof

The proof is by induction over the structure of t.

Base case $(t \stackrel{\text{def}}{=} x)$

$h(x_{\mathcal{A}}(v))$
$=$ { definition of $t_{\mathcal{A}}(v)$ }
$h(v(x))$
$=$ { definition of composition }
$(h \cdot v)(x)$
$=$ { definition of $t_{\mathcal{B}}(v)$ }
$x_{\mathcal{B}}(h \cdot v)$
□

Inductive step $(t \stackrel{\text{def}}{=} f(t1, \ldots tm))$

Assume that $h(t1_{\mathcal{A}}(v)) = t1_{\mathcal{B}}(h \cdot v), \ldots h(tm_{\mathcal{A}}(v)) = tm_{\mathcal{B}}(h \cdot v)$.

$h(f(t1, \ldots tm)_{\mathcal{A}}(v))$
$=$ { definition of $t_{\mathcal{A}}(v)$ }
$h(\mathcal{A}_f(t1_{\mathcal{A}}(v), \ldots tm_{\mathcal{A}}(v)))$
$=$ { homomorphism condition }
$\mathcal{B}_f(h(t1_{\mathcal{A}}(v)), \ldots h(tm_{\mathcal{A}}(v)))$
$=$ { ind. assumption: $h(t1_{\mathcal{A}}(v)) = t1_{\mathcal{B}}(h \cdot v), \ldots h(tm_{\mathcal{A}}(v)) = tm_{\mathcal{B}}(h \cdot v)$ }
$\mathcal{B}_f(t1_{\mathcal{B}}(h \cdot v), \ldots tm_{\mathcal{B}}(h \cdot v))$
$=$ { definition of $t_{\mathcal{B}}(v)$ }
$f(t1, \ldots tm)_{\mathcal{B}}(h \cdot v)$
□

So, $h(x_{\mathcal{A}}(v)) = x_{\mathcal{B}}(h \cdot v)$ and, if $h(t1_{\mathcal{A}}(v)) = t1_{\mathcal{B}}(h \cdot v), \ldots h(tm_{\mathcal{A}}(v)) = tm_{\mathcal{B}}(h \cdot v)$, then $h(f(t1, \ldots tm)_{\mathcal{A}}(v)) = f(t1, \ldots tm)_{\mathcal{B}}(h \cdot v)$. Thus, by the principle of structural induction, $h(t_{\mathcal{A}}(v)) = t_{\mathcal{B}}(h \cdot v)$.

End Lemma.

An element $a \in \mathcal{A}_\tau$ is reachable if a can be constructed using the operations named in Σ. That is, if for some $t \in W(\Sigma, \emptyset)_\tau$, $t_\mathcal{A} = a$.

More generally, for some subsignature Σ' of Σ and subset T' of the sorts of Σ, a is $\Sigma'(T')$-reachable if a can be constructed using the operations named in Σ' and the values in $\mathcal{A}|_{T'}$. More formally,

Definition 2.13 (reachability, reachable subalgebras)

Let Σ be a signature with sort symbols T, Σ' a subsignature of Σ, T' a subset of T and $\mathcal{A}$ a Σ-algebra.

Let X' be the T-indexed set of variables defined for each $\tau \in T$ by

$$X'_\tau \overset{\text{def}}{=} \begin{cases} X_\tau, & \text{if } \tau \in T'; \text{ and} \\ \emptyset, & \text{otherwise.} \end{cases}$$

For each sort symbol $\tau \in T$ and value $a \in \mathcal{A}_\tau$, we say that a is $\Sigma'(T')$-reachable if $\mathcal{R}(\Sigma', T', a)$ where

$$\mathcal{R}(\Sigma', T', a) \overset{\text{def}}{=} (\exists t, v : t \in W(\Sigma', X')_\tau \wedge v \in \mathbf{Val}(\mathcal{A}, t) : t_\mathcal{A}(v) = a)$$

The Σ'-algebra $\mathcal{R}(\Sigma', T', \mathcal{A})$ is defined for each sort symbol $\tau \in \Sigma'$ by

$$\mathcal{R}(\Sigma', T', \mathcal{A})_\tau \overset{\text{def}}{=} \{a \colon a \in \mathcal{A}_\tau \wedge \mathcal{R}(\Sigma', T', a) \colon a\}$$

and for each function symbol $f \colon \tau 1 \times \cdots \tau m \to \tau$ in Σ' by

$$\mathcal{R}(\Sigma', T', \mathcal{A})_f \overset{\text{def}}{=} (\mathcal{A}_f)|_{\mathcal{R}(\Sigma', T', \mathcal{A})_{\tau 1} \times \cdots \mathcal{R}(\Sigma', T', \mathcal{A})_{\tau m}}$$

Let $\mathcal{B}$ be a Σ-algebra and $h \colon \mathcal{A} \to \mathcal{B}$ a Σ-homomorphism. The homomorphism $\mathcal{R}(\Sigma', T', h) \colon \mathcal{R}(\Sigma', T', \mathcal{A}) \to \mathcal{R}(\Sigma', T', \mathcal{B})$ is defined for each sort $\tau \in T$ by

$$\mathcal{R}(\Sigma', T', h)_\tau \overset{\text{def}}{=} h|_{\mathcal{R}(\Sigma', T', \mathcal{A})_\tau}$$

End Definition.

[In early work on algebraic specification (including, for example, [5 section 3.15]), the word "generated" or "term-generated" is used instead of "reachable."]

It is well known (see, for example, [5 proof of theorem 4.5,41 proof of fact 2.2.5]) that there is an injective homomorphism to an algebra from any of its reachable subalgebras. (This fact is used in the discussion of behavioural equivalence in chapter 3.)

Lemma 2.14 (homomorphism from reachable subalgebras)

Let Σ be a signature with sorts T, T' a subset of T, and $\mathcal{A}$ a Σ-algebra.

The T-indexed function h defined for each $\tau \in T$ and $a \in \mathcal{R}(\Sigma, T', \mathcal{A})_\tau$ by $h_\tau(a) \overset{\text{def}}{=} a$ is an injective Σ-homomorphism from $\mathcal{R}(\Sigma, T', \mathcal{A})$ to $\mathcal{A}$.

Proof Since all elements of $\mathcal{R}(\Sigma, T', \mathcal{A})_\tau$ can be written in the form $t_\mathcal{A}(v)$ where $t \in W(\Sigma, X')_\tau$ and $v \in \mathbf{Val}(\mathcal{A}, t)$, it is straightforward to verify that the homomorphism condition holds. Injectivity follows immediately from the definition of h.

End Lemma.

The following property is less well known — it is used in chapter 3 when establishing properties of behavioural equivalence.

Lemma 2.15 (quotients of reachable subalgebras)

Let Σ be a signature with sorts T, Σ' a subsignature of Σ and T' a subset of T and $\mathcal{A}$ and $\mathcal{B}$ Σ-algebras.

If $h: \mathcal{A} \to \mathcal{B}$ and $h|_{T'}$ is surjective then

$$\mathcal{R}(\Sigma', T', \mathcal{A})/_\equiv \cong \mathcal{R}(\Sigma', T', \mathcal{B})$$

where $\equiv: \mathcal{A}|_T \leftrightarrow \mathcal{A}|_T$ is the Σ-congruence defined for each sort $\tau \in T$ and values $a1, a2 \in \mathcal{A}_\tau$ by

$$a1 \equiv_\tau a2 \overset{\text{def}}{=} h_\tau(a1) = h_\tau(a2)$$

Proof

Let the T-indexed function $g: \mathcal{R}(\Sigma', T', \mathcal{B}) \rightarrow \mathcal{R}(\Sigma', T', \mathcal{A})/_{\equiv}$ be defined for each sort $\tau \in T$ and (Σ', T')-reachable value $b \in \mathcal{B}_\tau$ by

$$g_\tau(b) \overset{\text{def}}{=} \{a: h_\tau(a) = b: a\}$$

Then:

1. g is bijective.

   Since $\mathcal{R}(\Sigma', T', h)$ is surjective, every equivalence class in $\mathcal{R}(\Sigma', T', \mathcal{A})/_{\equiv}$ corresponds to precisely one (Σ', T')-reachable value in $\mathcal{B}$.

2. g is a homomorphism.

   Since $\mathcal{R}(\Sigma', T', h)$ is surjective, it suffices to show that, for each function symbol $f: \tau 1 \times \cdots \tau m \rightarrow \tau$ and (Σ', T')-reachable values $a1, \ldots am \in \mathcal{A}_{\tau 1, \ldots \tau m}$

$$
\begin{array}{l}
g_\tau(\mathcal{B}_f(h_{\tau 1}(a1), \ldots h_{\tau m}(am))) \\
= \quad \{ \ h \text{ is a homomorphism } \} \\
g_\tau(h_\tau(\mathcal{A}_f(a1, \ldots am))) \\
= \quad \{ \text{ definition of } g \ \} \\
\{a: h_\tau(a) = h_\tau(\mathcal{A}_f(a1, \ldots am)): a\} \\
= \quad \{ \text{ definition of } \equiv \ \} \\
\{a: a \equiv_\tau \mathcal{A}_f(a1, \ldots am): a\} \\
= \quad \{ \text{ definition of } [\![\text{-}]\!]_{\equiv} \ \} \\
[\![\mathcal{A}_f(a1, \ldots am)]\!]_{\equiv_\tau} \\
= \quad \{ \text{ definition of } \mathcal{R}(\Sigma', T', \mathcal{A}) \ \} \\
[\![\mathcal{R}(\Sigma', T', \mathcal{A})_f(a1, \ldots am)]\!]_{\equiv_\tau} \\
= \quad \{ \ [\![\text{-}]\!]_\tau \text{ is a homomorphism } \} \\
\mathcal{R}(\Sigma', T', \mathcal{A})/\equiv_f ([\![a1]\!]_{\equiv_\tau}, \ldots [\![am]\!]_{\equiv_{\tau m}}) \\
= \quad \{ \ [\![a]\!]_\tau = g_\tau(h_\tau(a)) \ \} \\
\mathcal{R}(\Sigma', T', \mathcal{A})/\equiv_f (g_{\tau 1}(h_{\tau 1}(a1)), \ldots g_{\tau m}(h_{\tau m}(am)))
\end{array}
$$

$\square$

Since g is a bijective Σ-homomorphism, we conclude that g is a Σ-isomorphism. Hence result.

End Lemma.

2.1.3 Formulæ and Axioms

This section defines formulæ and axioms. Σ-formulæ are just the standard formulæ of first-order logic with the addition of equality over Σ-terms ($t1 = t2$) and "reachable" quantification ($\forall^{\Sigma'}_{T'} x{:}\,\tau.\ P$); Σ-axioms are Σ-formulæ with no free variables. Reachable quantification differs from normal quantification in that we only quantify over reachable values.

Definition 2.16 (well formed formulæ, axioms and satisfaction)

Let Σ be a signature.

The set $WFF(\Sigma)$ of well-formed Σ-formulæ is defined as the least set satisfying

$$
\begin{array}{lll}
\mathit{true} & \in WFF(\Sigma) & \\
t1 =_\tau t2 & \in WFF(\Sigma) & \text{if } t1, t2 \text{ in } W(\Sigma, X)_\tau \\
\neg P & \in WFF(\Sigma) & \text{if } P \in WFF(\Sigma) \\
P \wedge Q & \in WFF(\Sigma) & \text{if } P \in WFF(\Sigma) \text{ and } Q \in WFF(\Sigma) \\
\forall^{\Sigma'}_{T'} x{:}\,\tau.\ P & \in WFF(\Sigma) & \text{if } P \in WFF(\Sigma),\ \Sigma' \subseteq \Sigma \text{ and } x \in X_\tau \\
\forall x{:}\,\tau.\ P & \in WFF(\Sigma) & \text{if } P \in WFF(\Sigma) \text{ and } x \in X_\tau
\end{array}
$$

The set of free variables in a well-formed Σ-formula is defined as follows. (Note the use of $\mathbf{free}(P) - \{x\}$ in the last line which removes a variable x from the set of free variables when it is bound by a quantifier.)

$$
\begin{array}{ll}
\mathbf{free}(\mathit{true}) & = \emptyset \\
\mathbf{free}(t1 =_\tau t2) & = \mathbf{vars}(t1) \cup \mathbf{vars}(t2) \\
\mathbf{free}(\neg P) & = \mathbf{free}(P) \\
\mathbf{free}(P \wedge Q) & = \mathbf{free}(P) \cup \mathbf{free}(Q) \\
\mathbf{free}(\forall^{\Sigma'}_{T'} x{:}\,\tau.\ P) & = \mathbf{free}(P) - \{x\} \\
\mathbf{free}(\forall x{:}\,\tau.\ P) & = \mathbf{free}(P) - \{x\}
\end{array}
$$

The T-indexed set of free variables in a formula P (written $\mathbf{Free}(P)$) is defined for each $\tau \in T$ by $\mathbf{Free}_\tau(P) \stackrel{\text{def}}{=} X_\tau \cap \mathbf{free}(P)$. We extend the notation for valuations by defining $\mathbf{Val}(\mathcal{A}, P)$ to be the set of all partial T-indexed functions $v{:}\,X \nrightarrow \mathcal{A}$ such that $v_\tau(x)$ is defined for each $x \in \mathbf{Free}(P)$.

A well-formed Σ-formula ax is a Σ-axiom if $\mathbf{free}(ax) = \emptyset$. We write $\mathbf{Axm}(\Sigma)$ to denote the set of all Σ-axioms.

Let $\mathcal{A}$ be an algebra, P a well-formed Σ-formula and $v \in \mathbf{Val}(\mathcal{A}, P)$ a valuation.

The satisfaction of P by $\mathcal{A}$ with respect to v (written $\mathcal{A} \models_v P$) is defined by

$$\begin{array}{ll}
\mathcal{A} \models_v \mathit{true} & \stackrel{\text{def}}{=} \mathit{true} \\
\mathcal{A} \models_v \mathit{t1} =_\tau \mathit{t2} & \stackrel{\text{def}}{=} \mathit{t1}_{\mathcal{A}}(v) = \mathit{t2}_{\mathcal{A}}(v) \\
\mathcal{A} \models_v \neg P & \stackrel{\text{def}}{=} \neg(\mathcal{A} \models_v P) \\
\mathcal{A} \models_v P \wedge Q & \stackrel{\text{def}}{=} (\mathcal{A} \models_v P) \wedge (\mathcal{A} \models_v Q) \\
\mathcal{A} \models_v (\forall^{\Sigma'}_{T'} x{:}\tau.\ P) & \stackrel{\text{def}}{=} (\forall a : a \in \mathcal{A}_\tau \wedge \mathcal{R}(\Sigma', T', a) : \mathcal{A} \models_{v \oplus \{x:=a\}} P) \\
\mathcal{A} \models_v (\forall x{:}\tau.\ P) & \stackrel{\text{def}}{=} (\forall a : a \in \mathcal{A}_\tau : \mathcal{A} \models_{v \oplus \{x:=a\}} P)
\end{array}$$

For any Σ-algebra $\mathcal{A}$ and Σ-axiom ax, $\mathcal{A}$ satisfies ax (written $\mathcal{A} \models ax$) iff $\mathcal{A} \models_{\{\}} ax$ where $\{\}$ denotes the completely undefined valuation. Also, for any set Ax of Σ-axioms, we write $\mathcal{A} \models Ax$ as an abbreviation for $(\forall ax{:}\ ax \in Ax{:}\ \mathcal{A} \models ax)$.

End Definition.

Our definition of reachable quantification is based on that of Schoett [36,37]. It differs in that we make the signature Σ' and set of sorts T' explicit whereas Schoett requires $\Sigma' = \Sigma$ and makes the set T' implicit in what he calls an "observational signature."

The use of reachable quantification in the algebraic literature can be traced (at least) as far back as Maibaum et al. [14,15] and Poigné [21]. All these early works use reachable quantification for the same purpose as model-based specifications use invariants: to restrict the domain of concern to those values which the specifier expects programs to encounter during execution — that is, the reachable values.

Wirsing and Broy [42] define a family of predicates $\_ \in \Sigma'_\tau$ for each (non-empty) subsignature Σ' of Σ and each sort $\tau \in \mathbf{Tp}(\Sigma)$ with semantics

$$\mathcal{A} \models_v t \in \Sigma'_\tau \stackrel{\text{def}}{=} \mathcal{R}(\Sigma', \emptyset, t_{\mathcal{A}}(v))$$

which they use to define a less general form of reachable quantification (restricted to the case that $T' = \emptyset$) by

$$\forall^{\Sigma'} x{:}\,\tau.\ P \overset{\text{def}}{=} \forall x{:}\,\tau.\ x \in \Sigma'_\tau \Rightarrow P$$

We could remove $\forall x{:}\,\tau.\ P$ from the definition of $WFF(\Sigma)$ since it can be defined as follows:

$$\forall x{:}\,\tau.\ P \overset{\text{def}}{=} \forall^{\emptyset}_{\{\tau\}} x{:}\,\tau.\ P$$

We define the abbreviations $\neq$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, ... in the usual manner. For example, we have:

$$\begin{array}{lll}
t1 \neq_\tau t2 & \overset{\text{def}}{=} & \neg(t1 =_\tau t2) \\
P \vee Q & \overset{\text{def}}{=} & \neg((\neg P) \wedge (\neg Q)) \\
P \Rightarrow Q & \overset{\text{def}}{=} & (\neg P) \vee Q \\
P \Leftrightarrow Q & \overset{\text{def}}{=} & (P \Rightarrow Q) \wedge (Q \Rightarrow P) \\
\exists x{:}\,\tau.\ P & \overset{\text{def}}{=} & \neg\forall x{:}\,\tau.\ \neg P \\
\exists^{\Sigma'}_{T'} x{:}\,\tau.\ P & \overset{\text{def}}{=} & \neg\forall^{\Sigma'}_{T'} x{:}\,\tau.\ \neg P \\
t \in \Sigma'(T')_\tau & \overset{\text{def}}{=} & (\exists^{\Sigma'}_{T'} y{:}\,\tau.\ y =_\tau t)
\end{array}$$

It is well known that first-order logic cannot distinguish reachable and unreachable models of the natural numbers (see, for example, [3 corollary 2.1.7]) whereas the axiom

$$\forall x{:}\ Nat.\ \exists^{\{0,succ\}}_{\emptyset} y{:}\ Nat.\ x =_{Nat} y$$

can. Therefore, the addition of reachable quantification increases the expressive power of first-order logic.

2.2 Specifications

This section defines the semantics of the languages ASL and USL and presents some examples of their use. The bulk of this work lies in the definition of some "specification building operations" which are used to construct complex specifications out of simple specifications.

One important point to note about ASL and USL is that if an axiom holds in an ASL/USL specification, then it must hold in all implementations of that specification. For example, all implementations of the specification *Bool* in the introduction to this chapter will satisfy the axiom

$$\forall x\colon Bool.\ x = \text{True} \vee x = \text{False}$$

and so will have at most two elements in the sort *Bool*.

ASL and USL are unusual in this respect in that the semantics of many alternative specification languages allow implementations which do not literally satisfy the axioms as long as the user of such an implementation could not tell that the axiom was broken. For example, the notion of implementation proposed by ADJ [9 section 5.5] is based on the relationship $\underset{\sim}{\subset}$ meaning "isomorphic to a subalgebra of". Under this notion of implementation, it would be possible for an implementation of the specification *Bool* to have three elements in the sort *Bool* since such an implementation would have models which are isomorphic to a subalgebra of a model of *Bool*.

2.2.1 Specification Building Operations

Many papers have been written about ASL (see, for example, [27–30,32–34,40,41]); each defining a slightly different set of specification building operations.

Rather than list all operations ever defined for ASL, we shall consider only those operations which appear in all definitions of ASL. That is, we consider the following five specification building operations:

- The simplest form of specification consists of a signature and set of axioms. Such specifications are known as "flat" specifications.
- Just as reducts are used to hide, rename or copy objects in an algebra, so the specification building operation "**derive**" is used to hide, rename or copy objects in a specification.

- The converse of "**derive**" is "**translate**" which is primarily used to define the extension of a specification to a larger signature.
- The constraints placed on implementations of the specification $SP1 \cup SP2$ consists of the constraints placed on implementations of $SP1$ and the constraints placed on implementations of $SP2$.
- The abstractor "**behaviour**" closes a specification under behavioural equivalence. We have chosen to delay the definition and discussion of behavioural equivalence until chapter 3 to keep all discussion of behavioural equivalence and behavioural closure in one chapter. For now, it suffices to know that we are going to model behavioural equivalence of program modules by an equivalence relation $\xleftrightarrow[OUT]{IN}$: $\mathbf{Alg}(\Sigma) \leftrightarrow \mathbf{Alg}(\Sigma)$ between algebras.

These five specification building operations are defined as follows:

Definition 2.17 (specifications)

Let Σ and Σ' be signatures, Ax a set of Σ-axioms.

The set $\mathbf{Spec}(\Sigma)$ of Σ-specifications is defined as the least set satisfying

$\langle \Sigma, Ax \rangle$	$\in \mathbf{Spec}(\Sigma)$	if $Ax \subseteq \mathbf{Axm}\,\Sigma$
derive from SP' **by** σ	$\in \mathbf{Spec}(\Sigma)$	if $SP' \in \mathbf{Spec}(\Sigma')$ and $\sigma: \Sigma \to \Sigma'$
translate SP' **by** σ	$\in \mathbf{Spec}(\Sigma)$	if $SP' \in \mathbf{Spec}(\Sigma')$ and $\sigma: \Sigma' \to \Sigma$
$SP1 \cup SP2$	$\in \mathbf{Spec}(\Sigma)$	if $SP1, SP2 \in \mathbf{Spec}(\Sigma)$
behaviour SP **wrt** (IN, OUT)	$\in \mathbf{Spec}(\Sigma)$	if $SP \in \mathbf{Spec}(\Sigma)$ and $IN, OUT \subseteq \mathbf{Tp}(\Sigma)$

The signature of a specification $SP \in \mathbf{Spec}(\Sigma)$ (written $\mathbf{Sig}(SP)$) is the signature Σ.

Every well-formed specification $SP \in \mathbf{Spec}(\Sigma)$ determines a class of algebras $\mathbf{Mod}(SP) \subseteq \mathbf{Alg}(\Sigma)$ (the "models" of SP). This set is inductively defined by

$$\begin{array}{lcl}
\mathbf{Mod}(\langle \Sigma, Ax\rangle) & \overset{\text{def}}{=} & \{\mathcal{A} : \mathcal{A} \in \mathbf{Alg}(\Sigma) \wedge \mathcal{A} \models Ax : \mathcal{A}\} \\
\mathbf{Mod}(\textbf{derive from } SP' \textbf{ by } \sigma) & \overset{\text{def}}{=} & \{\mathcal{A} : \mathcal{A} \in \mathbf{Mod}(SP') : \mathcal{A}|_\sigma\} \\
\mathbf{Mod}(\textbf{translate } SP' \textbf{ by } \sigma) & \overset{\text{def}}{=} & \{\mathcal{A} : \mathcal{A}|_\sigma \in \mathbf{Mod}(SP') : \mathcal{A}\} \\
\mathbf{Mod}(SP1 \cup SP2) & \overset{\text{def}}{=} & \mathbf{Mod}(SP1) \cap \mathbf{Mod}(SP2) \\
\mathbf{Mod}(\textbf{behaviour } SP \textbf{ wrt } (IN, OUT)) & \overset{\text{def}}{=} & \mathbf{Cl}_{\underset{OUT}{\overset{IN}{\longleftrightarrow}}}(SP)
\end{array}$$

A Σ-algebra $\mathcal{A}$ is said to be a *model of a Σ-specification SP* (written $\mathcal{A}: SP$) if $\mathcal{A} \in \mathbf{Mod}(SP)$. Two specifications $SP1, SP2: \mathbf{Spec}(\Sigma)$ are said to be equivalent (written $SP1 = SP2$) if $\mathbf{Mod}(SP1) = \mathbf{Mod}(SP2)$. A specification is said to be "inconsistent" if $\mathbf{Mod}(SP) = \emptyset$ and "consistent" otherwise. A specification $SP \in \mathbf{Spec}(\Sigma)$ is said to *satisfy* a Σ-axiom ax (written $SP \models ax$) if every model of SP satisfies ax.

End Definition.

Omissions and Abbreviations

Many other ASL operations have been suggested in [28–30,32–34,40,41]. Some of these are easily defined as abbreviations using the above operations whilst others are rarely used but are included for completeness. (The first 6 are used in this thesis; **quotient** and **extend _ to _ via _** are used as examples in chapter 5.)

- **spec** **sign** Σ $\overset{\text{def}}{=} \langle \Sigma, Ax\rangle$
  axioms Ax
  end

- The operation **export** Σ' **from** _ hides those symbols not occurring in Σ'; the operation **hide** S **in** _ hides those symbols that occur in a set S of symbols.

  export Σ' **from** $SP \overset{\text{def}}{=}$ **derive from** SP **by** σ
  hide S **in** $SP \overset{\text{def}}{=}$ **export** $\Sigma - S$ **from** SP

  where $\sigma: \Sigma' \hookrightarrow \mathbf{Sig}(SP)$

- The operation **extend** _ **to** Σ adds the symbols occurring in Σ to a specification.

  extend SP' **to** $\Sigma \overset{\text{def}}{=}$ **translate** SP' **by** σ

where $\sigma\colon \mathbf{Sig}(SP') \hookrightarrow \Sigma$

- The operation $\_ + \_$ is like $\_ \cup \_$ but is used to combine specifications with overlapping (rather than identical) signatures.

$$SP1 + SP2 \overset{\text{def}}{=} (\textbf{extend } SP1 \textbf{ to } \Sigma 1 \cup \Sigma 2) \cup (\textbf{extend } SP2 \textbf{ to } \Sigma 1 \cup \Sigma 2)$$

where $\Sigma 1 = \mathbf{Sig}(SP1)$ and $\Sigma 2 = \mathbf{Sig}(SP2)$.

- The operation **impose** Ax **on** $\_$ restricts the models of a specification to those satisfying the axioms Ax.

$$\textbf{impose } Ax \textbf{ on } SP \overset{\text{def}}{=} SP \cup \langle \mathbf{Sig}(SP), Ax \rangle$$

- The operation **enrich** $\_$ **by sign** S **axioms** Ax **end** takes a specification SP' and both adds the symbols S to the signature of SP' and imposes the axioms Ax on SP'.

$$\begin{array}{ll} \textbf{enrich } SP' & \overset{\text{def}}{=} \; SP' + \langle \Sigma, Ax \rangle \\ \textbf{by sign } \Sigma - \Sigma' & \\ \quad \textbf{axioms } Ax & \\ \textbf{end} & \end{array}$$

where $\Sigma' = \mathbf{Sig}(SP')$.

- The operation **reachable** $\_$ **on** T' restricts the models of a specification to those which are reachable on the sorts T'.

$$\begin{array}{lll} \textbf{reachable } SP \textbf{ on } T' \overset{\text{def}}{=} & \textbf{impose} & \forall x1\colon \tau 1.\ x1 \in \Sigma(T - T')_{\tau 1} \\ & & \vdots \\ & & \forall xm\colon \tau m.\ xm \in \Sigma(T - T')_{\tau m} \\ & \textbf{on } SP & \end{array}$$

where $\Sigma = \mathbf{Sig}(SP)$, $T = \mathbf{Tp}(\Sigma)$ and $T' = \{\tau 1, \ldots \tau m\}$.

- The operation **quotient** _ **wrt** E is defined in [29] by

$$\mathbf{Mod}(\textbf{quotient } SP \textbf{ wrt } E) \stackrel{\text{def}}{=} \{\mathcal{A}{:}\, \mathcal{A} \in \mathbf{Mod}(SP){:}\, \mathcal{A}/_{\equiv}\}$$

  where $\equiv$ is a Σ-congruence determined by the set E of Σ-equations (see [29] for details of $\equiv$).

- The operation **extend** _ **to** SP' **via** σ is defined in [29] by

$$\mathbf{Mod}(\textbf{extend } SP \textbf{ to } SP' \textbf{ via } \sigma) \stackrel{\text{def}}{=} \{\mathcal{A}{:}\, \mathcal{A} \in \mathbf{Mod}(SP){:}\, F_\sigma(\mathcal{A})\}$$

  where SP' is an equational specification and F_σ: $\mathbf{Alg}(\mathbf{Sig}(SP)) \to \mathbf{Mod}(SP')$ is a free functor (see [29] for details of F_σ).

- Most papers describing ASL describe an operator λ for forming parameterised specifications; two recent papers describing ASL [30,33] define an operator Π for forming specifications of parameterised programs (cf. Standard ML's "functors").

  We do not attempt to discuss parameterisation in this thesis.

Implementation

Informal notions of stepwise implementation of a specification or stepwise program design are based on the idea that one program design is an implementation of another if it incorporates more design decisions. Sannella and Tarlecki [29] formalise this using the refinement relation "$\rightsquigarrow$" on specifications defined by

$$SP1 \rightsquigarrow SP2 \quad \stackrel{\text{def}}{=} \quad \mathbf{Mod}(SP2) \subseteq \mathbf{Mod}(SP1)$$

Since the relation $\rightsquigarrow$ is transitive, we can conclude that $SP1 \rightsquigarrow SPm$ if we have shown that $SP1 \rightsquigarrow SP2 \rightsquigarrow \cdots SPm$. That is, if we develop an implementation SPm from $SP1$ in a series of refinement steps, we are guaranteed that SPm is an implementation of $SP1$.

At first glance, this idea of "implementing" a specification by another specification might seem to be rather useless: if an "implementation" is itself an ASL specification, it will not be possible to directly execute the "implementation." Broy et al.'s justification for such a definition is as follows [1 section 7]:

> ... if it happens that *SP2* specifies precisely the behaviour of particular data structures in a concrete programming language then a program over these types is both abstract *and* concrete. So a class of data types (for which software realisations are available) should be given, and the types of an abstract program should be replaced by algebraic implementations until it is based on the given target types.

An example of a specification language which includes a set of concrete data types is Extended ML [25,26,31] which consists of a blend of first-order logic and (a functional subset of) the programming language Standard ML.

2.2.2 Examples

This section gives some examples of typical specifications written in the above specification language and informally explains their meaning.

The first specification is a repeat of the specification of booleans from the introduction to this chapter. The specification has a single sort and two constant operations on that sort. The first axiom requires that these constants have different values; and the second axiom requires that every value of the sort *Bool* is equal to one constant or another. The result is that the sort *Bool* has precisely two values in it: True and False.

```
BoolBase ≝
spec sign Bool :type
          True, False :→ Bool
     axioms True ≠ False
            ∀x: Bool. x = True ∨ x = False
end
```

Using the abbreviation $x \in \Sigma'(T')$ from section 2.1.3, the second axiom could also have been written as:

$$\forall x\colon Bool.\ x \in \{\text{True}, \text{False}\}(\emptyset)$$

The second specification is a small specification of the natural numbers. The specification has a single sort *Nat* and operations *0* and *succ* for constructing the naturals ($1 = succ(0)$, $2 = succ(1)$, etc.). The third axiom ensures that each value in *Nat* can be constructed using *0* and *succ* (eliminating values like -5 or infinity) while the first and second axioms ensure that every value can be uniquely constructed (so $0 \neq 1 \neq 2 \neq 3 \neq \ldots$).

$$\boxed{\begin{array}{l} NatBase \overset{\text{def}}{=} \\ \textbf{spec}\ \ \textbf{sign}\ \ Nat\ \textbf{:type} \\ \qquad\qquad\ \ 0 :\rightarrow Nat \\ \qquad\qquad\ \ succ : Nat \rightarrow Nat \\ \qquad\ \ \textbf{axioms}\ \ \forall m\colon Nat.\ succ(m) \neq 0 \\ \qquad\qquad\qquad\ \ \forall m, n\colon Nat.\ succ(m) = succ(n) \Rightarrow m = n \\ \qquad\qquad\qquad\ \ \forall m\colon Nat.\ m \in \{0, succ\}(\emptyset) \\ \textbf{end} \end{array}}$$

If we allowed infinitary axioms, it would be possible to replace the third axiom by

$$\forall m\colon Nat.\ m = 0 \vee m = succ(0) \vee m = succ(succ(0)) \vee \ldots$$

Our final example (figure 2.1) shows how these simple specifications can be combined and extended to give larger specifications. The first four axioms specify the usual logical operators on booleans. The remaining axioms specify addition, subtraction and comparision operators and the constants *1* and *2*.

Note that all axioms are either an exhaustive case analysis or of the form

$$\forall xs\colon \tau s.\ f(xs) = t$$

so these operations are fully defined over their inputs.

$$
\begin{array}{l}
\mathit{Nat} \overset{\mathrm{def}}{=} \\
\textbf{enrich } \mathrm{BoolBase} + \mathrm{NatBase} \\
\textbf{by sign } \_\mathit{and}\_, \_\mathit{or}\_ : \mathit{Bool} \times \mathit{Bool} \to \mathit{Bool} \\
\qquad\qquad \mathit{not} : \mathit{Bool} \to \mathit{Bool} \\
\qquad\qquad \_+\_, \_-\_ : \mathit{Nat} \times \mathit{Nat} \to \mathit{Nat} \\
\qquad\qquad \mathit{1}, \mathit{2} :\to \mathit{Nat} \\
\qquad\qquad \_\leq\_, \_<\_, \_>\_, \_\geq\_ : \mathit{Nat} \times \mathit{Nat} \to \mathit{Bool} \\
\quad \textbf{axioms } \mathrm{True}\ \mathit{and}\ \mathrm{True} = \mathrm{True}\ \wedge\ \mathrm{True}\ \mathit{and}\ \mathrm{False} = \mathrm{False} \\
\qquad\qquad\quad \mathrm{False}\ \mathit{and}\ \mathrm{True} = \mathrm{False}\ \wedge\ \mathrm{False}\ \mathit{and}\ \mathrm{False} = \mathrm{False} \\
\qquad\qquad\quad \mathit{not}(\mathrm{True}) = \mathrm{False}\ \wedge\ \mathit{not}(\mathrm{False}) = \mathrm{False} \\
\qquad\qquad\quad \forall \mathit{b1}, \mathit{b2}{:}\,\mathit{Bool}.\ \mathit{b1}\ \mathit{or}\ \mathit{b2} = \mathit{not}(\mathit{not}(\mathit{b1})\ \mathit{and}\ \mathit{not}(\mathit{b2})) \\
\qquad\qquad\quad \forall m, n{:}\,\mathit{Nat}.\quad m + \mathit{succ}(n) = \mathit{succ}(m+n) \\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\ m + \mathit{0} = m \\
\\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\ \mathit{succ}(m) - \mathit{succ}(n) = m - n \\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\ m - \mathit{0} = m \\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\ \mathit{0} - n = \mathit{0} \\
\\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\ \mathit{succ}(m) \geq \mathit{succ}(n) = m \geq n \\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\ \mathit{succ}(m) \geq \mathit{0} = \mathrm{True} \\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\ \mathit{0} \geq \mathit{succ}(n) = \mathrm{False} \\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\ \mathit{0} \geq \mathit{0} = \mathrm{True} \\
\\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\ m < n = \mathit{not}(m \geq n) \\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\ m \leq n = n \geq m \\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\ m > n = \mathit{not}(m \leq n) \\
\qquad\qquad\quad \mathit{1} = \mathit{succ}(\mathit{0})\ \wedge\ \mathit{2} = \mathit{succ}(\mathit{1}) \\
\textbf{end}
\end{array}
$$

Figure 2.1: Specification of Natural Numbers

2.3 The ASL and USL sublanguages

This chapter has defined a large specification language containing two smaller languages. This section defines these sublanguages.

- The first sublanguage is ASL: a specification language developed by Sannella, Tarlecki and Wirsing [29,30,34].

  When it was first described [34], a distinguishing feature was the use of the behavioural abstraction operator **behaviour** _ **wrt** (_, _). Although ASL is generally described in an "institution independent" manner (that is without reference to any particular logical framework), we shall only consider that of first order logic in this thesis.

That is, ASL specifications are all specifications in **Spec** which do not use reachable quantification.

- The second sublanguage is the language developed by Wirsing and Broy [2,42] which we call USL.

  A notable feature of this language is its use of reachable quantification and its lack of the behavioural abstraction operator **behaviour** _ **wrt** (_, _) (chapter 4 explores how "abstract" specifications can be written without such an operator.)

  That is, USL specifications are all specifications in **Spec** which do not use behavioural abstraction.

In summary, the differences between the languages ASL and USL lies in how they avoid overspecification: ASL allows the behavioural abstraction operator but USL does not; and ASL does not allow reachable quantification but USL does.

Chapter 3

Behavioural Equivalence

Chapter 1 argues that it is important that any framework for *formal* program development should allow any "legitimate" *informal* program development and focusses on the following closure property:

> If a program module implements a specification then so should all behaviourally equivalent program modules.

This chapter formally defines the notion of behavioural equivalence used in this thesis.

There are a variety of alternative definitions used in the literature and so it is important to show how our definition relates to these definitions and to justify our choice over the alternatives. We show that our definition is a slight generalisation of that of Meseguer and Goguen [17] and slightly stronger than the notions of behavioural reduction and behavioural equivalence of Sannella and Tarlecki [27].

Section 3.3 explores the utility of our generalisation of Meseguer and Goguen's definition — demonstrating that special cases correspond to isomorphism, isomorphism of subalgebras, etc. and investigating a few simple properties of behavioural equivalence.

Section 3.4 discusses how behavioural equivalence may be applied to specifications and shows how the special cases discussed in the previous section give rise to a variety of common specification building operations.

In [37], Schoett identifies a set of axioms he calls "observational axioms" (written $\mathbf{Axm}(IN, OUT)^{1}$) and argues (without proof) that, for any observational axiom $ax \in \mathbf{Axm}(IN, OUT)$, and behaviourally equivalent algebras $\mathcal{A}$ and $\mathcal{B}$,

$$\mathcal{A} \models ax \quad \Leftrightarrow \quad \mathcal{B} \models ax$$

Section 3.5 defines observational axioms and verifies that their satisfaction is indeed invariant under our definition of behavioural equivalence.

This result is interesting for three reasons:

1. Schoett's argues that any notion of behavioural equivalence should satisfy this condition — verifying that it does so increases confidence that our definition is useful.

2. Sannella and Tarlecki have demonstrated that their notion of behavioural equivalence does not quite satisfy the above condition (they are forced to constrain the form of quantification allowed). We use this fact in demonstrating that our definition of behavioural equivalence is indeed stronger than that of Sannella and Tarlecki.

3. Finally, this result *seems* to point the way towards a subset of USL based only on observational axioms in which all specifications would naturally be behaviourally closed. Sadly, an "impossibility theorem" by Schoett [37] shows that the resulting language would be too weak to be useful (this result is discussed in section 3.6.) The weakness of such a language motivates the work in the next chapter where we examine the approach taken by Wirsing and Broy [2,42].

We begin with an informal justification for behavioural equivalence; similar justifications can be found in, for example, [17,27,34,35,37].)

[1] Schoett's notation is $\mathcal{L}(\Sigma, V)$ where $V = IN = OUT$.

The essence of behavioural equivalence is an attempt to model the effect of *information hiding* in modular programming. Specifically, two algebras are regarded as behaviourally equivalent (with respect to a given "interface") if their "visible parts" (as determined by the "interface") are indistinguishable.

An example (due to Sannella and Tarlecki [27]) is of a module providing a set-like abstract data type *Bunch* and operations $empty\colon \rightarrow Bunch$, $add\colon Nat \times Bunch \rightarrow Bunch$ and $\in\colon Nat \times Bunch \rightarrow Bool$ (as well as the types *Bool* and *Nat* and the usual operations on these types). There are several sensible implementations of such a module differing only in their implementation of the type *Bunch* and operations on that type. For example, one might have implementation $\mathcal{A}$ which represents a *Bunch* by an unordered array of naturals; and implementation $\mathcal{B}$ which represents a *Bunch* by an ordered binary tree with no duplicates. Despite $\mathcal{A}$ and $\mathcal{B}$ having different representations, a program which uses implementation $\mathcal{A}$ should be able to use implementation $\mathcal{B}$ (and vice-versa) without (other) modification. We say "should" because this will obviously only be true if the program treats *Bunch* as an abstract data type — that is, if it only constructs, modifies and accesses values of type *Bunch* through the available operations (*empty*, *add* and $\in$) and if its correctness depends only on the values of expressions of the form

$$a \in add(a1, \ldots add(am, empty) \ldots)$$

giving the appropriate answer. That is, $\mathcal{A}$ may be replaced by $\mathcal{B}$ because $\mathcal{A}$ is *behaviourally equivalent to* $\mathcal{B}$ *with respect to terms of sort Bool.*

We shall now formally define the behavioural equivalence relation. It is convenient to begin with a special case directly inspired by the above example and then generalise.

3.1 Behavioural Equivalence — Special Case

The "*Bunch* example" above immediately suggests the following definition (due to Sannella and Tarlecki [29 example 6.6]). Note that this is a *special case* of definition 3.5 below.

Definition 3.1 (Behavioural equivalence — ground case)

Let Σ be a signature with sorts T, W a subset of $W(\Sigma, \emptyset)$ and let $\mathcal{A}$ and $\mathcal{B}$ be two Σ-algebras.

The algebras $\mathcal{A}$ and $\mathcal{B}$ are *observationally equivalent with respect to* W (written $\mathcal{A} \equiv_W \mathcal{B}$) if, for all sorts $\tau \in T$ and terms $t1, t2 \in W_\tau$,

$$\mathcal{A} \models t1 =_\tau t2 \Leftrightarrow \mathcal{B} \models t1 =_\tau t2$$

Let OBS be a subset of T. The algebras $\mathcal{A}$ and $\mathcal{B}$ are *behaviourally equivalent with respect to OBS* (written $\mathcal{A} \equiv_{OBS} \mathcal{B}$) if

$$\mathcal{A} \equiv_{W(\Sigma,\emptyset)|_{OBS}} \mathcal{B}$$

(That is, if $\mathcal{A} \equiv_{W'} \mathcal{B}$ where W' is the T-indexed set of all ground terms with sort $\tau \in OBS$.)

End Definition.

There are two alternative (equivalent) ways of defining this special case of behavioural equivalence. The first is (also) due to Sannella and Tarlecki while the second is based on a definition due to Meseguer and Goguen.

Sannella and Tarlecki's alternative definition [27 section 2] has a more "axiomatic flavour" and is based on the use of axioms instead of terms. They define observational equivalence with respect to a set of axioms and suggest behavioural equivalence with respect to a set of sorts as a specific instance.

Definition 3.2 (Behavioural equivalence — alternative definition)

Let Σ be a signature, OBS a subset of the sorts of Σ, Ax a set of Σ-axioms and $\mathcal{A}$ and $\mathcal{B}$ two Σ-algebras.

The algebras $\mathcal{A}$ and $\mathcal{B}$ are *observationally equivalent with respect to* Ax (written $\mathcal{A} \equiv_{Ax} \mathcal{B}$) if, for all axioms $ax \in Ax$,

$$\mathcal{A} \models ax \Leftrightarrow \mathcal{B} \models ax$$

The set EQ_{OBS} of ground Σ-equations over sorts in OBS is defined by

$$EQ_{OBS} \overset{\text{def}}{=} \{\tau, t1, t2 : \tau \in OBS \wedge t1, t2 \in W(\Sigma, \emptyset)_\tau : t1 =_\tau t2\}$$

End Definition.

[We note that the property of observational axioms claimed by Schoett page 39 is almost identical to the definition of $\equiv_{Ax}$. The difference is that the former is a *property* of the set **Axm**(IN, OUT) while the latter is the *definition* of $\equiv_{Ax}$.]

It is trivial to show that $\equiv_{OBS}$ is equivalent to $\equiv_{EQ_{OBS}}$.

Meseguer and Goguen's alternative definition of behavioural equivalence [17 section 5] uses homomorphisms giving it a more "model-theoretic flavour." An appropriate *special case* of their definition is the following (we delay stating the precise relationship of this special case to their actual definition until after the general definition).

Definition 3.3 (Behavioural equivalence — alternative definition)

Let Σ be a signature, OBS a subset of the sorts of Σ and $\mathcal{A}$ and $\mathcal{B}$ two Σ-algebras.

If $h: \mathcal{A} \to \mathcal{B}$ is a Σ-homomorphism such that, for each sort $\tau \in OBS$, h_τ is injective, we say that h is a OBS-homomorphism and write $h: \mathcal{A} \underset{OBS}{\to} \mathcal{B}$.

The relation $\underset{OBS}{\to}$: **Alg**$(\Sigma) \leftrightarrow$ **Alg**(Σ) is defined for Σ-algebras $\mathcal{A}$ and $\mathcal{B}$ by

$$\mathcal{A} \underset{OBS}{\to} \mathcal{B} \quad \text{iff} \quad \text{there is a } OBS\text{-homomorphism } h: \mathcal{A} \underset{OBS}{\to} \mathcal{B}$$

The relation $\underset{OBS}{\longleftrightarrow}$: **Alg**$(\Sigma) \leftrightarrow$ **Alg**(Σ) is defined to be the least equivalence containing $\underset{OBS}{\to}$. If $\mathcal{A} \underset{OBS}{\longleftrightarrow} \mathcal{B}$ we say that $\mathcal{A}$ and $\mathcal{B}$ are *OBS-behaviourally equivalent.*

End Definition.

For the "*Bunch* example" given above, one could define a homomorphism h such that h_{Bunch} maps an array with elements $a1, \ldots am$ to an ordered binary tree with elements $a1, \ldots am$ and h_{Nat} and h_{Bool} are identity functions. Since h_{Bool} is injective, it follows that $\mathcal{A}$ and $\mathcal{B}$ are $\{Bool\}$-behaviourally equivalent.

Using lemma 2.12 it is straightforward to show that if $\mathcal{A} \underset{OBS}{\rightarrow} \mathcal{B}$ then $\mathcal{A} \equiv_{OBS} \mathcal{B}$. It follows from properties of equivalences that if $\mathcal{A} \underset{OBS}{\longleftrightarrow} \mathcal{B}$, then $\mathcal{A} \equiv_{OBS} \mathcal{B}$. (We omit details of the proof since it is a special case of a similar result (lemma 3.6) for the more general definition.)

Using the injectivity of the inclusions $\iota_{\mathcal{A}}: \mathcal{R}(\Sigma, \emptyset, \mathcal{A}) \hookrightarrow \mathcal{A}$ and $\iota_{\mathcal{B}}: \mathcal{R}(\Sigma, \emptyset, \mathcal{B}) \hookrightarrow \mathcal{B}$ it is straightforward to show that if $\mathcal{A} \equiv_{OBS} \mathcal{B}$ then $\mathcal{A} \underset{OBS}{\longleftrightarrow} \mathcal{B}$.

3.2 Behavioural Equivalence — General Case

Under most circumstances, the above definitions would be perfectly adequate. However, there are some algebras which are behaviourally equivalent to the two "*Bunch* algebras" above which we might want to exclude. For example, consider the algebra $\mathcal{C}$ which is identical to $\mathcal{A}$ except that $\mathcal{C}_{Nat}$ is the set of integers and, for any negative integer n, and bunch b

$$\begin{aligned} add(n, b) &\stackrel{\text{def}}{=} b \\ n \in b &\stackrel{\text{def}}{=} \text{False} \end{aligned}$$

Under the above definitions, the algebras $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ are behaviourally equivalent with respect to the sort *Bool* since the additional elements in $\mathcal{C}$ are unreachable and so cannot be observed using ground equations. However, whilst $\mathcal{A}$ and $\mathcal{B}$ are obviously quite similar, $\mathcal{C}$ behaves a bit differently. More precisely, while $\mathcal{A}$ and $\mathcal{B}$ both satisfy the axiom

$$\forall n: Nat, b: Bunch.\ n \in add(n, b) = \text{True}$$

the algebra $\mathcal{C}$ does not (if n is negative).

Such considerations suggest that we might sometimes want to use a stronger notion of behavioural equivalence which constrains not just the "output" sort of terms we consider but the "input" sorts too. That is, we might want to consider behavioural equivalence with respect to a set *IN* of "input" sorts and a set *OUT* of "output" sorts.

The definition of $\underset{OBS}{\longleftrightarrow}$ is the simplest to generalise.

Definition 3.4 ($\overset{IN}{\underset{OUT}{\to}}$ and $\overset{IN}{\underset{OUT}{\longleftrightarrow}}$)

Let Σ be a signature, *IN* and *OUT* subsets of the sorts in Σ and $\mathcal{A}$ and $\mathcal{B}$ two Σ-algebras.

If $h\colon \mathcal{A} \to \mathcal{B}$ is a Σ-homomorphism such that $h|_{IN}$ is surjective and $h|_{OUT}$ is injective, then h is said to be an (IN, OUT)-homomorphism (written $h\colon \mathcal{A} \overset{IN}{\underset{OUT}{\to}} \mathcal{B}$).

The preorder $\overset{IN}{\underset{OUT}{\to}}\colon \mathbf{Alg}(\Sigma) \leftrightarrow \mathbf{Alg}(\Sigma)$ is defined for Σ-algebras $\mathcal{A}$ and $\mathcal{B}$ by

$$\mathcal{A} \overset{IN}{\underset{OUT}{\to}} \mathcal{B} \quad \text{iff} \quad \text{there exists } h\colon \mathcal{A} \overset{IN}{\underset{OUT}{\to}} \mathcal{B}$$

The equivalence $\overset{IN}{\underset{OUT}{\longleftrightarrow}}\colon \mathbf{Alg}(\Sigma) \leftrightarrow \mathbf{Alg}(\Sigma)$ is the smallest equivalence containing $\overset{IN}{\underset{OUT}{\to}}$. If $\mathcal{A} \overset{IN}{\underset{OUT}{\longleftrightarrow}} \mathcal{B}$, we say that $\mathcal{A}$ and $\mathcal{B}$ are *(IN, OUT)-behaviourally equivalent.*

End Definition.

This is a straightforward generalisation of the definition of behavioural equivalence used by Meseguer and Goguen [17]. Meseguer and Goguen discuss V-behavioural equivalence which is a special case of (IN, OUT)-behavioural equivalence with $IN = V$ and $OUT = V$. Their definition uses what they call V-homomorphisms which are homomorphisms that are bijective on all sorts in V — again, this is just a special case of our (IN, OUT)-homomorphisms with $IN = V$ and $OUT = V$.

It is, perhaps, worth pointing out that this is not just generalisation for the sake of it. Lemma 3.11 below shows that, for the special case that $OUT = \mathbf{Tp}(\Sigma)$, two algebras are (IN, OUT)-behaviourally equivalent iff their (Σ, IN)-reachable subalgebras are isomorphic. That is,

$$\mathcal{A} \overset{IN}{\underset{\mathbf{Tp}(\Sigma)}{\longleftrightarrow}} \mathcal{B} \quad \text{iff} \quad \mathcal{R}(\Sigma, IN, \mathcal{A}) \cong \mathcal{R}(\Sigma, IN, \mathcal{B})$$

and so the relation $\xleftrightarrow[OUT]{IN}$ with $IN \neq OUT$ is both meaningful and, in appropriate circumstances, useful. (See also section 3.4.)

There is an obvious generalisation of the definition of $\equiv_{OBS}$ and $\equiv_{EQ_{OBS}}$. Namely

> For all sorts $\tau \in OUT$ and terms $t1, t2 \in W(\Sigma, X)_\tau$ (with all free variables in $t1$ and $t2$ being of a sort in IN) and all valuations $v \in \mathbf{Val}(\mathcal{A}, t1 =_\tau t2)$,
>
> $$\mathcal{A} \models_v t1 =_\tau t2 \Leftrightarrow \mathcal{B} \models_v t1 =_\tau t2$$

However, such a definition would be well defined only if $\mathcal{A}|_{IN} = \mathcal{B}|_{IN}$. Sannella and Tarlecki [27 section 3] avoid this requirement with the following definition. (Discussion of other inappropriate generalisations may be found in [27 section 3].)

Definition 3.5 (observational and behavioural equivalence)

Let Σ be a signature with sorts T and ϕ a set of Σ-formulæ,

For any two Σ-algebras $\mathcal{A}$ and $\mathcal{B}$, $\mathcal{A}$ is "observationally reducible to $\mathcal{B}$ with respect to ϕ" (written $\mathcal{A} \leq_\phi \mathcal{B}$) if, for any valuation $va \in \mathbf{Val}(\mathcal{A})$, there is a valuation $vb \in \mathbf{Val}(\mathcal{B})$ with $dom(va) = dom(vb)$ such that, for every formula $\varphi \in \phi$

$$\mathcal{A} \models_{va} \varphi \Leftrightarrow \mathcal{B} \models_{vb} \varphi$$

if $va \in \mathbf{Val}(\mathcal{A}, \varphi)$.

For any two Σ-algebras $\mathcal{A}$ and $\mathcal{B}$, $\mathcal{A}$ is "observationally equivalent to $\mathcal{B}$ with respect to ϕ" (written $\mathcal{A} \equiv_\phi \mathcal{B}$) if $\mathcal{A} \leq_\phi \mathcal{B}$ and $\mathcal{B} \leq_\phi \mathcal{A}$.

Let IN and OUT be subsets of T. The set $EQ(IN, OUT)$ consists of all equations $t1 =_\tau t2$ where $t1, t2 \in W(\Sigma, X)_\tau$ and all variables in $t1$ and $t2$ are of sorts in IN.

End Definition.

It is easy to show that $\xleftrightarrow[OUT]{IN}$ is at least as strong as $\equiv_{EQ(IN,OUT)}$.

Lemma 3.6 $(\overset{IN}{\underset{OUT}{\longleftrightarrow}} \subseteq \equiv_{EQ(IN,OUT)})$

Let Σ be a signature, IN and OUT be subsets of the sorts in Σ, $\mathcal{A}$ and $\mathcal{B}$ two Σ-algebras and $h: \mathcal{A} \to \mathcal{B}$ a Σ-homomorphism.

If $h: \mathcal{A} \overset{IN}{\underset{OUT}{\to}} \mathcal{B}$, then $\mathcal{A} \equiv_{EQ(IN,OUT)} \mathcal{B}$.

Proof.

For all sorts $\tau \in OUT$ and terms $t1, t2 \in W(\Sigma, X)_\tau$ (with all free variables in $t1$ and $t2$ being of a sort in IN) and all valuations $v \in \mathbf{Val}(\mathcal{A}, t1 =_\tau t2)$,

$\mathcal{A} \models_v t1 =_\tau t2$
$=$ { definition of $\models t1 = t2$ }
$t1_{\mathcal{A}}(v) = t2_{\mathcal{A}}(v)$
$=$ { injectivity of h_τ }
$h(t1_{\mathcal{A}}(v)) = h(t2_{\mathcal{A}}(v))$
$=$ { lemma 2.12 }
$t1_{\mathcal{B}}(h \cdot v) = t2_{\mathcal{B}}(h \cdot v)$
$=$ { definition of $\models t1 = t2$ }
$\mathcal{B} \models_{h \cdot v} t1 =_\tau t2$
□

Hence, $\mathcal{A} \leq_{EQ(IN,OUT)} \mathcal{B}$.

Since h is surjective on all sorts in IN, it follows that there is at least one injective IN-indexed function $g: \mathcal{B}|_{IN} \to \mathcal{A}|_{IN}$ such that $g \cdot h|_{IN} = id$.

Then, for each $\varphi \in EQ(IN, OUT)$ and valuation $va \in \mathbf{Val}(\mathcal{A}, \varphi)$,

$\mathcal{B} \models_{h \cdot va} \varphi$
$=$ { above }
$\mathcal{A} \models_{va} \varphi$
$=$ { $g \cdot h = id$ }
$\mathcal{A} \models_{g \cdot h \cdot va} \varphi$

Since h is surjective on IN, we can let $vb = h \cdot va$. It follows that, for each $\varphi \in EQ(IN, OUT)$ and valuation $vb \in \mathbf{Val}(\mathcal{B}, v)$

$\mathcal{B} \models_{vb} \varphi$
$=$
$\mathcal{A} \models_{g \cdot vb} \varphi$

and so $\mathcal{B} \leq_{EQ(IN,OUT)} \mathcal{A}$ and hence, $\mathcal{A} \equiv_{EQ(IN,OUT)} \mathcal{B}$. □

End Lemma.

It is considerably less obvious that $\equiv_{EQ(IN,OUT)}$ is weaker than $\overset{IN}{\underset{OUT}{\longleftrightarrow}}$. We delay the proof until later in this chapter (corollary 3.20).

3.3 Properties of Behavioural Equivalence

This section explores the extra generality of our definition: demonstrating that special cases correspond to isomorphism, isomorphism of subalgebras, etc. developing an alternative characterisation of $\overset{IN}{\underset{OUT}{\longleftrightarrow}}$.

An interesting special case of $\overset{IN}{\underset{OUT}{\longleftrightarrow}}$ concerns isomorphisms.

Lemma 3.7 (behavioural equivalence of isomorphic algebras)

Let Σ be a signature with sorts T and $\mathcal{A}$ and $\mathcal{B}$ two Σ-algebras.

Then $\mathcal{A} \overset{T}{\underset{T}{\longleftrightarrow}} \mathcal{B}$ iff $\mathcal{A} \cong \mathcal{B}$.

Proof. This follows from the fact that $(T,\ T)$-homomorphisms are bijective homomorphisms and hence isomorphisms; and that isomorphisms are bijective homomorphisms and hence $(T,\ T)$-homomorphisms. $\square$

End Lemma.

*

The next two lemmas demonstrate that any algebra $\mathcal{A}$ is behaviourally equivalent to its reachable subalgebra $\mathcal{R}(\Sigma, IN, \mathcal{A})$ and to any quotient $\mathcal{A}/_{\equiv}$ if $\equiv$ is what we call a $(\Sigma,\ OUT)$-congruence.

Lemma 3.8 (behavioural equivalence of reachable subalgebras)

Let Σ be a signature, IN and OUT subsets of the sorts in Σ and $\mathcal{A}$ a Σ-algebra.

Then,

$$\mathcal{R}(\Sigma, IN, \mathcal{A}) \overset{IN}{\underset{OUT}{\longleftrightarrow}} \mathcal{A}$$

Proof. The homomorphism from $\mathcal{R}(\Sigma, IN, \mathcal{A})$ to $\mathcal{A}$ defined in lemma 2.14 is injective on all sorts and surjective on all sorts in IN. Hence, $\mathcal{R}(\Sigma, IN, \mathcal{A}) \underset{OUT}{\overset{IN}{\rightarrow}} \mathcal{A}$ and, so, by the definition of $\underset{OUT}{\overset{IN}{\longleftrightarrow}}$, $\mathcal{R}(\Sigma, IN, \mathcal{A}) \underset{OUT}{\overset{IN}{\longleftrightarrow}} \mathcal{A}$. □

End Lemma.

Definition 3.9 ((Σ, OUT)-congruence)

Let Σ be a signature, OUT a subset of the sorts in Σ and $\mathcal{A}$ a Σ-algebra.

We say that a Σ-congruence $\equiv$ over $\mathcal{A}$ is a (Σ, OUT)-congruence if, for all sorts $\tau \in OUT$ and values $a1, a2 \in \mathcal{A}_\tau$,

$$a1 \equiv_\tau a2 \Rightarrow a1 = a2$$

End Definition.

Lemma 3.10 (behavioural equivalence of quotient algebras)

Let Σ be a signature, IN and OUT subsets of the sorts in Σ, $\mathcal{A}$ a Σ-algebra.

If $\equiv$ is a (Σ, OUT)-congruence over $\mathcal{A}$, then

$$\mathcal{A} \underset{OUT}{\overset{IN}{\longleftrightarrow}} \mathcal{A}/_\equiv$$

Proof. The homomorphism $[\![\_]\!]_\equiv$ defined in lemma 2.9 is surjective on all sorts and injective on all sorts in OUT. Hence, $\mathcal{A} \underset{OUT}{\overset{IN}{\rightarrow}} \mathcal{A}/\equiv$ and, so, by the definition of $\underset{OUT}{\overset{IN}{\longleftrightarrow}}$, $\mathcal{A} \underset{OUT}{\overset{IN}{\longleftrightarrow}} \mathcal{A}/\equiv$. □

End Lemma.

*

The following lemma shows the relationship between behavioural equivalence and reachability. (It is a generalisation of [27 fact 15].)

(Note here that IN and OUT will generally be different.)

Lemma 3.11 (behavioural equivalence and reachability)

Let Σ be a signature with sort T, IN a subset T and $\mathcal{A}$ and $\mathcal{B}$ two Σ-algebras.

Then,

$$\mathcal{A} \underset{T}{\overset{IN}{\longleftrightarrow}} \mathcal{B} \;\Leftrightarrow\; \mathcal{R}(\Sigma, IN, \mathcal{A}) \cong \mathcal{R}(\Sigma, IN, \mathcal{B})$$

Proof. We prove the two directions separately — the simplest is the left-to-right direction.

- $R \Rightarrow L$

  $\mathcal{R}(\Sigma, IN, \mathcal{A}) \cong \mathcal{R}(\Sigma, IN, \mathcal{B})$
  $=$ { lemma 3.7 }
  $\mathcal{R}(\Sigma, IN, \mathcal{A}) \underset{T}{\overset{T}{\longleftrightarrow}} \mathcal{R}(\Sigma, IN, \mathcal{B})$
  $\Rightarrow$ { $\underset{T}{\overset{T}{\longleftrightarrow}} \subseteq \underset{T}{\overset{IN}{\longleftrightarrow}}$ }
  $\mathcal{R}(\Sigma, IN, \mathcal{A}) \underset{T}{\overset{IN}{\longleftrightarrow}} \mathcal{R}(\Sigma, IN, \mathcal{B})$
  $=$ { lemma 3.8 }
  $\mathcal{A} \underset{T}{\overset{IN}{\longleftrightarrow}} \mathcal{B}$
  □

- $L \Rightarrow R$

  Let the relation $\sim$: $\mathbf{Alg}(\Sigma) \leftrightarrow \mathbf{Alg}(\Sigma)$ be defined by

$$\mathcal{A}1 \sim \mathcal{A}2 \overset{\text{def}}{=} \mathcal{R}(\Sigma, IN, \mathcal{A}1) \cong \mathcal{R}(\Sigma, IN, \mathcal{A}2)$$

  Suppose that there is an (IN, OUT)-homomorphism $h: \mathcal{A}1 \underset{T}{\overset{IN}{\rightarrow}} \mathcal{A}2$. Then,

$$\mathcal{R}(\Sigma, IN, h): \mathcal{R}(\Sigma, IN, \mathcal{A}1) \underset{T}{\overset{T}{\rightarrow}} \mathcal{R}(\Sigma, IN, \mathcal{A}2)$$

  and so, by lemma 3.7, $\mathcal{R}(\Sigma, IN, \mathcal{A}1) \cong \mathcal{R}(\Sigma, IN, \mathcal{A}2)$. That is,

$$\underset{OUT}{\overset{IN}{\rightarrow}} \subseteq \sim$$

  Let us write R^* for the least equivalence containing a binary relation R. The result follows by straightforward calculation:

true
= { above }
$\underset{OUT}{\overset{IN}{\rightarrow}} \subseteq \sim$
$\Rightarrow$ { set theory }
$(\underset{OUT}{\overset{IN}{\rightarrow}})^* \subseteq (\sim)^*$
= { definition of $\underset{OUT}{\overset{IN}{\longleftrightarrow}}$ and $\sim$ is an equivalence }
$\underset{OUT}{\overset{IN}{\longleftrightarrow}} \subseteq \sim$
□

So, if $\mathcal{A} \underset{OUT}{\overset{IN}{\longleftrightarrow}} \mathcal{B}$, then $\mathcal{A} \sim \mathcal{B}$. Hence result.

End Lemma.

*

We end this section with an alternative characterisation of $\underset{OUT}{\overset{IN}{\longleftrightarrow}}$.

Theorem 3.12 $(\mathcal{A} \underset{OUT}{\overset{IN}{\longleftrightarrow}} \mathcal{B} \;\Leftrightarrow\; \mathcal{R}(\Sigma, IN, \mathcal{A})/_{\equiv^{\mathcal{A}}} \cong \mathcal{R}(\Sigma, IN, \mathcal{B})/_{\equiv^{\mathcal{B}}})$

Let $\mathcal{A}$ and $\mathcal{B}$ be Σ-algebras.

There exist (Σ, OUT)-congruences $\equiv^{\mathcal{A}}$ over $\mathcal{R}(\Sigma, IN, \mathcal{A})$ and $\equiv^{\mathcal{B}}$ over $\mathcal{R}(\Sigma, IN, \mathcal{B})$ such that

$$\mathcal{R}(\Sigma, IN, \mathcal{A})/_{\equiv^{\mathcal{A}}} \cong \mathcal{R}(\Sigma, IN, \mathcal{B})/_{\equiv^{\mathcal{B}}}$$

if and only if

$$\mathcal{A} \underset{OUT}{\overset{IN}{\longleftrightarrow}} \mathcal{B}$$

Proof

Let the relation $\sim$: $\mathbf{Alg}(\Sigma) \leftrightarrow \mathbf{Alg}(\Sigma)$ be defined by $\mathcal{A} \sim \mathcal{B}$ iff there exist (Σ, OUT)-congruences $\equiv^{\mathcal{A}}$, $\equiv^{\mathcal{B}}$ such that

$$\mathcal{R}(\Sigma, IN, \mathcal{A})/_{\equiv^{\mathcal{A}}} \cong \mathcal{R}(\Sigma, IN, \mathcal{B})/_{\equiv^{\mathcal{B}}}$$

By lemma 2.15 we have that, if $h: \mathcal{A} \underset{OUT}{\overset{IN}{\rightarrow}} \mathcal{B}$, then

$$\mathcal{R}(\Sigma, IN, \mathcal{A})/_{\equiv^{\mathcal{A}}} \cong \mathcal{R}(\Sigma, IN, \mathcal{B})$$

where $\equiv^{\mathcal{A}}: \mathcal{A}|_T \leftrightarrow \mathcal{A}|_T$ is the (Σ, OUT)-congruence defined for each sort $\tau \in T$ and values $a1, a2 \in \mathcal{A}_\tau$ by

$$a1 \equiv^{\mathcal{A}}_{\tau} a2 \overset{\text{def}}{=} h_\tau(a1) = h_\tau(a2)$$

Let $\equiv^{\mathcal{B}}: \mathcal{B}|_T \leftrightarrow \mathcal{B}|_T$ be equality. It is straightforward to show that

$$\mathcal{R}(\Sigma, IN, \mathcal{B}) \cong \mathcal{R}(\Sigma, IN, \mathcal{B})/_{\equiv^{\mathcal{B}}}$$

and so

$$\mathcal{A} \underset{OUT}{\overset{IN}{\rightarrow}} \mathcal{B} \;\Rightarrow\; \mathcal{A} \sim \mathcal{B}$$

Let us write R^* for the least equivalence containing a binary relation R. The result follows by straightforward calculation:

$true$
$=$ { above }
$\underset{OUT}{\overset{IN}{\rightarrow}} \subseteq \sim$
$\Rightarrow$ { set theory }
$(\underset{OUT}{\overset{IN}{\rightarrow}})^* \subseteq (\sim)^*$
$=$ { definition of $\underset{OUT}{\overset{IN}{\longleftrightarrow}}$ and $\sim$ is an equivalence }
$\underset{OUT}{\overset{IN}{\longleftrightarrow}} \subseteq \sim$
□

So, if $\mathcal{A} \underset{OUT}{\overset{IN}{\longleftrightarrow}} \mathcal{B}$, then $\mathcal{A} \sim \mathcal{B}$.

To see that the converse is true, observe that

$\mathcal{R}(\Sigma, IN, \mathcal{A})/_{\equiv^{\mathcal{A}}} \cong \mathcal{R}(\Sigma, IN, \mathcal{B})/_{\equiv^{\mathcal{B}}}$
$=$ { lemma 3.7 }
$\mathcal{R}(\Sigma, IN, \mathcal{A})/_{\equiv^{\mathcal{A}}} \underset{T}{\overset{T}{\longleftrightarrow}} \mathcal{R}(\Sigma, IN, \mathcal{B})/_{\equiv^{\mathcal{B}}}$
$\Rightarrow$ { lemma 3.10 twice }
$\mathcal{R}(\Sigma, IN, \mathcal{A}) \underset{OUT}{\overset{T}{\longleftrightarrow}} \mathcal{R}(\Sigma, IN, \mathcal{B})$

$\Rightarrow$ { lemma 3.8 twice }
$\mathcal{A} \underset{OUT}{\overset{IN}{\longleftrightarrow}} \mathcal{B}$
$\square$

Hence result.

End Theorem.

3.4 Behavioural Equivalence and Specifications

This section uses the definitions and results of the previous chapter to define the notion of behavioural closure (verifying that it matches the discussion in chapter 1) and relates the behavioural abstraction operator to various specification building operations discussed in the literature.

Definition 3.13 (behavioural semantics, equivalence and closure)

Let Σ be a signature and IN and OUT subsets of the sorts of Σ.

The *(IN, OUT)-behavioural semantics of a Σ-specification SP* is the set $\mathbf{Mod}^{IN}_{OUT}(SP)$ of Σ-algebras defined by

$$\mathbf{Mod}^{IN}_{OUT}(SP) \stackrel{\text{def}}{=} \mathbf{Cl}_{\underset{OUT}{\overset{IN}{\longleftrightarrow}}}(\mathbf{Mod}(SP))$$

Two Σ-specifications $SP1$, $SP2$ are *(IN, OUT)-behaviourally equivalent* (written $SP1 \underset{OUT}{\overset{IN}{\longleftrightarrow}} SP2$) if

$$\mathbf{Mod}^{IN}_{OUT}(SP1) = \mathbf{Mod}^{IN}_{OUT}(SP1)$$

A Σ-specification SP is *(IN, OUT)-behaviourally closed* if

$$\mathbf{Mod}(SP) = \mathbf{Mod}^{IN}_{OUT}(SP)$$

End Definition.

In chapter 1 we informally characterised behavioural closure as follows:

> If a program module implements a specification then so should all behaviourally equivalent program modules.

"Translating" this into the usual terminology (i.e. that used throughout this report), that is

> If a specification *SP* is behaviourally closed then whenever a specification *SP1* implements *SP*, all behaviourally equivalent specifications *SP2* also implement *SP*.

The following lemma confirms that (whichever equivalence we choose) this statement is true.

Lemma 3.14 $(\mathit{SP1} \equiv \mathit{SP2} \;\Rightarrow\; (\mathit{SP} \rightsquigarrow \mathit{SP1}) \Leftrightarrow (\mathit{SP} \rightsquigarrow \mathit{SP2}))$

Let Σ be a signature, $\equiv : \mathbf{Alg}(\Sigma) \leftrightarrow \mathbf{Alg}(\Sigma)$ an equivalence and *SP* a Σ-specification.

If *SP* is closed with respect to $\equiv$ then for any Σ-specifications *SP1* and *SP2*, such that $\mathit{SP1} \equiv \mathit{SP2}$,

$$(\mathit{SP} \rightsquigarrow \mathit{SP1}) \Leftrightarrow (\mathit{SP} \rightsquigarrow \mathit{SP2})$$

Proof

The proof is by straightforward calculation:

$(\mathit{SP} \rightsquigarrow \mathit{SP1}) \Leftrightarrow (\mathit{SP} \rightsquigarrow \mathit{SP2})$
$=$ $\quad\{$ definition of $\rightsquigarrow$ $\}$
$(\mathbf{Mod}(\mathit{SP}) \supseteq \mathbf{Mod}(\mathit{SP1})) \Leftrightarrow (\mathbf{Mod}(\mathit{SP}) \supseteq \mathbf{Mod}(\mathit{SP2}))$
$=$ $\quad\{$ closure of *SP* $\}$
$(\mathbf{Cl}_{\equiv}(\mathbf{Mod}(\mathit{SP})) \supseteq \mathbf{Mod}(\mathit{SP1})) \Leftrightarrow (\mathbf{Cl}_{\equiv}(\mathbf{Mod}(\mathit{SP})) \supseteq \mathbf{Mod}(\mathit{SP2}))$
$=$ $\quad\{\ \mathbf{Cl}_{\equiv}(A) \supseteq B \;\Leftrightarrow\; \mathbf{Cl}_{\equiv}(A) \supseteq \mathbf{Cl}_{\equiv}(B)\ \}$
$(\mathbf{Cl}_{\equiv}(\mathbf{Mod}(\mathit{SP})) \supseteq \mathbf{Cl}_{\equiv}(\mathbf{Mod}(\mathit{SP1}))) \Leftrightarrow (\mathbf{Cl}_{\equiv}(\mathbf{Mod}(\mathit{SP})) \supseteq \mathbf{Cl}_{\equiv}(\mathbf{Mod}(\mathit{SP2})))$
$\Leftarrow$ $\quad\{$ Liebniz $\}$
$\mathbf{Cl}_{\equiv}(\mathbf{Mod}(\mathit{SP1})) = \mathbf{Cl}_{\equiv}(\mathbf{Mod}(\mathit{SP2}))$

$=$ { definition of $\equiv$ for specifications }
SP1 $\equiv$ *SP2*
□

End Lemma.

In the previous section, we saw that behavioural equivalence is a special case of a variety of standard equivalences used in the literature. It is therefore possible to use the behavioural abstraction operator **behaviour** _ **wrt** (_, _) to define several standard closure operations.[2] For example:

- In [30], Sannella and Tarlecki define the closure operator **isoclose** by

  $$\mathbf{Mod}(\mathbf{isoclose}\ SP) \stackrel{\text{def}}{=} \mathbf{Cl}_{\cong}(\mathbf{Mod}(SP))$$

  But lemma 3.7 shows that, for a signature Σ with sorts T, $\cong = \overset{T}{\underset{T}{\longleftrightarrow}}$. Therefore **isoclose** is just a special case of **behaviour** _ **wrt** (_, _).

  $$\mathbf{isoclose}\ SP \stackrel{\text{def}}{=} \mathbf{behaviour}\ SP\ \mathbf{wrt}\ (T, T)$$

  where $T = \mathbf{Tp}(\mathbf{Sig}(SP))$

- In [30], Sannella and Tarlecki define the closure operator **junk** by

  $$\mathbf{Mod}(\mathbf{junk}\ \ IN\ \mathbf{on}\ SP) \stackrel{\text{def}}{=} \mathbf{Cl}_{\sim}(\mathbf{Mod}(SP))$$

  where

  $$\mathcal{A} \sim \mathcal{B} \stackrel{\text{def}}{=} \mathcal{R}(\Sigma, IN, \mathcal{A}) \cong \mathcal{R}(\Sigma, IN, \mathcal{B})$$

  But lemma 3.11 shows that, for a signature Σ with sorts T, $\sim = \overset{IN}{\underset{T}{\longleftrightarrow}}$. Therefore **junk** is just a special case of **behaviour** _ **wrt** (_, _).

  $$\mathbf{junk}\ \ IN\ \mathbf{on}\ SP \stackrel{\text{def}}{=} \mathbf{behaviour}\ SP\ \mathbf{wrt}\ (IN, T)$$

  where $T = \mathbf{Tp}(\mathbf{Sig}(SP))$

[2]These operations are not used in the sequel — they are included to demonstrate that the generality provided in behavioural abstraction is a useful operation.

- In [30] Sannella and Tarlecki define the operator **restrict _ to _** by

$$\textbf{Mod}(\textbf{restrict } SP \textbf{ to } IN) \overset{\text{def}}{=} \{\mathcal{A}\colon \mathcal{A} \in \textbf{Mod}(SP)\colon \mathcal{R}(\textbf{Sig}(SP), IN, \mathcal{A})\}$$

  This operator removes the junk from each model of *SP*.

  A similar operation can be defined using the **junk _ on _** operator.

$$\textbf{restrict } SP \textbf{ to } IN \overset{\text{def}}{=} \textbf{reachable (junk } IN \textbf{ on } SP) \textbf{ on } IN$$

  The use of the **junk _ on _** operator adds models with differ only with respect to non-junk values and, in particular, adds reachable models; the use of the **reachable** operator removes the unreachable models leaving the reachable subalgebras of *SP*.

  Our version of the operation differs slightly in that, even if *SP* is not closed under isomorphism, the result (under our definition) will be closed under isomorphism whereas it may not be with the original definition.

3.5 Observational Axioms

So far we have considered behavioural equivalence from a very "model-theoretic" viewpoint — defining the relation in terms of homomorphisms. This is useful because it is clear how this style of definition can be used to derive a notion of behavioural equivalence for program modules. (See Hoare [13] for a demonstration).

This section examines behavioural equivalence from a more "axiomatic" viewpoint: it identifies a set $\textbf{Axm}(IN, OUT)$ of axioms such that, for any axiom $ax \in \textbf{Axm}(IN, OUT)$.

$$\mathcal{A} \models ax \;\;\Leftrightarrow\;\; \mathcal{B} \models ax$$

if $\mathcal{A} \underset{OUT}{\overset{IN}{\longleftrightarrow}} \mathcal{B}$. That is, $\textbf{Axm}(IN, OUT)$ is a set of axioms whose satisfaction is invariant under behavioural equivalence.

Schoett considers almost exactly this question in the introduction to [37]. (The main differences are that, like Meseguer and Goguen, Schoett considers the special case $IN = OUT$ and that Schoett does not prove that he achieves his goal.)

Schoett argues that, in an "observational specification" (that is a behaviourally closed specification) one should disallow equations between terms of unobservable sorts since such equations "demand equality of representation values that is not relevant to (nor even observable by) a user" [37 p. 601]. He also argues that one should use reachable quantification instead of (plain) quantification since unreachable values "cannot be generated with the user operations, and hence [their] existence and [their] properties are not relevant to the users" [37 p. 602].

Following this lead, we define the set $\mathbf{Axm}(IN, OUT)$ of "observational axioms" to be the set of all axioms using only equations over sorts in OUT and reachable quantification with respect to the sorts IN. That is,

Definition 3.15 (observational formulæ and axioms)

Let Σ be a signature, let IN and OUT be subsets of the sort symbols in Σ.

The set $WFF(IN, OUT)$ of well-formed (IN, OUT)-observational Σ-formulæ is the least subset of $WFF(\Sigma)$ satisfying

$$\begin{array}{lll}
\mathit{true} & \in WFF(IN, OUT) & \\
t1 =_\tau t2 & \in WFF(IN, OUT) & \text{if } \tau \in OUT \text{ and } t1, t2 \in W(\Sigma, X)_\tau \\
\neg P & \in WFF(IN, OUT) & \text{if } P \in WFF(IN, OUT) \\
P \wedge Q & \in WFF(IN, OUT) & \text{if } P \in WFF(IN, OUT) \text{ and } Q \in WFF(IN, OUT) \\
\forall^{\Sigma'}_{T'} x{:}\tau.\ P & \in WFF(IN, OUT) & \text{if } P \in WFF(IN, OUT) \text{ and } T' \subseteq IN \\
\forall x{:}\tau.\ P & \in WFF(IN, OUT) & \text{if } P \in WFF(IN, OUT) \text{ and } \tau \in IN
\end{array}$$

We write $\mathbf{Axm}(IN, OUT)$ to denote the set of all Σ-axioms in $WFF(IN, OUT)$.

End Definition.

Theorem 3.16 $(\xleftrightarrow[OUT]{IN} \subseteq \equiv_{\mathbf{Axm}(IN,OUT)})$

Let Σ be a signature and IN and OUT subsets of the sorts of Σ.

If $\mathcal{A}$ and $\mathcal{B}$ are Σ-algebras and $ax \in \mathbf{Axm}(IN, OUT)$, then

$$\mathcal{A} \underset{OUT}{\overset{IN}{\longleftrightarrow}} \mathcal{B} \;\Rightarrow\; \mathcal{A} \models ax \Leftrightarrow \mathcal{B} \models ax$$

Proof

We begin by showing that if $h\colon \mathcal{A} \underset{OUT}{\overset{IN}{\rightarrow}} \mathcal{B}$ then, for each Σ-formula $\varphi \in WFF(IN, OUT)$ and valuation $v \in Val(\mathcal{A}, \varphi)$,

$$\mathcal{A} \models_v \varphi \Leftrightarrow \mathcal{B} \models_{h \cdot v} \varphi$$

The proof is by induction over the structure of φ.

Base case: ($\varphi = true$)

For any valuation $v \in \mathbf{Val}(\mathcal{A}, \varphi)$.

$\mathcal{A} \models_v true$
$=$ { definition of $\models$ }
$true$
$=$ { definition of $\models$ }
$\mathcal{B} \models_{h \cdot v} true$
□

Base case: ($\varphi = t1 =_\tau t2$ and $\tau \in OUT$)

This follows immediately from lemma 3.6.

Inductive step: ($\varphi = \neg P$)

Suppose that, for all $v \in \mathbf{Val}(\mathcal{A}, P)$, $\mathcal{A} \models_v P \Leftrightarrow \mathcal{B} \models_{h \cdot v} P$. Then,

$\mathcal{A} \models_v \neg P$
$=$ { definition of $\models \neg P$ }
$\neg \mathcal{A} \models_v P$
$=$ { inductive assumption }
$\neg \mathcal{B} \models_{h \cdot v} P$
□

Inductive step: ($\varphi = P \wedge Q$)

Suppose that, for all $v \in \mathbf{Val}(\mathcal{A}, P \wedge Q)$, $\mathcal{A} \models_v P \Leftrightarrow \mathcal{B} \models_{h \cdot v} P$ and $\mathcal{A} \models_v Q \Leftrightarrow \mathcal{B} \models_{f \cdot v} Q$. Then,

$\mathcal{A} \models_v P \wedge Q$
$=$ $\{$ definition of $\models P \wedge Q$ $\}$
$\mathcal{A} \models_v P \wedge \mathcal{A} \models_v Q$
$=$ $\{$ inductive assumption $\}$
$\mathcal{B} \models_{h \cdot v} P \wedge \mathcal{B} \models_{h \cdot v} Q$
$=$ $\{$ definition of $\models P \wedge Q$ $\}$
$\mathcal{B} \models_{h \cdot v} P \wedge Q$
$\square$

Inductive step: ($\forall x \colon \tau.\ P$ and $\tau \in IN$)

Suppose that, for all $v \in \mathbf{Val}(\mathcal{A}, P)$, $\mathcal{A} \models_v P \Leftrightarrow \mathcal{B} \models_{h \cdot v} P$. Then,

$\mathcal{A} \models_v \forall x \colon \tau.\ P$
$=$ $\{$ definition of $\models \forall$ $\}$
$(\forall a \colon a \in \mathcal{A}_\tau \colon \mathcal{A} \models_{v \cup \{x := a\}} P)$
$=$ $\{$ inductive assumption $\}$
$(\forall a \colon a \in \mathcal{A}_\tau \colon \mathcal{B} \models_{h \cdot (v \oplus \{x := a\})} P)$
$=$ $\{$ $\cdot$ distributes over $\oplus$ $\}$
$(\forall a \colon a \in \mathcal{A}_\tau \colon \mathcal{B} \models_{(h \cdot v) \oplus \{x := h(a)\})} P)$
$=$ $\{$ surjectivity of $h|_{IN}$ $\}$
$(\forall b \colon b \in \mathcal{B}_\tau \colon \mathcal{B} \models_{(h \cdot v) \oplus \{x := b\})} P)$
$=$ $\{$ definition of $\models \forall$ $\}$
$\mathcal{B} \models_{h \cdot v} \forall x \colon \tau.\ P$
$\square$

Inductive step: ($\forall^{\Sigma'}_{T'} x \colon \tau.\ P$ and $T' \subseteq IN$)

Suppose that, for all $v \in \mathbf{Val}(\mathcal{A}, P)$, $\mathcal{A} \models_v P \Leftrightarrow \mathcal{B} \models_{h \cdot v} P$. Then,

$\mathcal{A} \models_v \forall^{\Sigma'}_{T'} x \colon \tau.\ P$
$=$ $\{$ definition of $\models \forall$ $\}$
$(\forall a : a \in \mathcal{A}_\tau \wedge \mathcal{R}(\Sigma', T', a) : \mathcal{A} \models_{v \oplus \{x := a\}} P)$
$=$ $\{$ inductive assumption $\}$
$(\forall a : a \in \mathcal{A}_\tau \wedge \mathcal{R}(\Sigma', T', a) : \mathcal{B} \models_{h \cdot (v \oplus \{x := a\})} P)$
$=$ $\{$ $\cdot$ distributes over $\oplus$ $\}$
$(\forall a : a \in \mathcal{A}_\tau \wedge \mathcal{R}(\Sigma', T', a) : \mathcal{B} \models_{(h \cdot v) \oplus \{x := h(a)\}} P)$
$=$ $\{$ surjectivity of $h|_{IN}$ $\}$
$(\forall b : b \in \mathcal{B}_\tau \wedge \mathcal{R}(\Sigma', T', b) : \mathcal{B} \models_{(h \cdot v) \oplus \{x := b\}} P)$
$=$ $\{$ definition of $\models \forall$ $\}$
$\mathcal{B} \models_{h \cdot v} \forall^{\Sigma'}_{T'} x \colon \tau.\ P$
$\square$

Hence, by the principle of structural induction, we conclude that, for any formula $\varphi \in WFF(IN, OUT)$, and valuation $v \in \mathbf{Val}(\mathcal{A}, \varphi)$

$$\mathcal{A} \models_v \varphi \Leftrightarrow \mathcal{B} \models_{h \cdot v} \varphi$$

and hence, for any axiom $ax \in \mathbf{Axm}(IN, OUT)$,

$$\mathcal{A} \models ax \Leftrightarrow \mathcal{B} \models ax$$

To see that the same holds when $\mathcal{A} \overset{IN}{\underset{OUT}{\longleftrightarrow}} \mathcal{B}$, let $\equiv_{Ax}$ be defined as in definition 3.2 and write R^* for the least equivalence containing a binary relation R. The result follows by straightforward calculation:

true
$=$ { above }
$\overset{IN}{\underset{OUT}{\rightarrow}} \subseteq \equiv_{\{ax\}}$
$\Rightarrow$ { set theory }
$(\overset{IN}{\underset{OUT}{\rightarrow}})^* \subseteq (\equiv_{\{ax\}})^*$
$=$ { definition of $\overset{IN}{\underset{OUT}{\longleftrightarrow}}$ and $\equiv_{\{ax\}}$ is an equivalence }
$\overset{IN}{\underset{OUT}{\longleftrightarrow}} \subseteq \equiv_{\{ax\}}$
□

Hence, for any Σ-algebras $\mathcal{A}$ and $\mathcal{B}$ and any observational axioms $ax \in \mathbf{Axm}(IN, OUT)$,

$$\mathcal{A} \overset{IN}{\underset{OUT}{\longleftrightarrow}} \mathcal{B} \ \Rightarrow\ \mathcal{A} \models ax \Leftrightarrow \mathcal{B} \models ax$$

End Theorem.

An immediate corollary is that a specification satisfies exactly the same set of "observations" as does its behavioural closure.

Corollary 3.17 ($SP \models ax \Leftrightarrow$ **behaviour** SP **wrt** $(IN, OUT) \models ax$)

Let Σ be a signature, IN and OUT subsets of the sorts of Σ and SP a Σ-specification.

For any axiom $ax \in \mathbf{Axm}(IN, OUT)$,

$$SP \models ax \quad \text{iff} \quad \textbf{behaviour } SP \textbf{ wrt } (IN, OUT) \models ax$$

Proof

behaviour SP **wrt** $(IN, OUT) \models ax$
$=$ $\{$ definition $\models$ for specifications $\}$
$(\forall \mathcal{A}{:}\, \mathcal{A}{:}\, \textbf{behaviour } SP \textbf{ wrt } (IN, OUT){:}\, \mathcal{A} \models ax)$
$=$ $\{$ definition $\mathcal{A}{:}\, \textbf{behaviour } SP \textbf{ wrt } (IN, OUT)$, set theory $\}$
$(\forall \mathcal{A}, \mathcal{A}'{:}\, \mathcal{A}{:}\, SP \wedge \mathcal{A} \underset{OUT}{\overset{IN}{\longleftrightarrow}} \mathcal{A}'{:}\, \mathcal{A}' \models ax)$
$=$ $\{$ theorem 3.16 $\}$
$(\forall \mathcal{A}, \mathcal{A}'{:}\, \mathcal{A}{:}\, SP \wedge \mathcal{A} \underset{OUT}{\overset{IN}{\longleftrightarrow}} \mathcal{A}'{:}\, \mathcal{A} \models ax)$
$=$ $\{$ set theory $\}$
$(\forall \mathcal{A}{:}\, \mathcal{A}{:}\, SP{:}\, \mathcal{A} \models ax)$
$=$ $\{$ definition of $\models$ for specifications $\}$
$SP \models ax$
□

End Corollary.

An interesting corollary to theorem 3.16 is that $\underset{OUT}{\overset{IN}{\longleftrightarrow}}$ is stronger than $\equiv_{EQ(IN,OUT)}$. We repeat two results from [27] before stating and proving the corollary.

The first result states that the satisfaction of a set $Cl(\phi)$ of axioms is invariant under $\equiv_\phi$.

Lemma 3.18 ($\equiv_\phi \subseteq \equiv_{Cl(\phi)}$ ([27 Fact 18].))

Let Σ be a signature, and ϕ a set of Σ-formulæ.

Then, for any two Σ-algebras $\mathcal{A}$ and $\mathcal{B}$, and axiom $ax \in Cl(\phi)$,

$$\mathcal{A} \equiv_\phi \mathcal{B} \;\Rightarrow\; \mathcal{A} \models ax \Leftrightarrow \mathcal{B} \models ax$$

where $Cl(\phi)$ is defined to be the closure of ϕ under negation, conjunction, equivalence and uniform quantification, that is, $\varphi \in Cl(\phi)$ implies $\forall xs{:}\, \tau s.\ \varphi \in Cl(\phi)$ and $\exists xs{:}\, \tau s.\ \varphi \in Cl(\phi)$ where $xs = \textbf{vars}(\varphi)$.

End Lemma.

Note that, because of the restriction to uniform quantification, $Cl(EQ(IN, OUT))$ is a proper subset of $\textbf{Axm}(IN, OUT)$. In particular, $Cl(EQ(IN, OUT))$ does not contain any formulæ of the form

$$\forall x{:}\, \tau.\ \exists y{:}\, \tau.\ \varphi$$

The second result is that $\equiv_{EQ(IN,OUT)} \not\subseteq \equiv_{\mathbf{Axm}(IN,OUT)}$. Sannella and Tarlecki prove this with the aid of the following counterexample.

Counterexample 3.19 ($\equiv_{EQ(IN,OUT)} \not\subseteq \equiv_{\mathbf{Axm}(IN,OUT)}$)

Let Σ = **sign** *Rat*, *Bool* :**type** , <: *Rat* × *Rat* → *Bool* , True :→ *Bool* **end**. Let $\mathcal{A}$ and $\mathcal{B}$ be algebras with *Rat* being, respectively, the open and closed interval $0 \dots 1$ of rational numbers. That is,

$$\mathcal{A} \overset{\text{def}}{=} \langle \begin{array}{l} Bool = \{0, 1\} \\ \text{True} = 1 \\ Rat = \{r\colon 0 < r < 1\colon r\} \\ r1 < r2 = r1 < r2 \end{array} \rangle \quad \text{and} \quad \mathcal{B} \overset{\text{def}}{=} \langle \begin{array}{l} Bool = \{0, 1\} \\ \text{True} = 1 \\ Rat = \{r\colon 0 \le r \le 1\colon r\} \\ r1 < r2 = r1 < r2 \end{array} \rangle$$

To see that $\mathcal{A} \le_{EQ(\{Rat\},\{Bool\})} \mathcal{B}$, consider any valuation $va \in \mathbf{Val}(\mathcal{A})$ and let $vb \overset{\text{def}}{=} va$ and observe that

$$\mathcal{A} \models_{va} (r1 < r2) = \text{True} \quad \text{iff} \quad \mathcal{B} \models_{vb} (r1 < r2) = \text{True}$$

To see that $\mathcal{B} \le_{EQ(\{Rat\},\{Bool\})} \mathcal{A}$, consider any valuation $vb \in \mathbf{Val}(\mathcal{B})$ and let $va \overset{\text{def}}{=} f \cdot vb$ where $f(r) \overset{\text{def}}{=} \frac{(r+1)}{3}$. Again, observe that

$$\mathcal{A} \models_{va} (r1 < r2) = \text{True} \quad \text{iff} \quad \mathcal{B} \models_{vb} (r1 < r2) = \text{True}$$

Thus, $\mathcal{A} \equiv_{EQ(\{Rat\},\{Bool\})} \mathcal{B}$.

However, it is easy to show that

$$\mathcal{A} \not\equiv_{\mathbf{Axm}(\{Rat\},\{Bool\})} \mathcal{B}$$

Consider the observational axiom $\forall x\colon Rat.\ \exists y\colon Rat.\ (y < x) = \text{True}$ which asserts that there is no smallest *Rat*. It is easy to show that

$$\mathcal{A} \models \forall x\colon Rat.\ \exists y\colon Rat.\ (y < x) = \text{True}$$

(consider $y = x \div 2$) but that

$$\mathcal{B} \not\models \forall x\colon Rat.\ \exists y\colon Rat.\ (y < x) = \text{True}$$

(consider $x = 0$).

Hence, $\mathcal{A} \equiv_{EQ(\{Rat\},\{Bool\})} \mathcal{B} \not\Rightarrow \mathcal{A} \equiv_{\mathbf{Axm}(\{Rat\},\{Bool\})} \mathcal{B}$.

End Counterexample.

An immediate corollary is that the relation $\underset{OUT}{\overset{IN}{\longleftrightarrow}}$ is strictly stronger than $\equiv_{EQ(IN,OUT)}$.

Corollary 3.20 ($\underset{OUT}{\overset{IN}{\longleftrightarrow}} \subset \equiv_{EQ(IN,OUT)}$)

Let Σ be a signature, *IN* and *OUT* subsets of the sorts of Σ and $\mathcal{A}$ and $\mathcal{B}$ two Σ-algebras.

Then, by lemma 3.6,

$$\mathcal{A} \underset{OUT}{\overset{IN}{\longleftrightarrow}} \mathcal{B} \;\Rightarrow\; \mathcal{A} \equiv_{EQ(IN,OUT)} \mathcal{B}$$

but, by counterexample 3.19,

$$\mathcal{A} \equiv_{EQ(IN,OUT)} \mathcal{B} \;\not\Rightarrow\; \mathcal{A} \underset{OUT}{\overset{IN}{\longleftrightarrow}} \mathcal{B}.$$

End Corollary.

The above inequality presents us with a dilemma. If the two general cases of behavioural equivalence (that is, $\underset{OUT}{\overset{IN}{\longleftrightarrow}}$ and $\equiv_{EQ(IN,OUT)}$) were equivalent, we could be confident that the two definitions were appropriate generalisations; since the definitions are not equivalent, at most one definition is appropriate (or, at least, they are appropriate for different tasks).

For our purposes, our generalisation of Meseguer and Goguen's behavioural equivalence (that is, $\underset{OUT}{\overset{IN}{\longleftrightarrow}}$) is appropriate — in chapter 4 we use $\underset{OUT}{\overset{IN}{\longleftrightarrow}}$ to give a precise semantics to Wirsing and Broy's ultraloose specification style.

We also find Sannella and Tarlecki's restriction to uniform quantification in lemma 3.18 rather "untidy" in comparision to theorem 3.16.

However, Sannella and Tarlecki's definition depends only on the notions of axiom, model and satisfaction provided by a logical framework and so is easily generalised whereas our definition depends on a rather delicate choice of relationship between models and so is considerably harder to generalise. This is a significant advantage of their definition — if such generality is required.

Before leaving the topic of observational axioms, we note the following open question:

> It is straightforward to show that
>
> $$\underset{OUT}{\overset{IN}{\longleftrightarrow}} \;=\; \equiv_{\mathbf{Axm}(IN,OUT)}$$
>
> if $IN = \emptyset$. It is interesting to ask whether the same holds if IN is non-empty.

Such a result is important for two reasons:

1. Showing that the two quite different styles of definition are equivalent would improve confidence that our notion of behavioural equivalence is appropriate.

2. It would allow us to prove results of the form

   $$\textbf{behaviour } SP \textbf{ wrt } (IN, OUT) \rightsquigarrow SP'$$

   by showing that, for all axioms $ax \in \mathbf{Axm}(IN, OUT)$,

   $$SP \models ax \Leftrightarrow SP' \models ax$$

   Hennicker [11,12] describes a proof technique he calls "context induction" for showing results of this kind. (We shall not explore this in detail since it requires a rather unusual notion of behavioural equivalence where the set of observable values is defined axiomatically.)

Lemma 3.16 shows that

$$\overset{IN}{\underset{OUT}{\longleftrightarrow}} \subseteq \equiv_{\mathbf{Axm}(IN,OUT)}$$

We *believe* (but have not tried to prove) that the reverse only holds in the presence of at most countably many unreachable elements.

3.6 Schoett's Impossibility Theorem

The introduction to this thesis suggests that it is methodologically important to write behaviourally closed specifications since this ensures that the following closure property holds.

> If a program module implements a specification then so should all behaviourally equivalent program modules.

Since theorem 3.16 shows that any flat specification containing only observational axioms will be behaviourally closed it seems that we need only restrict ourselves to observational axioms to ensure that our specifications are behaviourally closed.

However, Schoett [36,37] has shown that observational axioms alone are too weak to be useful. This section outlines his result.

* *

Schoett considers "counter algebras". That is, models of the specification *Counter* (figure 3.1).

Schoett shows that there is no *finite* set Ax of observational axioms such that all algebras satisfying Ax are behaviourally equivalent (with respect to *Bool*) to a model of *Counter*. (In the following it is sufficient to consider a single finite axiom since every finite set of observational axioms is equivalent to a single observational axiom, namely their conjunction.)

```
Counter ≝ enrich BoolBase
          by sign  Ctr: type
                   zero: → Ctr
                   inc, dec: Ctr → Ctr
                   isZero: Ctr → Bool
             axioms  dec(zero) =_Ctr zero
                     ∀c: ctr. dec(inc(c)) =_Ctr c
                     isZero(zero) =_Bool True
                     ∀c: ctr. isZero(inc(c)) =_Bool False
          end
```

Figure 3.1: Counter Specification

Theorem 3.21 (Schoett's impossibility result [37 theorem 3.4])

For every algebra $\mathcal{A}$: *Counter* and observational axiom $ax \in \mathbf{Axm}(\{Bool\}, \{Bool\})$ such that $\mathcal{A} \models ax$, there is a **Sig**(*Counter*)-algebra $\mathcal{B}$ such that $\mathcal{B} \models ax$ but $\mathcal{A}$ and $\mathcal{B}$ are not $(\{Bool\}, \{Bool\})$-behaviourally equivalent.

In particular, there exists $n \geq 0$ such that for all terms c composed of *zero*, *inc* and *dec*,

$$\mathcal{B} \models isZero(dec^n(c)) =_{Bool} \text{False}$$

(that is, $\mathcal{B}$ cannot count higher than n).

End Theorem.

Schoett proves this theorem by showing how, for any algebra $\mathcal{A}$: *Counter* and any number i, one can define an algebra $\mathcal{A}^{(i)}$ differing in its interpretation of numbers above i. That is, such that,

$$inc^j(zero)_{\mathcal{A}^{(i)}} = \begin{cases} inc^j(zero)_{\mathcal{A}}, & \text{if } j < i\text{; and} \\ inc^i(zero)_{\mathcal{A}}, & \text{if } j \geq i. \end{cases}$$

(Obviously, $\mathcal{A}$ and $\mathcal{A}^{(i)}$ are not $(\{Bool\}, \{Bool\})$-behaviourally equivalent — consider the axiom $isZero(dec^{i+1}(inc^{i+2}(zero))) =_{Bool}$ True.)

He then shows that, for any $\mathcal{A}$: *Counter* and any finite observational axiom $ax \in$ $\mathbf{Axm}(\{Bool\}, \{Bool\})$,

$$\mathcal{A} \models ax \Leftrightarrow \mathcal{A}^{(k(ax))} \models ax$$

where $k(ax)$ is the maximum number of occurrences of the symbol *dec* in a term in ax.

For example, consider the algebra

$$\begin{array}{lll} \mathcal{A} \stackrel{\text{def}}{=} & \langle & Bool = \{0, 1\} \\ & & Ctr = \{0, 1, 2, \ldots\} \\ & & \text{True} = 1 \\ & & \text{False} = 0 \\ & & zero = 0 \\ & & inc(x) = x + 1 \\ & & dec(c) = if\ x > 0\ then\ x - 1\ else\ 0 \\ & & isZero(x) = if\ x > 0\ then\ 0\ else\ 1 \\ & \rangle & \end{array}$$

and the axiom

$$ax \stackrel{\text{def}}{=} \forall_{\emptyset}^{\{zero, inc, dec\}} c\colon Ctr.\ isZero(dec(c)) = \text{False} \Rightarrow isZero(c) = \text{False}$$

In this case, $k(ax) = 1$ and $\mathcal{A}^{(1)}$ is defined by

$$\begin{array}{lll} \mathcal{A}^{(1)} \stackrel{\text{def}}{=} & \langle & Bool = \{0, 1\} \\ & & Ctr = \{0, 1, 2, \ldots\} \\ & & \text{True} = 1 \\ & & \text{False} = 0 \\ & & zero = 0 \\ & & inc(x) = if\ x > 1\ then\ x\ else\ x + 1 \\ & & dec(c) = if\ x > 0\ then\ x - 1\ else\ 0 \\ & & isZero(x) = if\ x > 0\ then\ 0\ else\ 1 \\ & \rangle & \end{array}$$

Obviously both $\mathcal{A}$ and $\mathcal{A}^{(1)}$ satisfy ax.

□

An immediate corollary to theorem 3.21 is that the class of all algebras having the same behaviour as a counter cannot be specified by a finite set of observational axioms.

Corollary 3.22 (weakness of observational axioms)

There is no finite set $Ax = \{ax1, \ldots axm\} \subseteq \mathbf{Axm}(\{Bool\}, \{Bool\})$ such that

$$\langle \mathbf{Sig}(Counter), Ax\rangle \;=\; \textbf{behaviour } Counter \textbf{ wrt } (\{Bool\}, \{Bool\})$$

Proof

Suppose that $\mathcal{A} \models Ax \Rightarrow \mathcal{A} \in \mathbf{Mod}^{\{Bool\}}_{\{Bool\}}(Counter)$. Then, from the definition of $\mathbf{Mod}^{\{Bool\}}_{\{Bool\}}$ and corollary 3.17, we have that

$$\mathcal{A}\colon Counter \;\Leftrightarrow\; \mathcal{A} \models Ax$$

Since *Counter* is consistent, it follows that there is a model $\mathcal{A}$ such that

$$\mathcal{A}\colon Counter \wedge \mathcal{A} \models Ax$$

But, by theorem 3.21,

$$\mathcal{A}^{(k(ax1 \wedge \ldots axm))} \models Ax$$

Since $\mathcal{A}^{(i)}$ does not satisfy the observational axiom

$$isZero(dec^{i+1}(inc^{i+2}(zero))) =_{Bool} \text{True}$$

(where $i = k(ax1 \wedge \ldots axm)$), $\mathcal{A}^{(i)}$ cannot be a member of $\mathbf{Mod}^{IN}_{OUT}(Counter)$. It follows that

$$\mathcal{A}^{(k(ax1 \wedge \ldots axm))}\colon \langle \mathbf{Sig}(Counter), Ax\rangle$$
$$\not\Leftrightarrow \;\; \mathcal{A}^{(k(ax1 \wedge \ldots axm))}\colon \textbf{behaviour } Counter \textbf{ wrt } (\{Bool\}, \{Bool\})$$

End Corollary.

Since no flat specification consisting of observational axioms can specify the set of counter-like algebras, we conclude that any language based on observational axioms alone will be too weak to be useful.

3.7 Summary

This chapter defines the notion of behavioural equivalence used in this thesis and compares it with two significant alternative definitions — showing that our definition is a generalisation of that of Meseguer and Goguen and slightly stronger than that of Sannella and Tarlecki.

It also defines the notion of observational axiom and shows that satisfaction of observational axioms is invariant under behavioural equivalence but shows that, although a specification language which permits only observational axioms would result in behaviourally closed specifications, the language would be too weak to be useful.

Chapter 4

Ultraloose Specifications

In chapter 3 we presented a subset of $\mathbf{Axm}(\Sigma)$ which has the property that any (flat) specification written using these axioms will be behaviourally closed. Sadly, Schoett has shown that an infinite number of these "observational axioms" are required to specify something as simple as a counter.

This chapter describes a slightly indirect way of writing specifications used by Wirsing and Broy [2,42] which we call "ultraloose style" (section 4.1) and characterises the semantic effect of this style in three previously unpublished theorems.

- Theorem 4.5 shows that specifications written in the ultraloose style ("ultraloose specifications") are downward closed under $\xrightarrow[OUT]{IN}$.

  To our knowledge, this is the first attempt to relate the (syntactic) ultraloose specification style to a semantic concept such as a behavioural ordering.

- Theorem 4.7 shows that ultraloose specifications are closed under $\xleftrightarrow[OUT]{IN}$ provided they contain no inequations.

  This result is interesting because it provides specifiers with a precise methodology for developing behaviourally closed USL specifications.

- Theorem 4.10 uses theorem 4.7 to demonstrate that any ultraloose specification SP^{IN}_{OUT} is semantically equivalent to the corresponding ASL specification

behaviour SP **wrt** (IN, OUT) if SP contains no inequations or existential quantification.

As well as providing a precise characterisation of the semantic effect of adopting the ultraloose specification style, this result shows that, contrary to what Schoett's "impossibility theorem" [37] (discussed in section 3.5) *seems to suggest* it is possible to write useful behaviourally closed specifications in first-order logic — we explain why there is no conflict.

We consider theorem 4.10 to be the most important result in this thesis: it precisely characterises the semantic consequences of the ultraloose style; and it precisely describes the relationship between ASL and USL.

4.1 Defining Ultraloose Style

The "ultraloose style" of specification has two distinctive characteristics:

1. The use of reachable quantification ($\forall^{\Sigma}_{IN}$ instead of $\forall$); and
2. The use of a Σ-congruence $\equiv$ instead of equality.

[Later sections use these properties and lemmas 3.8 (which relates reachability to $\underset{OUT}{\overset{IN}{\rightarrow}}$) and 3.10 (which relates congruences to $\underset{OUT}{\overset{IN}{\rightarrow}}$) to relate this style to the relations $\underset{OUT}{\overset{IN}{\rightarrow}}$ and $\underset{OUT}{\overset{IN}{\longleftrightarrow}}$.]

Figure 4.1 gives a simple example of an "ultraloose specification" (that is a specification written in the ultraloose style). It is identical to the specification in figure 1.3 except that it is more explicit about the quantification and the specifications of *Nat* and *Bool* have been expanded.

For comparision, figure 4.2 contains a more usual specification of a stack. The main differences between the two specifications are:

$\mathit{Stack'} \overset{\text{def}}{=}$
export $\mathit{StackSig}$ **from**
enrich Bool **by**
spec sign $\mathit{Nat}, \mathit{Stack}\colon$ **type**

$0\colon \rightarrow \mathit{Nat}$
$\mathit{succ}\colon \mathit{Nat} \rightarrow \mathit{Nat}$
$\mathit{empty}\colon \rightarrow \mathit{Stack}$
$\mathit{push}\colon \mathit{Nat} \times \mathit{Stack} \rightarrow \mathit{Stack}$
$\mathit{pop}\colon \mathit{Stack} \rightarrow \mathit{Stack}$
$\mathit{top}\colon \mathit{Stack} \rightarrow \mathit{Nat}$
$\mathit{isEmpty}\colon \mathit{Stack} \rightarrow \mathit{Bool}$
$\equiv\colon \mathit{Stack} \times \mathit{Stack} \rightarrow \mathit{Bool}$

axioms $\forall^{\{empty,pop,push\}}_{\{Nat\}} s\colon \mathit{Stack}, x\colon \mathit{Nat}.\ \mathit{top}(\mathit{push}(x,s)) =_{\mathit{Nat}} x$
$\forall^{\{empty,pop,push\}}_{\{Nat\}} s\colon \mathit{Stack}, x\colon \mathit{Nat}.\ \mathit{pop}(\mathit{push}(x,s)) \equiv s$
$\mathit{isEmpty}(\mathit{empty}) = \text{True}$
$\forall^{\{empty,pop,push\}}_{\{Nat\}} s\colon \mathit{Stack}, x\colon \mathit{Nat}.\ \mathit{isEmpty}(\mathit{push}(x,s)) = \text{False}$

$\forall m\colon \mathit{Nat}.\ \mathit{succ}(m) \neq_{\mathit{Nat}} 0$
$\forall m, n\colon \mathit{Nat}.\ \mathit{succ}(m) =_{\mathit{Nat}} \mathit{succ}(n) \Rightarrow m =_{\mathit{Nat}} n$
$\forall m\colon \mathit{Nat}.\ \exists^{\{0,succ\}}_{\emptyset} n\colon \mathit{Nat}.\ m =_{\mathit{Nat}} n$

$\forall s\colon \mathit{Stack}.\ s \equiv s$
$\forall s1, s2\colon \mathit{Stack}.\ s1 \equiv s2 \Leftrightarrow s2 \equiv s1$
$\forall s1, s2, s3\colon \mathit{Stack}.\ s1 \equiv s2 \wedge s2 \equiv s3 \Rightarrow s1 \equiv s3$

$\mathit{empty} \equiv \mathit{empty}$
$\forall s1, s2\colon \mathit{Stack}, x\colon \mathit{Nat}.\ s1 \equiv s2 \Rightarrow \mathit{push}(x, s1) \equiv \mathit{push}(x, s2)$
$\forall s1, s2\colon \mathit{Stack}.\ s1 \equiv s2 \Rightarrow \mathit{pop}(s1) \equiv \mathit{pop}(s2)$
$\forall s1, s2\colon \mathit{Stack}.\ s1 \equiv s2 \Rightarrow \mathit{top}(s1) = \mathit{top}(s2)$
$\forall s1, s2\colon \mathit{Stack}.\ s1 \equiv s2 \Rightarrow \mathit{isEmpty}(s1) = \mathit{isEmpty}(s2)$
end

Figure 4.1: An Ultraloose Stack Specification

- The first four axioms in figure 4.1 (which specify operations on *Stack*) are obtained from the first four axioms in figure 4.2 by replacing (plain) quantification by reachable quantification and replacing equations $t1 =_{Stack} t2$ by $t1 \equiv t2$.

- The next three axioms in figure 4.1 (which specify the operations on *Nat*) are identical to the last three axioms in figure 4.2.

- The next two axioms in figure 4.1 (which specify the sort *Bool*) are the standard axioms specifying the booleans. (Since the specification language introduced in chapter 2 does not provide a way of specifying new predicates, we *model* predicates by functions with result type *Bool* and we distinguish the atomic formulæ *true* and *false* from the constant function symbols *True* and *False*.)

- The remaining axioms in figure 4.1 specify that $=_{Nat}$, $=_{Bool}$ and $\equiv_{Stack}$ together make up a congruence.

The remainder of this section defines how to *transform* a "normal specification" *SP* into an "ultraloose specification" SP^{IN}_{OUT}. We begin with the axioms that specify the congruence $\equiv$.

In the above example, (a characteristic function representing) a relation $\equiv_\tau$ was specified only for the sorts τ which were not directly observable (i.e. for *Stack*). To simplify the specification of $\equiv$, it is convenient to define $\equiv_\tau$ for all sorts τ in Σ and define a set $Equality(OUT)$ of axioms which specify that $(\equiv_\tau) = (=)$ if $\tau \in OUT$.

Definition 4.1 (congruence axioms)

Let Σ be a signature with sorts T and OUT a subset of T.

The "OUT-ultraloose signature" Σ_{OUT} is defined by

$$\Sigma_{OUT} \overset{\text{def}}{=} \Sigma \cup \mathbf{Sig}(Bool) \cup \{\tau : \tau \in T : (\equiv_\tau : \tau \times \tau \to Bool)\}$$

$$
\begin{array}{lll}
\multicolumn{3}{l}{\mathit{Stack} \stackrel{\text{def}}{=}} \\
\textbf{spec} & \textbf{sign} & \mathit{Nat}, \mathit{Bool}, \mathit{Stack}\colon \textbf{type} \\
 & & \mathit{0}\colon \to \mathit{Nat} \\
 & & \mathit{succ}\colon \mathit{Nat} \to \mathit{Nat} \\
 & & \text{True}, \text{False}\colon \to \mathit{Bool} \\
 & & \mathit{empty}\colon \to \mathit{Stack} \\
 & & \mathit{push}\colon \mathit{Nat} \times \mathit{Stack} \to \mathit{Stack} \\
 & & \mathit{pop}\colon \mathit{Stack} \to \mathit{Stack} \\
 & & \mathit{top}\colon \mathit{Stack} \to \mathit{Nat} \\
 & & \mathit{isEmpty}\colon \mathit{Stack} \to \mathit{Bool} \\
 & \textbf{axioms} & \forall s\colon \mathit{Stack}, x\colon \mathit{Nat}.\ \mathit{top}(\mathit{push}(x, s)) =_{\mathit{Nat}} x \\
 & & \forall s\colon \mathit{Stack}, x\colon \mathit{Nat}.\ \mathit{pop}(\mathit{push}(x, s)) =_{\mathit{Stack}} s \\
 & & \mathit{isEmpty}(\mathit{empty}) =_{\mathit{Bool}} \text{True} \\
 & & \forall s\colon \mathit{Stack}, x\colon \mathit{Nat}.\ \mathit{isEmpty}(\mathit{push}(x, s)) =_{\mathit{Bool}} \text{False} \\
 & & \\
 & & \text{True} \neq \text{False} \\
 & & \forall x\colon \mathit{Bool}.\ x = \text{True} \vee x = \text{False} \\
 & & \\
 & & \forall m\colon \mathit{Nat}.\ \mathit{succ}(m) \neq_{\mathit{Nat}} \mathit{0} \\
 & & \forall m, n\colon \mathit{Nat}.\ \mathit{succ}(m) =_{\mathit{Nat}} \mathit{succ}(n) \Rightarrow m =_{\mathit{Nat}} n \\
 & & \forall m\colon \mathit{Nat}.\ \exists_{\emptyset}^{\{\mathit{0}, \mathit{succ}\}} n\colon \mathit{Nat}.\ m =_{\mathit{Nat}} n \\
\textbf{end} & &
\end{array}
$$

Figure 4.2: A "Normal" Stack Specification

The set of axioms $Equiv_\tau$ that specify that (the relation with characteristic function) $\equiv_\tau$ is an equivalence is defined by

$$\begin{array}{lll} Equiv_\tau \overset{\text{def}}{=} \{ & \forall x{:}\,\tau.\ \equiv_\tau(x,x) =_{Bool} \text{True} & \\ & \forall x,y{:}\,\tau.\ \equiv_\tau(x,y) =_{Bool} \equiv_\tau(y,x) & \\ & \forall x,y,z{:}\,\tau.\ \equiv_\tau(x,y) =_{Bool} \text{True} \wedge & \} \\ & \qquad \equiv_\tau(y,z) =_{Bool} \text{True} \Rightarrow \equiv_\tau(x,z) =_{Bool} \text{True} & \end{array}$$

The set of axioms $Subst(\Sigma)$ that specify that (the relation with characteristic function) $\equiv$ is substitutive with respect to the operations in Σ is defined by

$$\begin{aligned} Subst(\Sigma) \overset{\text{def}}{=} \{&f,\tau s,\tau{:}\,(f{:}\,\tau s \to \tau) \in \mathbf{Op}(\Sigma){:}\,\forall xs,ys{:}\,\tau s.\\ &\equiv_{\tau s}(xs,ys) =_{Bool} \text{True} \Rightarrow \equiv_\tau(f(xs),f(ys)) =_{Bool} \text{True}\}\end{aligned}$$

The set of axioms $Equality(OUT)$ that specify that (the relation with characteristic function) $\equiv_\tau$ is the equality for each sort $\tau \in OUT$ is defined by

$$Equality(OUT) \overset{\text{def}}{=} \{\tau{:}\,\tau \in OUT{:}\,(\forall x,y{:}\,\tau.\ \equiv_\tau(x,y) =_{Bool} \text{True} \Leftrightarrow x =_\tau y)\}$$

Finally, the "OUT-congruence axioms" $Cong(\Sigma)_{OUT}$ which specify that (the T-indexed relation with characteristic function) $\equiv$ is a (Σ, OUT)-congruence is the set of Σ_{OUT}-axioms defined by

$$Cong(\Sigma)_{OUT} \overset{\text{def}}{=} (\bigcup \tau{:}\,\tau \in T{:}\,Equiv_\tau) \cup Subst(\Sigma) \cup Equality(OUT)$$

End Definition.

The remainder of the ultraloose transformation consists of a straightforward transformation of Σ-axioms (replacing $\forall$ by $\forall^{\Sigma}_{IN}$ and $=_\tau$ by $\equiv_\tau$).

Definition 4.2 (ultraloose axiom and specification transformation)

Let Σ be a signature and IN and OUT subsets of the sorts of Σ.

For any formula $\varphi \in \mathit{WFF}(\Sigma)$, the "$\mathit{IN}$-ultraloose transformation of φ" (written φ^{IN}) is the Σ_{OUT}-formula inductively defined by

$$
\begin{array}{lcl}
\mathit{true}^{\mathit{IN}} & \overset{\text{def}}{=} & \mathit{true} \\
(t1 =_\tau t2)^{\mathit{IN}} & \overset{\text{def}}{=} & \equiv_\tau(t1, t2) =_{\mathit{Bool}} \mathrm{True} \\
(\neg P)^{\mathit{IN}} & \overset{\text{def}}{=} & \neg(P^{\mathit{IN}}) \\
(P \wedge Q)^{\mathit{IN}} & \overset{\text{def}}{=} & P^{\mathit{IN}} \wedge Q^{\mathit{IN}} \\
(\forall x{:}\,\tau.\ P)^{\mathit{IN}} & \overset{\text{def}}{=} & \forall^{\Sigma}_{\mathit{IN}} x{:}\,\tau.\ P^{\mathit{IN}}
\end{array}
$$

For any set Ax of Σ-axioms, the "IN-ultraloose transformation of Ax" (written Ax^{IN}) is defined by

$$Ax^{\mathit{IN}} \overset{\text{def}}{=} \{ax{:}\ ax \in Ax{:}\ ax^{\mathit{IN}}\}$$

Finally, for any flat Σ-specification $\mathit{SP} \overset{\text{def}}{=} \langle \Sigma, Ax \rangle$, the "$(\mathit{IN}, \mathit{OUT})$-ultraloose transformation of SP" (written $\mathit{SP}^{\mathit{IN}}_{\mathit{OUT}}$) is the Σ-specification defined by

$$
\begin{array}{l}
\mathit{SP}^{\mathit{IN}}_{\mathit{OUT}} \overset{\text{def}}{=} \textbf{export } \Sigma \\
\qquad \textbf{from enrich } \mathrm{Bool} \\
\qquad\qquad \textbf{by sign } \Sigma_{\mathit{OUT}} \\
\qquad\qquad\quad \textbf{axioms } Ax^{\mathit{IN}} \\
\qquad\qquad\qquad\qquad \mathit{Cong}(\Sigma)_{\mathit{OUT}} \\
\qquad\qquad \textbf{end}
\end{array}
$$

End Definition.

For example, figure 4.3 shows the effect of the transformation on the specification in figure 4.2.[1] We note that this differs slightly from the specification in figure 4.1 which omits the congruence axioms for the sorts *Nat*, uses $=_{\mathit{Nat}}$ instead of $\equiv_{\mathit{Nat}}$ and names only the operations *empty*, *pop* and *push* in the quantification. It is straightforward to show that the two specifications are, in fact, equivalent.

In the interests of readability, we often abbreviate such specifications as follows:

[1] We omit the enrichment since *Stack* already contains the specification *Bool*.

$\mathit{Stack}' \overset{\text{def}}{=}$

export *StackSig* **from**

spec sign

$\quad \mathit{Nat}, \mathit{Bool}, \mathit{Stack}\colon$ **type**

$\quad \mathit{0}\colon \rightarrow \mathit{Nat}$

$\quad \mathit{succ}\colon \mathit{Nat} \rightarrow \mathit{Nat}$

$\quad \text{True}, \text{False}\colon \rightarrow \mathit{Bool}$

$\quad \mathit{empty}\colon \rightarrow \mathit{Stack}$

$\quad \mathit{push}\colon \mathit{Nat} \times \mathit{Stack} \rightarrow \mathit{Stack}$

$\quad \mathit{pop}\colon \mathit{Stack} \rightarrow \mathit{Stack}$

$\quad \mathit{top}\colon \mathit{Stack} \rightarrow \mathit{Nat}$

$\quad \mathit{isEmpty}\colon \mathit{Stack} \rightarrow \mathit{Bool}$

$\quad \equiv_{\mathit{Nat}}\colon \mathit{Nat} \times \mathit{Nat} \rightarrow \mathit{Bool}$

$\quad \equiv_{\mathit{Stack}}\colon \mathit{Stack} \times \mathit{Stack} \rightarrow \mathit{Bool}$

axioms

$\quad \forall^{\mathit{StackSig}}_{\{\mathit{Nat}\}} s\colon \mathit{Stack}, x\colon \mathit{Nat}.\ \equiv_{\mathit{Nat}}(\mathit{top}(\mathit{push}(x, s)), x) = \text{True}$

$\quad \forall^{\mathit{StackSig}}_{\{\mathit{Nat}\}} s\colon \mathit{Stack}, x\colon \mathit{Nat}.\ \equiv_{\mathit{Stack}}(\mathit{pop}(\mathit{push}(x, s)), s) = \text{True}$

$\quad \equiv_{\mathit{Bool}}(\mathit{isEmpty}(\mathit{empty}), \text{True}) = \text{True}$

$\quad \forall^{\mathit{StackSig}}_{\{\mathit{Nat}\}} s\colon \mathit{Stack}, x\colon \mathit{Nat}.\ \equiv(\mathit{isEmpty}(\mathit{push}(x, s)), \text{False}) = \text{True}$

$\quad \forall^{\mathit{StackSig}}_{\{\mathit{Nat}\}} m\colon \mathit{Nat}.\ \equiv_{\mathit{Nat}}(\mathit{succ}(m), \mathit{0}) \neq \text{True}$

$\quad \forall^{\mathit{StackSig}}_{\{\mathit{Nat}\}} m, n\colon \mathit{Nat}.$

$\qquad \equiv_{\mathit{Nat}}(\mathit{succ}(m), \mathit{succ}(n)) = \text{True} \Rightarrow \equiv_{\mathit{Nat}}(m, n) = \text{True}$

$\quad \forall^{\mathit{StackSig}}_{\{\mathit{Nat}\}} m\colon \mathit{Nat}.\ \exists^{\{\mathit{0}, \mathit{succ}\}}_{\emptyset} n\colon \mathit{Nat}.\ \equiv_{\mathit{Nat}}(m, n) = \text{True}$

$\quad \text{True} \neq \text{False}$

$\quad \forall b\colon \mathit{Bool}.\ b = \text{True} \vee b = \text{False}$

$\quad \forall n\colon \mathit{Nat}.\ \equiv_{\mathit{Nat}}(n, n) = \text{True}$

$\quad \forall n1, n2\colon \mathit{Nat}.\ \equiv_{\mathit{Nat}}(n1, n2) = \text{True} \Leftrightarrow \equiv_{\mathit{Nat}}(n2, n1) = \text{True}$

$\quad \forall n1, n2, n3\colon \mathit{Nat}.\ \equiv_{\mathit{Nat}}(n1, n2) = \text{True} \wedge$

$\qquad \equiv_{\mathit{Nat}}(n2, n3) = \text{True} \Rightarrow \equiv_{\mathit{Nat}}(n1, n3) = \text{True}$

$\quad \forall s\colon \mathit{Stack}.\ \equiv_{\mathit{Stack}}(s, s) = \text{True}$

$\quad \forall s1, s2\colon \mathit{Stack}.\ \equiv_{\mathit{Stack}}(s1, s2) = \text{True} \Leftrightarrow \equiv_{\mathit{Stack}}(s2, s1) = \text{True}$

$\quad \forall s1, s2, s3\colon \mathit{Stack}.\ \equiv_{\mathit{Stack}}(s1, s2) = \text{True} \wedge$

$\qquad \equiv_{\mathit{Stack}}(s2, s3) = \text{True} \Rightarrow \equiv_{\mathit{Stack}}(s1, s3) = \text{True}$

$\quad \equiv_{\mathit{Nat}}(\mathit{0}, \mathit{0}) = \text{True}$

$\quad \forall n1, n2\colon \mathit{Nat}.\ \equiv_{\mathit{Nat}}(n1, n2) = \text{True} \Rightarrow \equiv_{\mathit{Nat}}(\mathit{succ}(n1), \mathit{succ}(n2)) = \text{True}$

$\quad \equiv_{\mathit{Stack}}(\mathit{empty}, \mathit{empty}) = \text{True}$

$\quad \forall n1, n2\colon \mathit{Nat}, s1, s2\colon \mathit{Stack}.\ \equiv_{\mathit{Stack}}(s1, s2) = \text{True} \wedge$

$\qquad \equiv_{\mathit{Nat}}(n1, n2) = \text{True} \Rightarrow \equiv_{\mathit{Stack}}(\mathit{push}(n1, s1), \mathit{push}(n2, s2)) = \text{True}$

$\quad \forall s1, s2\colon \mathit{Stack}. \equiv_{\mathit{Stack}}(s1, s2) = \text{True} \Rightarrow \equiv_{\mathit{Stack}}(\mathit{pop}(s1), \mathit{pop}(s2)) = \text{True}$

$\quad \forall s1, s2\colon \mathit{Stack}.\ \equiv_{\mathit{Stack}}(s1, s2) = \text{True} \Rightarrow \equiv_{\mathit{Stack}}(\mathit{top}(s1), \mathit{top}(s2)) = \text{True}$

$\quad \forall s1, s2\colon \mathit{Stack}.\ \equiv_{\mathit{Stack}}(s1, s2) = \text{True} \Rightarrow$

$\qquad \equiv_{\mathit{Bool}}(\mathit{isEmpty}(s1), \mathit{isEmpty}(s2)) = \text{True}$

$\quad \forall n1, n2\colon \mathit{Nat}.\ \equiv_{\mathit{Nat}}(n1, n2) = \text{True} \Leftrightarrow x =_{\mathit{Nat}} y$

end

Figure 4.3: An Ultraloose Stack Specification

- We abbreviate $\equiv_\tau(t1, t2) = \text{True}$ to $t1 \equiv_\tau t2$ and $\neg(\equiv_\tau(t1, t2) = \text{True})$ to $t1 \not\equiv_\tau t2$.
- If $\tau \in OUT$ we omit $\equiv_\tau$ from the signature and write $t1 =_\tau t2$ instead of $t1 \equiv_\tau t2$ in the axioms.
- We abbreviate $\forall^{\Sigma}_{IN} x{:}\,\tau.\ P$ to $\forall^{r} x{:}\,\tau.\ P$ and add the line **IN** $=$ *IN* to the signature.

The effect of these abbreviations is shown in figure 4.4.

We find it convenient to keep the axioms $Cong(\Sigma)_{OUT}$ explicit since they are required when reasoning about specifications. It would probably be more convenient to abbreviate them if one was more interested in creating or understanding specifications than in formal proofs.

We noted above that the function $\equiv_\tau$ is not a relation but rather the characteristic function of a relation. The following two definitions make this more precise.

Definition 4.3 ($\equiv(\mathcal{A})$)

Let Σ be a signature, *IN* and *OUT* subsets of the sorts of Σ, $\mathcal{A}$ a Σ-algebra and $\equiv$ a (Σ, OUT)-congruence over $\mathcal{A}$.

The Σ_{OUT}-algebra $\equiv(\mathcal{A})$ is defined for each symbol $s \in \Sigma_{OUT}$ by

$$\equiv(\mathcal{A})_s \stackrel{\text{def}}{=} \begin{cases} \mathcal{A}_s, & \text{if } s \in \Sigma; \\ \{0, 1\}, & \text{if } s = Bool; \\ 1, & \text{if } s = \text{True}; \\ 0, & \text{if } s = \text{False}; \\ \equiv_\tau & \text{if } s = \equiv_\tau. \end{cases}$$

where, for each $\tau \in T$, the function $\equiv_\tau{:}\, \mathcal{A}_\tau \times \mathcal{A}_\tau \to \{0, 1\}$ is defined by $\equiv_\tau(a1, a2) \stackrel{\text{def}}{=} a1 \equiv_\tau a2$.

End Definition.

$\mathit{StackU} \overset{\text{def}}{=}$ **export** $\Sigma\mathit{Stack}$ **from** $\mathit{StackU'}$

$\mathit{StackU'} \overset{\text{def}}{=}$
enrich Nat + Bool
by sign Stack: **type**
$\mathit{empty}\colon \to \mathit{Stack}$
$\mathit{push}\colon \mathit{Nat} \times \mathit{Stack} \to \mathit{Stack}$
$\mathit{pop}\colon \mathit{Stack} \to \mathit{Stack}$
$\mathit{top}\colon \mathit{Stack} \to \mathit{Nat}$
$\mathit{isEmpty}\colon \mathit{Stack} \to \mathit{Bool}$
$\equiv\colon \mathit{Stack} \times \mathit{Stack} \to \mathit{Bool}$
$\mathrm{IN} = \mathit{Nat}$
axioms $\forall^r s\colon \mathit{Stack}, x\colon \mathit{Nat}.\ \mathit{top}(\mathit{push}(x,s)) = x$
$\forall^r s\colon \mathit{Stack}, x\colon \mathit{Nat}.\ \mathit{pop}(\mathit{push}(x,s)) \equiv s$
$\mathit{isEmpty}(\mathit{empty}) = \mathrm{True}$
$\forall^r s\colon \mathit{Stack}, x\colon \mathit{Nat}.\ \mathit{isEmpty}(\mathit{push}(x,s)) = \mathrm{False}$

$\forall s\colon \mathit{Stack}.\ s \equiv s$
$\forall s1, s2\colon \mathit{Stack}.\ s1 \equiv s2 \Leftrightarrow s2 \equiv s1$
$\forall s1, s2, s3\colon \mathit{Stack}.\ s1 \equiv s2 \wedge$
$s2 \equiv s3 \Rightarrow s1 \equiv s3$

$\mathit{empty} \equiv \mathit{empty}$
$\forall s1, s2\colon \mathit{Stack}, x\colon \mathit{Nat}.\ s1 \equiv s2 \Rightarrow \mathit{push}(x, s1) \equiv \mathit{push}(x, s2)$
$\forall s1, s2\colon \mathit{Stack}.\ s1 \equiv s2 \Rightarrow \mathit{pop}(s1) \equiv \mathit{pop}(s2)$
$\forall s1, s2\colon \mathit{Stack}.\ s1 \equiv s2 \Rightarrow \mathit{top}(s1) = \mathit{top}(s2)$
$\forall s1, s2\colon \mathit{Stack}.\ s1 \equiv s2 \Rightarrow \mathit{isEmpty}(s1) = \mathit{isEmpty}(s2)$
end

Figure 4.4: A Behaviourally Closed USL Stack

Definition 4.4 $(\equiv^{\mathcal{A}})$

Let Σ be a signature with sorts T, IN and OUT subsets of T and $\mathcal{A}$ a Σ_{OUT}-algebra.

The Σ-congruence $\equiv^{\mathcal{A}}$ is defined for each sort $\tau \in T$ and values $a1, a2 \in \mathcal{A}_\tau$ by

$$a1 \equiv_\tau a2 \overset{\text{def}}{=} \mathcal{A}_{\equiv^{\mathcal{A}}_\tau}(t1, t2) = \mathcal{A}_{\mathrm{True}}$$

End Definition.

4.2 Closure of SP^{IN}_{OUT} under $\underset{OUT}{\overset{IN}{\rightarrow}}$

The previous section defined a transformation from a "normal specification" SP to an "ultraloose specification" SP^{IN}_{OUT} and hinted that, because of the use of reachable quantification and congruences in SP^{IN}_{OUT}, there is a strong link with the relations $\underset{OUT}{\overset{IN}{\rightarrow}}$ and $\underset{OUT}{\overset{IN}{\longleftrightarrow}}$.

This section presents the first of three theorems which states this link more precisely. In particular, we show that the specification SP^{IN}_{OUT} is downward closed under $\underset{OUT}{\overset{IN}{\rightarrow}}$.

Theorem 4.5 (downward closure of SP^{IN}_{OUT})

Let Σ be a signature, *IN* and *OUT* subsets of the sorts of Σ, Ax a set of Σ-axioms and SP the flat specification $\langle \Sigma, Ax \rangle$.

SP^{IN}_{OUT} is downward closed under $\underset{OUT}{\overset{IN}{\rightarrow}}$.

That is, if $\mathcal{A} \underset{OUT}{\overset{IN}{\rightarrow}} \mathcal{B}$ and $\mathcal{B}\colon SP^{IN}_{OUT}$ then $\mathcal{A}\colon SP^{IN}_{OUT}$.

Proof

Let SP' be the specification

```
    enrich Bool
    by sign $\Sigma_{OUT}$
        axioms $Ax^{IN}$
                $Cong(\Sigma)_{OUT}$
    end
```

By the semantics of **export** Σ **from** _, for every model $\mathcal{B}$ of SP^{IN}_{OUT}, there is an extension $\mathcal{B}'\colon SP'$.

Let $\mathcal{B}'$ be a model of SP' and let $\mathcal{A}$ and $\mathcal{B}$ be Σ-algebras and h a Σ-homomorphism such that

$$\mathcal{B} = \mathcal{B}'|_{\Sigma} \text{ and } h\colon \mathcal{A} \underset{OUT}{\overset{IN}{\rightarrow}} \mathcal{B}$$

Let $\equiv$ be the Σ-congruence over $\mathcal{A}$ is defined for $a1, a2 \in \mathcal{A}_\tau$ by

$$a1 \equiv_\tau a2 \stackrel{\text{def}}{=} = h_\tau(a1) \equiv^{\mathcal{B}}_\tau h_\tau(a2)$$

Since $\equiv^{\mathcal{B}}$ is a congruence, it is obvious that $\equiv(\mathcal{A})$ satisfies the congruence axioms. It remains to show that $\equiv(\mathcal{A}) \models Ax^{IN}$.

We shall show that, for each Σ-formula φ and valuation $v \in Val(\mathcal{A}, \varphi^{IN})$,

$$\equiv(\mathcal{A}) \models_v \varphi^{IN} \Leftrightarrow \mathcal{B}' \models_{h \cdot v} \varphi^{IN}$$

Since $\equiv(\mathcal{A})$ and $\mathcal{B}'$ satisfy the same axioms (i.e. closed formulae) and $\mathcal{B}'$ satisfies all axioms in Ax^{IN}, it follows that $\equiv(\mathcal{A})$ satisfies all axioms in Ax^{IN} and so $\equiv(\mathcal{A})|_\Sigma\colon SP^{IN}_{OUT}$.

The proof is by induction over the structure of φ.

Base case: $(\varphi = true)$

For any valuation $v \in \textbf{Val}(\equiv(\mathcal{A}), \varphi)$.

$\equiv(\mathcal{A}) \models_v true^{IN}$
$=$ { definition of ax^{IN} }
$\equiv(\mathcal{A}) \models_v true$
$=$ { definition of $\models$ }
$true$
$=$ { definition of $\models$ }
$\mathcal{B}' \models_{h \cdot v} true$
$=$ { definition of ax^{IN} }
$\mathcal{B}' \models_{h \cdot v} true^{IN}$
□

Base case: $(\varphi = t1 =_\tau t2)$

For any valuation $v \in \textbf{Val}(\equiv(\mathcal{A}), \varphi)$.

$\equiv(\mathcal{A}) \models_v (t1 =_\tau t2)^{IN}$
$=$ { definition of ax^{IN} }
$\equiv(\mathcal{A}) \models_v \equiv_\tau(t1, t2) = \text{True}$
$=$ { definition of $\equiv(\mathcal{A})$ }
$\mathcal{B}'_{\equiv_\tau}(h(t1_{\equiv(\mathcal{A})}(v)), h(t2_{\equiv(\mathcal{A})}(v))) = \mathcal{B}'_{\text{True}}$
$=$ { lemma 2.12 }

$\mathcal{B}'_{\equiv_\tau}(t1_{\equiv(\mathcal{A})}(h(v)), t2_{\equiv(\mathcal{A})}(h(v))) = \mathcal{B}'_{\text{True}}$
$=$ { definition of $\models t1 = t2$ }
$\mathcal{B}' \models_{h\cdot v} \equiv_\tau(t1, t2) = \text{True}$
$=$ { definition of ax^{IN} }
$\mathcal{B}' \models_{h\cdot v} (t1 = t2)^{IN}$
□

Inductive step: $(\varphi = \neg P)$

Suppose that, for all $v \in \mathbf{Val}(\equiv(\mathcal{A}), P^{IN})$, $\equiv(\mathcal{A}) \models_v P^{IN} \Leftrightarrow \mathcal{B}' \models_{h\cdot v} P^{IN}$. Then,

$\equiv(\mathcal{A}) \models_v (\neg P)^{IN}$
$=$ { definition of ax^{IN} }
$\equiv(\mathcal{A}) \models_v \neg(P^{IN})$
$=$ { definition of $\models \neg P$ }
$\neg\equiv(\mathcal{A}) \models_v P^{IN}$
$=$ { inductive assumption }
$\neg\mathcal{B}' \models_{h\cdot v} P^{IN}$
$=$ { definition of $\models \neg P$ }
$\mathcal{B}' \models_{h\cdot v} \neg(P^{IN})$
$=$ { definition of ax^{IN} }
$\mathcal{B}' \models_{h\cdot v} (\neg P)^{IN}$
□

Inductive step: $(\varphi = P \wedge Q)$

Suppose that, for all $v \in \mathbf{Val}(\equiv(\mathcal{A}), (P \wedge Q)^{IN})$, $\equiv(\mathcal{A}) \models_v P^{IN} \Leftrightarrow \mathcal{B} \models_{h\cdot v} P^{IN}$ and $\equiv(\mathcal{A}) \models_v Q^{IN} \Leftrightarrow \mathcal{B} \models_{f\cdot v} Q^{IN}$. Then,

$\equiv(\mathcal{A}) \models_v (P \wedge Q)^{IN}$
$=$ { definition of ax^{IN} }
$\equiv(\mathcal{A}) \models_v P^{IN} \wedge Q^{IN}$
$=$ { definition of $\models P \wedge Q$ }
$\equiv(\mathcal{A}) \models_v P^{IN} \wedge \equiv(\mathcal{A}) \models_v Q^{IN}$
$=$ { inductive assumption }
$\mathcal{B} \models_{h\cdot v} P^{IN} \wedge \mathcal{B} \models_{h\cdot v} Q^{IN}$
$=$ { definition of $\models P \wedge Q$ }
$\mathcal{B} \models_{h\cdot v} P^{IN} \wedge Q^{IN}$
$=$ { definition of ax^{IN} }
$\mathcal{B} \models_{h\cdot v} (P \wedge Q)^{IN}$
□

Inductive step: $(\varphi = \forall x{:}\,\tau.\ P)$

Suppose that, for all $v \in \mathbf{Val}(\equiv(\mathcal{A}), P^{IN})$, $\equiv(\mathcal{A}) \models_v P^{IN} \Leftrightarrow \mathcal{B} \models_{h\cdot v} P^{IN}$. Then,

$\equiv(\mathcal{A}) \models_v (\forall x{:}\tau.\ P)^{IN}$
$= \quad \{$ definition of φ^{IN} $\}$
$\equiv(\mathcal{A}) \models_v \forall^{\Sigma}_{IN} x{:}\tau.\ P^{IN}$
$= \quad \{$ definition of $\models \forall^{\Sigma}_{IN}$ $\}$
$(\forall a : a \in \equiv(\mathcal{A})_\tau \wedge \mathcal{R}(\Sigma, IN, a) : \equiv(\mathcal{A}) \models_{v\oplus\{x:=a\}} P^{IN})$
$= \quad \{$ inductive assumption $\}$
$(\forall a : a \in \equiv(\mathcal{A})_\tau \wedge \mathcal{R}(\Sigma, IN, a) : \mathcal{B} \models_{h\cdot(v\oplus\{x:=a\})} P^{IN})$
$= \quad \{ \cdot$ distributes over $\oplus$ $\}$
$(\forall a : a \in \equiv(\mathcal{A})_\tau \wedge \mathcal{R}(\Sigma, IN, a) : \mathcal{B} \models_{(h\cdot v)\oplus\{x:=h(a)\}} P^{IN})$
$= \quad \{$ surjectivity of $h|_{IN}$ $\}$
$(\forall b : b \in \equiv(\mathcal{B})_\tau \wedge \mathcal{R}(\Sigma, IN, b) : \mathcal{B} \models_{(h\cdot v)\oplus\{x:=b\}} P^{IN})$
$= \quad \{$ definition of $\models \forall$ $\}$
$\mathcal{B}' \models_{h\cdot v} \forall^{\Sigma'}_{T'} x{:}\tau.\ P^{IN}$
$= \quad \{$ definition of φ^{IN} $\}$
$\mathcal{B} \models_{h\cdot v} (\forall x{:}\tau.\ P)^{IN}$
$\square$

Hence, by the principle of structural induction, we conclude that, for any formula $\varphi \in WFF(IN, OUT)$, and valuation $v \in \mathbf{Val}(\equiv(\mathcal{A}), \varphi^{IN})$

$$\equiv(\mathcal{A}) \models_v \varphi^{IN} \Leftrightarrow \mathcal{B}' \models_{h\cdot v} \varphi^{IN}$$

Since $\equiv(\mathcal{A})$ and $\mathcal{B}'$ satisfy the same axioms (i.e. closed formulae) and $\mathcal{B}'$ satisfies all axioms in Ax^{IN}, it follows that $\equiv(\mathcal{A})$ satisfies all axioms in Ax^{IN} and so $\equiv(\mathcal{A})|_\Sigma$: SP^{IN}_{OUT}. Since $\equiv(\mathcal{A})|_\Sigma = \mathcal{A}$, it follows that, if $\mathcal{A} \underset{OUT}{\overset{IN}{\rightarrow}} \mathcal{B}$ and $\mathcal{B}$: SP^{IN}_{OUT} then $\mathcal{A}$: SP^{IN}_{OUT}.

End Theorem.

Following the observation that the use of reachable quantification and congruences in ultraloose specifications suggests that there should be a link between the ultraloose style and the relation $\underset{OUT}{\overset{IN}{\rightarrow}}$, we have shown that $SP^{IN}OUT$ is downward closed under $\underset{OUT}{\overset{IN}{\rightarrow}}$.

It is tempting to conjecture that

$$\mathbf{Mod}(SP^{IN}_{OUT}) = \mathbf{Cl}_{\underset{OUT}{\overset{IN}{\rightarrow}}}(\mathbf{Mod}(SP))$$

To see that this does not hold, consider the following pathological example:

$SP \overset{\text{def}}{=}$ **spec** τ :**type**
axioms $\exists x{:}\,\tau.\ \mathit{true}$
end

which has as models any Σ-algebra $\mathcal{A}$ with non-empty carrier $\mathcal{A}_\tau$ (where $\Sigma \overset{\text{def}}{=} \mathbf{Sig}(SP)$). Applying the ultraloose transformation, we obtain the specification

$SP^{\emptyset}_{\emptyset} \overset{\text{def}}{=}$ **spec** τ :**type**
$\vdots$
axioms $\exists^{\Sigma}_{\emptyset} x{:}\,\tau.\ \mathit{true}$
$\vdots$

Since there are no terms in $W(\Sigma, \emptyset)$, we can easily show that for any $\Sigma^{\emptyset}_{\emptyset}$-algebra $\mathcal{A}'$, $\mathcal{A}' \not\models \exists^{\Sigma}_{\emptyset} x{:}\,\tau.\ \mathit{true}$:

$\mathcal{A}' \models \exists^{\Sigma}_{\emptyset} x{:}\,\tau.\ \mathit{true}$
$=$ { definition of $\exists$ }
$\mathcal{A}' \models \neg\forall^{\Sigma}_{\emptyset} x{:}\,\tau.\ \mathit{false}$
$=$ { definition of $\models$ }
$\neg(\forall a{:}\ a \in \mathcal{A}'_\tau \wedge \mathcal{R}(\Sigma, \emptyset, a){:}\mathit{false})$
$=$ { $\mathcal{A}'_\tau$ contains no $(\Sigma, \emptyset)$-reachable values }
$\neg\mathit{true}$
$=$
false
□

Hence, $SP^{\emptyset}_{\emptyset}$ is inconsistent and therefore cannot be the closure of the (consistent) specification SP.

This counterexample shows a significant difference between the ultraloose transformation and behavioural abstraction. Namely, that in certain circumstances the ultraloose transformation can reduce the number of models of a specification whereas behavioural abstraction can only increase the number of models. Sections 4.3 and 4.4 show that this can happen only if the specification uses inequations or existential quantification and identifies "safe" ways of using these constructs.

4.3 Closure of SP^{IN}_{OUT} under $\underset{OUT}{\overset{IN}{\longleftrightarrow}}$

The previous section showed that SP^{IN}_{OUT} is downward closed under $\underset{OUT}{\overset{IN}{\rightarrow}}$.

It is tempting to assume that SP^{IN}_{OUT} will also be closed under the (weaker) relation $\underset{OUT}{\overset{IN}{\longleftrightarrow}}$. However, this is not so: consider the following pathological example:

$$
\begin{array}{l}
SP \overset{\text{def}}{=} \\
\textbf{spec}\ \ \textbf{sign}\ \ \tau\ \textbf{:type} \\
\qquad\qquad\quad a, b :\to \tau \\
\qquad\ \ \textbf{axioms}\ \ a \neq b \\
\textbf{end}
\end{array}
$$

Under the ultraloose transformation, this is transformed into

$$
\begin{array}{lll}
SP^{\emptyset}_{\emptyset} \overset{\text{def}}{=} & & \\
\textbf{export}\ \{\tau, a, b\} & & \\
\textbf{from spec sign} & & Bool, \tau\ \textbf{:type} \\
 & & \text{True}, \text{False} :\to Bool \\
 & & a, b :\to \tau \\
 & & \equiv_\tau : \tau \times \tau \to Bool \\
 & \textbf{axioms} & a \not\equiv_\tau b \\
 & & \text{True} \neq \text{False} \\
 & & \forall b{:}\, Bool.\ b = \text{True} \vee b = \text{False} \\
 & & \forall x{:}\, \tau.\ x \equiv_\tau x \\
 & & \forall x, y{:}\, \tau.\ x \equiv_\tau y \Leftrightarrow y \equiv_\tau x \\
 & & \forall x, y, z{:}\, \tau.\ x \equiv_\tau y \wedge y \equiv_\tau z \Rightarrow x \equiv_\tau z \\
 & & a \equiv_\tau a \\
 & & b \equiv_\tau b \\
\quad \textbf{end} & &
\end{array}
$$

One model of $SP^{\emptyset}_{\emptyset}$ is the algebra

$$
\mathcal{A} \overset{\text{def}}{=} \langle\ \ \tau = \{0, 1\} \quad a = 0 \quad b = 1\ \ \rangle
$$

This is $(\emptyset,\emptyset)$-behaviourally equivalent to the algebra $\mathcal{B}$ defined by

$$
\begin{aligned}
\mathcal{B} \stackrel{\text{def}}{=} \langle\ & \tau = \{0\} \\
& a = 0 \\
& b = 0 \\
\rangle &
\end{aligned}
$$

(to see that $\mathcal{A} \underset{\emptyset}{\overset{\emptyset}{\longleftrightarrow}} \mathcal{B}$, consider the $(\emptyset,\emptyset)$-homomorphism $h_\tau(x) \stackrel{\text{def}}{=} 0$).

Since $\mathcal{B}_a = \mathcal{B}_b$, it is easy to show that there is no reflexive relation $\equiv: \mathcal{B}_\tau \leftrightarrow \mathcal{B}_\tau$ such that $\mathcal{B}_a \not\equiv \mathcal{B}_b$ and so $\mathcal{B}$ cannot be a model of $SP^{\emptyset}_{\emptyset}$. Since $\mathcal{A} \underset{\emptyset}{\overset{\emptyset}{\longleftrightarrow}} \mathcal{B}$, we conclude that $SP^{\emptyset}_{\emptyset}$ is not closed under $\underset{\emptyset}{\overset{\emptyset}{\longleftrightarrow}}$.

Thus the (IN, OUT)-ultraloose transformation of a specification SP which contains inequations is not (IN, OUT)-behaviourally closed.

One complication in this statement is that is not clear what we mean when we say that an axiom "contains an inequation". To see why this is vague, consider the following eight axioms:

$\neg\forall x{:}\,\tau.\ t1 = t2$	$\exists x{:}\,\tau.\ t1 \neq t2$
$\forall x{:}\,\tau.\ \neg t1 = t2$	$\forall x{:}\,\tau.\ t1 \neq t2$
$\neg\forall x{:}\,\tau.\ t1 \neq t2 \wedge t3 \neq t4$	$\exists x{:}\,\tau.\ t1 = t2 \vee t3 = t4$
$\neg\forall x{:}\,\tau.\ t1 \neq t2$	$\exists x{:}\,\tau.\ t1 = t2$

Considering the first column, the first two axioms don't *lexically* contain inequations but the last two axioms do. However, if we consider the equivalent axioms in the second column, the first two axioms do *lexically* contain inequations but the last two axioms don't.

We resolve this ambiguity by "pushing negations inwards": transforming axioms into "negation normal form" (see, for example, [22 section 4.4.2]). This makes use of $\vee$ and $\exists$ explicit allowing us to apply a simple lexical test. (The axioms in the second column are in negation normal form.)

Definition 4.6 (negation normal form)

Let Σ be a signature.

The "negation normal form" of any Σ-formula φ (written $NNF(\varphi)$) is inductively defined by

$$\begin{array}{lcl}
NNF(true) & \stackrel{\text{def}}{=} & true \\
NNF(\neg true) & \stackrel{\text{def}}{=} & false \\
NNF(t1 =_\tau t2) & \stackrel{\text{def}}{=} & t1 =_\tau t2 \\
NNF(\neg(t1 =_\tau t2)) & \stackrel{\text{def}}{=} & t1 \neq_\tau t2 \\
NNF(P \wedge Q) & \stackrel{\text{def}}{=} & NNF(P) \wedge NNF(Q) \\
NNF(\neg(P \wedge Q)) & \stackrel{\text{def}}{=} & NNF(\neg P) \vee NNF(\neg Q) \\
NNF(\forall x{:}\tau.\ P) & \stackrel{\text{def}}{=} & \forall x{:}\tau.\ NNF(P) \\
NNF(\neg(\forall x{:}\tau.\ P)) & \stackrel{\text{def}}{=} & \exists x{:}\tau.\ NNF(\neg P)
\end{array}$$

We say that a formula φ *is in negation normal form* if $\varphi = NNF(\varphi)$.

End Definition.

Theorem 4.7 (closure of SP^{IN}_{OUT})

Let Σ be a signature, IN and OUT subsets of the sorts of Σ, Ax a set of Σ-axioms and SP the flat specification $\langle \Sigma, Ax \rangle$.

If all inequations in $NNF(\!|Ax|\!)$ are over sorts in OUT, then SP^{IN}_{OUT} is (IN, OUT)-behaviourally closed.

Proof

Let SP' be the specification

> **enrich** Bool
> **by sign** Σ_{OUT}
>   **axioms** Ax^{IN}
> $Cong(\Sigma)_{OUT}$
> **end**

By the semantics of **export** Σ **from** _, for every model $\mathcal{A}$ of SP^{IN}_{OUT}, there is an extension $\mathcal{A}'\colon SP'$.

Let $\mathcal{A}'$ be a model of SP' and $\mathcal{A} \overset{\text{def}}{=} \mathcal{A}'|_{\Sigma}$.

Let $\mathcal{B}$ be a Σ-algebra such that $\mathcal{A} \xleftrightarrow[OUT]{IN} \mathcal{B}$. Since $\mathcal{A}'$ satisfies the congruence axioms, $\equiv^{\mathcal{A}'}$ is a congruence over $\mathcal{A}|_{\Sigma}$ and so, by lemma 4.9, there is a Σ-congruence $\equiv^{\mathcal{B}}$ over $\mathcal{B}$ and a homomorphism $h{:}\,\mathcal{R}(\Sigma, IN, \mathcal{A}) \to \mathcal{R}(\Sigma, IN, \mathcal{B})/_{\equiv^{\mathcal{B}}}$ such that, for each sort $\tau \in T$ and (Σ, IN)-reachable values $a1, a2 \in \mathcal{A}_{\tau}$,

$$a1 \equiv^{\mathcal{A}'}_{\tau} a2 \Rightarrow h_{\tau}(a1) = h(a2)$$

Since $\equiv^{\mathcal{A}'}$ is a congruence, $\equiv^{\mathcal{B}} (\mathcal{B})$ satisfies the congruence axioms.

We shall show that, for each Σ-formula φ and valuation $v \in Val(\mathcal{A}, \varphi^{IN})$,

$$\mathcal{A}' \models_v \varphi^{IN} \Rightarrow \equiv^{\mathcal{B}} (\mathcal{B}) \models_{h \cdot v} \varphi^{IN}$$

provided that all inequations in $NNF(\varphi)$ are over sorts in OUT. From which it follows that

$$\mathcal{A}' \models Ax^{IN} \Rightarrow \equiv^{\mathcal{B}} (\mathcal{B}) \models Ax^{IN}$$

provided that all inequations in $NNF(\varphi)$ are over sorts in OUT. Hence the result.

The proof is by induction over the structure of $NNF(\varphi)$.

Base cases: ($\varphi = true$ and $\varphi = false$)

For any valuation $v \in \mathbf{Val}(\mathcal{A}', \varphi)$.

$\mathcal{A}' \models_v true^{IN}$
$=$ { definition of ax^{IN} }
$\mathcal{A}' \models_v true$
$=$ { definition of $\models$ }
$true$
$=$ { definition of $\models$ }
$\equiv^{\mathcal{B}} (\mathcal{B}) \models_{h \cdot v} true$
$=$ { definition of ax^{IN} }
$\equiv^{\mathcal{B}} (\mathcal{B}) \models_{h \cdot v} true^{IN}$

$\square$

Hence,

$$\mathcal{A}' \models_v \textit{true}^{IN} \Rightarrow \equiv^{\mathcal{B}} (\mathcal{B}) \models_{h\cdot v} \textit{true}^{IN}$$

and

$$\mathcal{A}' \models_v \textit{false}^{IN} \Rightarrow \equiv^{\mathcal{B}} (\mathcal{B}) \models_{h\cdot v} \textit{false}^{IN}$$

Base case: $(\varphi = t1 =_\tau t2)$

For any valuation $v \in \mathbf{Val}(\mathcal{A}', \varphi)$.

$\mathcal{A}' \models_v (t1 =_\tau t2)^{IN}$
$=$ { definition of ax^{IN} }
$\mathcal{A}' \models_v \equiv_\tau(t1, t2) = \text{True}$
$=$ { definition of $\models$ and $\equiv^{\mathcal{A}'}$ }
$t1_{\mathcal{A}}(v) \equiv_\tau^{\mathcal{A}'} t2_{\mathcal{A}}(v)$
$\Rightarrow$ { Liebniz }
$h(t1_{\mathcal{A}}(v)) \equiv_\tau^{\mathcal{B}} h(t2_{\mathcal{A}}(v))$
$=$ { lemma 2.12 }
$t1_{\mathcal{B}}(h \cdot v) \equiv_\tau^{\mathcal{B}} t2_{\mathcal{B}}(h \cdot v)$
$=$ { definition of $\equiv^{\mathcal{A}'}$ and $\models$ }
$\equiv^{\mathcal{B}} (\mathcal{B}) \models_{h\cdot v} \equiv_\tau(t1, t2) = \text{True}$
$=$ { definition of ax^{IN} }
$\equiv^{\mathcal{B}} (\mathcal{B}) \models_{h\cdot v} (t1 =_\tau t2)^{IN}$
□

Base case: $(\varphi = t1 \neq_\tau t2$ and $\tau \in OUT)$

For any valuation $v \in \mathbf{Val}(\mathcal{A}', \varphi)$.

$\mathcal{A}' \models_v (t1 \neq_\tau t2)^{IN}$
$=$ { definition of ax^{IN} }
$\mathcal{A}' \models_v \equiv_\tau(t1, t2) = \text{False}$
$=$ { definition of $\models$ and $\equiv^{\mathcal{A}'}$ }
$t1_{\mathcal{A}}(v) \not\equiv_\tau^{\mathcal{A}'} t2_{\mathcal{A}}(v)$
$=$ { lemma 4.9 and $\tau \in OUT$ }
$h(t1_{\mathcal{A}}(v)) \not\equiv_\tau^{\mathcal{B}} h(t2_{\mathcal{A}}(v))$
$=$ { lemma 2.12 }
$t1_{\mathcal{B}}(h \cdot v) \not\equiv_\tau^{\mathcal{B}} t2_{\mathcal{B}}(h \cdot v)$
$=$ { definition of $\equiv^{\mathcal{A}'}$ and $\models$ }
$\equiv^{\mathcal{B}} (\mathcal{B}) \models_{h\cdot v} \equiv_\tau(t1, t2) = \text{False}$
$=$ { definition of ax^{IN} }
$\equiv^{\mathcal{B}} (\mathcal{B}) \models_{h\cdot v} (t1 \neq_\tau t2)^{IN}$
□

Inductive steps: ($\varphi = P \wedge Q$ and $\varphi = P \vee Q$)

Suppose that, for all $v \in \mathbf{Val}(\mathcal{A}', (P \wedge Q)^{IN})$, $\mathcal{A}' \models_v P^{IN} \Rightarrow \mathcal{B} \models_{h \cdot v} P^{IN}$ and $\mathcal{A}' \models_v Q^{IN} \Rightarrow \mathcal{B} \models_{h \cdot v} Q^{IN}$. Then,

$\mathcal{A}' \models_v (P \wedge Q)^{IN}$
$=$ { definition of ax^{IN} }
$\mathcal{A}' \models_v P^{IN} \wedge Q^{IN}$
$=$ { definition of $\models P \wedge Q$ }
$\mathcal{A}' \models_v P^{IN} \wedge \mathcal{A}' \models_v Q^{IN}$
$\Rightarrow$ { inductive assumption }
$\mathcal{B} \models_{h \cdot v} P^{IN} \wedge \mathcal{B} \models_{h \cdot v} Q^{IN}$
$=$ { definition of $\models P \wedge Q$ }
$\mathcal{B} \models_{h \cdot v} P^{IN} \wedge Q^{IN}$
$=$ { definition of ax^{IN} }
$\mathcal{B} \models_{h \cdot v} (P \wedge Q)^{IN}$
□

By a similar proof, we may show that, if $\mathcal{A}' \models_v P^{IN} \Rightarrow \mathcal{B} \models_{h \cdot v} P^{IN}$ and $\mathcal{A}' \models_v Q^{IN} \Rightarrow \mathcal{B} \models_{h \cdot v} Q^{IN}$, then

$$\mathcal{A}' \models_v (P \vee Q)^{IN} \quad \Rightarrow \quad \mathcal{B} \models_{h \cdot v} (P \vee Q)^{IN}$$

Inductive step: ($\varphi = \forall x\colon \tau.\ P$ and $\varphi = \exists x\colon \tau.\ P$)

Suppose that, for all $v \in \mathbf{Val}(\mathcal{A}', P^{IN})$, $\mathcal{A}' \models_v P^{IN} \Rightarrow \mathcal{B} \models_{h \cdot v} P^{IN}$. Then,

$\mathcal{A}' \models_v (\forall x\colon \tau.\ P)^{IN}$
$=$ { definition of φ^{IN} }
$\mathcal{A}' \models_v \forall^{\Sigma}_{IN} x\colon \tau.\ P^{IN}$
$=$ { definition of $\models \forall^{\Sigma}_{IN}$ }
$(\forall a : a \in \mathcal{A}_\tau \wedge \mathcal{R}(\Sigma, IN, a) : \mathcal{A} \models_{v \oplus \{x := a\}} P^{IN})$
$\Rightarrow$ { inductive assumption }
$(\forall a : a \in \mathcal{A}_\tau \wedge \mathcal{R}(\Sigma, IN, a) : \mathcal{B} \models_{h \cdot (v \oplus \{x := a\})} P^{IN})$
$=$ { $\cdot$ distributes over $\oplus$ }
$(\forall a : a \in \mathcal{A}_\tau \wedge \mathcal{R}(\Sigma, IN, a) : \mathcal{B} \models_{(h \cdot v) \oplus \{x := h(a)\}} P^{IN})$
$=$ { surjectivity of $h|_{IN}$ }
$(\forall b : b \in \mathcal{B}_\tau \wedge \mathcal{R}(\Sigma, IN, b) : \mathcal{B} \models_{(h \cdot v) \oplus \{x := b\}} P^{IN})$
$=$ { definition of $\models \forall$ }
$\equiv^{\mathcal{B}} (\mathcal{B}) \models_{h \cdot v} \forall^{\Sigma}_{IN} x\colon \tau.\ P^{IN}$
$=$ { definition of φ^{IN} }
$\mathcal{B} \models_{h \cdot v} (\forall x\colon \tau.\ P)^{IN}$
□

By a similar argument, we may show that

$$\mathcal{A}' \models_v (\exists x\colon \tau.\ P)^{IN} \quad \Rightarrow \quad \mathcal{B} \models_{h \cdot v} (\exists x\colon \tau.\ P)^{IN}$$

Hence, by the principle of structural induction, we conclude that, for any formula $\varphi \in \mathit{WFF}(\mathit{IN}, \mathit{OUT})$, and valuation $v \in \mathbf{Val}(\mathcal{A}', \varphi)$ such that all inequations in $\mathit{NNF}(\varphi)$ are over sorts in OUT, that

$$\mathcal{A}' \models_v \varphi^{\mathit{IN}} \Leftrightarrow \equiv^{\mathcal{B}} (\mathcal{B}) \models_{h \cdot v} \varphi^{\mathit{IN}}$$

Hence, whenever $\mathcal{A}$ is a model of $SP^{\mathit{IN}}_{\mathit{OUT}}$ and $\mathcal{A} \xleftrightarrow[\mathit{OUT}]{\mathit{IN}} \mathcal{B}$, then $\equiv^{\mathcal{B}}(\mathcal{B})$ is a model of $SP^{\mathit{IN}}_{\mathit{OUT}}$ and we conclude that $SP^{\mathit{IN}}_{\mathit{OUT}}$ is behaviourally closed.

End Theorem.

This result is important because it provides specifiers with a precise methodology for writing behaviourally closed specifications: first write the specification in the normal way (but avoiding inequations!); then apply the ultraloose transformation to obtain a behaviourally closed specification.

We have given counterexamples to show that it is not true that $SP^{\mathit{IN}}_{\mathit{OUT}}$ is $(\mathit{IN}, \mathit{OUT})$-behaviourally closed for any specification SP. Observing that the problem lay in inequations, we defined negation normal form to let us identify problematic axioms (those whose negation normal form contains inequations $t1 \neq_\tau t2$ where $\tau \notin \mathit{OUT}$) and showed that, if SP does not contain such axioms then $SP^{\mathit{IN}}_{\mathit{OUT}}$ is $(\mathit{IN}, \mathit{OUT})$-behaviourally closed.

4.4 Equivalence of ASL and USL

The previous section showed that $SP^{\mathit{IN}}_{\mathit{OUT}}$ is closed under $\xleftrightarrow[\mathit{OUT}]{\mathit{IN}}$ (provided SP contains no inequations). It is natural to wonder whether, for such specifications,

$$\mathbf{Mod}(SP^{\mathit{IN}}_{\mathit{OUT}}) = \mathbf{Cl}_{\underset{\mathit{OUT}}{\overset{\mathit{IN}}{\longleftrightarrow}}}(SP)$$

But, at the end of section 4.2 we showed that this does not hold for $\xrightarrow[\mathit{OUT}]{\mathit{IN}}$. Since $\xleftrightarrow[\mathit{OUT}]{\mathit{IN}}$ is weaker than $\xrightarrow[\mathit{OUT}]{\mathit{IN}}$ we may conclude that it does not hold for $\xleftrightarrow[\mathit{OUT}]{\mathit{IN}}$ either.

In this case, the problem lies in the use of existential quantification. We shall show that the semantics of the transformed specification SP^{IN}_{OUT} is exactly the behavioural closure of the models of SP provided SP contains neither existential quantification nor inequations.

The proof requires two supporting lemmas. The first states that all models of SP^{IN}_{OUT} are behaviourally equivalent to a model of SP.

Lemma 4.8 ($\mathbf{Mod}(SP^{IN}_{OUT}) \subseteq \mathbf{Mod}^{IN}_{OUT}(SP)$)

Let Σ be a signature, IN and OUT subsets of the sorts of Σ, Ax a set of Σ-axioms and SP the flat specification $\langle \Sigma, Ax \rangle$.

Then, for any Σ-algebra $\mathcal{A}$,

$$\mathcal{A} \in \mathbf{Mod}(SP^{IN}_{OUT}) \Rightarrow \mathcal{A} \in \mathbf{Mod}^{IN}_{OUT}(SP)$$

($\mathbf{Mod}^{IN}_{OUT}(SP)$ is defined in definition 3.13.)

Proof

We shall show that for any model $\mathcal{A}$ of SP^{IN}_{OUT}, there is an (IN, OUT)-behaviourally equivalent Σ-algebra $\mathcal{B}$ such that $\mathcal{B}$ is a model of SP.

Let SP' be the specification

```
enrich Bool
by sign $\Sigma_{OUT}$
    axioms $Ax^{IN}$
            $Cong(\Sigma)_{OUT}$
 end
```

Let $\mathcal{A}'$ be any model of SP' and let $\mathcal{A} \overset{\text{def}}{=} \mathcal{A}'|_{\Sigma}$.

Since $\mathcal{A}'$ satisfies the congruence axioms, $\equiv^{\mathcal{A}'}$ is a Σ-congruence. Let $\mathcal{B}$ be defined by $\mathcal{B} \overset{\text{def}}{=} \mathcal{R}(\Sigma, IN, \mathcal{A})/_{\equiv^{\mathcal{A}'}}$ and let $h \overset{\text{def}}{=} [\![\_]\!]_{\equiv^{\mathcal{A}'}}$.

We shall show that, for each Σ-formula φ and valuation $v \in \mathbf{Val}(\mathcal{A}, \varphi^{IN})$

$$\mathcal{A}' \models_v \varphi^{IN} \Leftrightarrow \mathcal{B} \models_{h \cdot v} \varphi$$

The proof is by induction over the structure of φ.

Base case: ($\varphi = \mathit{true}$)

For any valuation $v \in \mathbf{Val}(\mathcal{A}', \varphi)$.

$\mathcal{A}' \models_v \mathit{true}^{IN}$
$=$ { definition of ax^{IN} }
$\mathcal{A}' \models_v \mathit{true}$
$=$ { definition of $\models$ }
true
$=$ { definition of $\models$ }
$\mathcal{B} \models_{h \cdot v} \mathit{true}$
□

Base case: ($\varphi = t1 =_\tau t2$)

For any valuation $v \in \mathbf{Val}(\mathcal{A}', \varphi)$.

$\mathcal{A}' \models_v (t1 =_\tau t2)^{IN}$
$=$ { definition of ax^{IN} }
$\mathcal{A}' \models_v \equiv_\tau(t1, t2) = \mathrm{True}$
$=$ { definition of $\models$ }
$\equiv^{\mathcal{A}'}{}_\tau(t1_{\mathcal{A}'}(v), t2_{\mathcal{A}'}(v)) = \mathcal{A}'_{\mathrm{True}}$
$=$ { definition of $\equiv^{\mathcal{A}'}$ }
$t1'_{\mathcal{A}}(v) \equiv^{\mathcal{A}'}{}_\tau t2'_{\mathcal{A}}(v)$
$=$ { definition of h }
$h(t1'_{\mathcal{A}}(v)) = h(t2'_{\mathcal{A}}(v))$
$=$ { lemma 2.12 }
$t1_{\mathcal{B}}(h \cdot v) = t2_{\mathcal{B}}(h \cdot v)$
$=$ { definition of $\models t1 = t2$ }
$\mathcal{B} \models_{h \cdot v} t1 = t2$
□

Inductive step: ($\varphi = \neg P$)

Suppose that, for all $v \in \mathbf{Val}(\mathcal{A}', P^{IN})$, $\mathcal{A}' \models_v P^{IN} \Leftrightarrow \mathcal{B} \models_{h \cdot v} P^{IN}$. Then,

$\mathcal{A}' \models_v (\neg P)^{IN}$
$=$ { definition of ax^{IN} }
$\mathcal{A}' \models_v \neg(P^{IN})$

$=$ $\{$ definition of $\models \neg P$ $\}$
$\neg\mathcal{A}' \models_v P^{IN}$
$\Leftrightarrow$ $\{$ inductive assumption $\}$
$\neg\mathcal{B} \models_{h \cdot v} P$
$=$ $\{$ definition of $\models \neg P$ $\}$
$\mathcal{B} \models_{h \cdot v} \neg(P)$
□

Inductive step: $(\varphi = P \wedge Q)$

Suppose that, for all $v \in \mathbf{Val}(\mathcal{A}', (P \wedge Q)^{IN})$, $\mathcal{A}' \models_v P^{IN} \Leftrightarrow \mathcal{B} \models_{h \cdot v} P^{IN}$ and $\mathcal{A}' \models_v Q^{IN} \Leftrightarrow \mathcal{B} \models_{f \cdot v} Q^{IN}$. Then,

$\mathcal{A}' \models_v (P \wedge Q)^{IN}$
$=$ $\{$ definition of ax^{IN} $\}$
$\mathcal{A}' \models_v P^{IN} \wedge Q^{IN}$
$=$ $\{$ definition of $\models P \wedge Q$ $\}$
$\mathcal{A}' \models_v P^{IN} \wedge \mathcal{A}' \models_v Q^{IN}$
$\Leftrightarrow$ $\{$ inductive assumption $\}$
$\mathcal{B} \models_{h \cdot v} P \wedge \mathcal{B} \models_{h \cdot v} Q$
$=$ $\{$ definition of $\models P \wedge Q$ $\}$
$\mathcal{B} \models_{h \cdot v} P \wedge Q$
□

Inductive step: $(\varphi = \forall x{:}\,\tau.\ P)$

Suppose that, for all $v \in \mathbf{Val}(\mathcal{A}', P^{IN})$, $\mathcal{A}' \models_v P^{IN} \Leftrightarrow \mathcal{B} \models_{h \cdot v} P^{IN}$. Then,

$\mathcal{A}' \models_v (\forall x{:}\,\tau.\ P)^{IN}$
$=$ $\{$ definition of φ^{IN} $\}$
$\mathcal{A}' \models_v \forall^{\Sigma}_{IN} x{:}\,\tau.\ P^{IN}$
$=$ $\{$ definition of $\models \forall^{\Sigma}_{IN}$ $\}$
$(\forall a : a \in \mathcal{A}_\tau \wedge \mathcal{R}(\Sigma, IN, a) : \mathcal{A} \models_{v \oplus \{x:=a\}} P^{IN})$
$=$ $\{$ inductive assumption $\}$
$(\forall a : a \in \mathcal{A}_\tau \wedge \mathcal{R}(\Sigma, IN, a) : \mathcal{B} \models_{h \cdot (v \oplus \{x:=a\})} P)$
$=$ $\{$ $\cdot$ distributes over $\oplus$ $\}$
$(\forall a : a \in \mathcal{A}_\tau \wedge \mathcal{R}(\Sigma, IN, a) : \mathcal{B} \models_{(h \cdot v) \oplus \{x:=h(a)\}} P)$
$=$ $\{$ surjectivity of $h|_{IN}$ $\}$
$(\forall b : b \in \mathcal{B}_\tau \wedge \mathcal{R}(\Sigma, IN, b) : \mathcal{B} \models_{(h \cdot v) \oplus \{x:=b\}} P)$
$=$ $\{$ definition of $\models \forall$ $\}$
$\mathcal{B} \models_{h \cdot v} \forall^{\Sigma}_{IN} x{:}\,\tau.\ P$
□

Hence, by the principle of structural induction, we conclude that, for any formula $\varphi \in \mathit{WFF}(\mathit{IN}, \mathit{OUT})$, and valuation $v \in \mathbf{Val}(\mathcal{A}', \varphi)$

$$\mathcal{A}' \models_v \varphi^{IN} \Leftrightarrow \mathcal{B} \models_{h \cdot v} \varphi$$

Therefore we conclude that, for any Σ-algebra $\mathcal{A}$,

$$\mathcal{A}\colon SP^{IN} \;\Rightarrow\; \mathcal{B}\colon SP$$

Since $\mathcal{A} \xleftrightarrow[OUT]{IN} \mathcal{B}$, the result immediately follows.

End Lemma.

The second lemma states that all models of SP are models of SP^{IN}_{OUT}.

Lemma 4.9 ($SP^{IN}_{OUT} \rightsquigarrow SP$)

Let Σ be a signature, IN and OUT subsets of the sorts of Σ, Ax a set of Σ-axioms and SP the flat specification $\langle \Sigma, Ax \rangle$.

If all existential quantification in $NNF(\llparenthesis Ax \rrparenthesis)$ is over sorts in IN then, for any Σ-algebra $\mathcal{A}$,

$$\mathcal{A}\colon SP \;\Rightarrow\; \mathcal{A}\colon SP^{IN}_{OUT}$$

Proof

Let SP' be the specification

```
enrich Bool
by sign $\Sigma_{OUT}$
    axioms $Ax^{IN}$
               $Cong(\Sigma)_{OUT}$
 end
```

We shall show that for every model $\mathcal{A}\colon SP$, there is an extension $\mathcal{A}'$ such that $\mathcal{A}'\colon SP'$ and, hence, $\mathcal{A}\colon SP^{IN}_{OUT}$.

Let $\mathcal{A}$ be a model of SP.

Since equality is a congruence relation, $=(\mathcal{A})$ satisfies the congruence axioms.

We shall show that, for each Σ-formula φ and valuation $v \in \mathbf{Val}(\mathcal{A}, \varphi)$ such that all existential quantification in $NNF(\varphi)$ is over sorts in IN, that

$$\mathcal{A} \models_v \varphi \Rightarrow =(\mathcal{A}) \models_v \varphi^{IN}$$

The proof is by induction over the structure of $NNF(\varphi)$.

Base cases: ($\varphi = true$ and $\varphi = false$)

For any valuation $v \in \mathbf{Val}(\mathcal{A}, \varphi)$.

$\mathcal{A} \models_v true$
$=$ { definition of $\models$ }
$true$
$=$ { definition of $\models$ }
$=(\mathcal{A}) \models_{h \cdot v} true$
$=$ { definition of ax^{IN} }
$=(\mathcal{A}) \models_{h \cdot v} true^{IN}$
□

Base cases: ($\varphi = t1 =_\tau t2$ and $\varphi = t1 \neq_\tau t2$

For any valuation $v \in \mathbf{Val}(\mathcal{A}, \varphi)$.

$\mathcal{A} \models_v t1 =_\tau t2$
$=$ { definition of $\models$ }
$t1_{\mathcal{A}}(v) = t2_{\mathcal{A}}(v)$
$=$ { definition of $\equiv$ and $=(\mathcal{A})$ }
$\equiv_\tau (t1_{=(\mathcal{A})}(v), t2_{=(\mathcal{A})}(v)) = =(\mathcal{A})_{\mathrm{True}}$
$=$ { definition of $\models$ }
$=(\mathcal{A}) \models_v \equiv_\tau(t1, t2) = \mathrm{True}$
$=$ { definition of ax^{IN} }
$=(\mathcal{A}) \models_v (t1 =_\tau t2)^{IN}$
□

Hence,

$$\mathcal{A} \models_v t1 =_\tau t2 \Rightarrow =(\mathcal{A}) \models_v t1 =_\tau t2$$

and

$$\mathcal{A} \models_v t1 \neq_\tau t2 \Rightarrow =(\mathcal{A}) \models_v t1 \neq_\tau t2$$

Inductive step: ($\varphi = P \wedge Q$ and $\varphi = P \vee Q$)

Suppose that, for all $v \in \mathbf{Val}(\mathcal{A}, P \wedge Q)$, $\mathcal{A} \models_v P \Rightarrow {=}(\mathcal{A}) \models_v P^{IN}$ and $\mathcal{A} \models_v Q \Rightarrow {=}(\mathcal{A}) \models_v Q^{IN}$. Then,

$\mathcal{A} \models_v P \wedge Q$
$=$ $\{$ definition of $\models P \wedge Q$ $\}$
$\mathcal{A} \models_v P \wedge \mathcal{A} \models_v Q$
$\Rightarrow$ $\{$ inductive assumption $\}$
${=}(\mathcal{A}) \models_v P^{IN} \wedge {=}(\mathcal{A}) \models_v Q^{IN}$
$=$ $\{$ definition of $\models P \wedge Q$ $\}$
${=}(\mathcal{A}) \models_v P^{IN} \wedge Q^{IN}$
$=$ $\{$ definition of ax^{IN} $\}$
${=}(\mathcal{A}) \models_v (P \wedge Q)^{IN}$
□

By a similar proof, we may show that, if $\mathcal{A} \models_v P \Rightarrow {=}(\mathcal{A}) \models_v P^{IN}$ and $\mathcal{A} \models_v Q \Rightarrow {=}(\mathcal{A}) \models_v Q^{IN}$, then

$$\mathcal{A} \models_v P \vee Q \quad \Rightarrow \quad {=}(\mathcal{A}) \models_v (P \vee Q)^{IN}$$

Inductive step: ($\varphi = \forall x{:}\,\tau.\ P$)

Suppose that, for all $v \in \mathbf{Val}(\mathcal{A}, P)$, $\mathcal{A} \models_v P \Rightarrow {=}(\mathcal{A}) \models_v P^{IN}$. Then,

$\mathcal{A} \models_v \forall x{:}\,\tau.\ P$
$=$ $\{$ definition of $\models \forall$ $\}$
$(\forall a{:}\ a \in \mathcal{A}_\tau{:}\ \mathcal{A} \models_{v\oplus\{x:=a\}} P)$
$\Rightarrow$ $\{$ predicate calculus $\}$
$(\forall a{:}\ a \in \mathcal{A}_\tau \wedge \mathcal{R}(\Sigma, IN, a){:}\ \mathcal{A} \models_{v\oplus\{x:=a\}} P)$
$\Rightarrow$ $\{$ inductive assumption $\}$
$(\forall a{:}\ a \in \mathcal{A}_\tau \wedge \mathcal{R}(\Sigma, IN, a){:}\ {=}(\mathcal{A}) \models_{v\oplus\{x:=a\}} P^{IN})$
$=$ $\{$ definition of $\models \forall^{\Sigma}_{IN}$ $\}$
${=}(\mathcal{A}) \models_v \forall^{\Sigma}_{IN} x{:}\,\tau.\ P^{IN}$
$=$ $\{$ definition of φ^{IN} $\}$
${=}(\mathcal{A}) \models_v (\forall x{:}\,\tau.\ P)^{IN}$
□

Inductive step: ($\varphi = \exists x{:}\,\tau.\ P$ and $\tau \in IN$)

Suppose that, for all $v \in \mathbf{Val}(\mathcal{A}, P^{IN})$, $\mathcal{A} \models_v P^{IN} \Rightarrow \mathcal{B} \models_{h \cdot v} P^{IN}$. Then,

$\mathcal{A} \models_v \exists x{:}\,\tau.\ P$
$=$ $\{$ definition of $\models \exists^{\Sigma}_{IN}$ $\}$
$(\exists a{:}\ a \in \mathcal{A}_\tau{:}\ \mathcal{A} \models_{v\oplus\{x:=a\}} P)$
$=$ $\{$ $\tau \in IN$ $\}$

$(\exists a \colon a \in \mathcal{A}_\tau \wedge \mathcal{R}(\Sigma, IN, a) \colon \mathcal{A} \models_{v \oplus \{x := a\}} P)$
$\Rightarrow \quad \{ \text{ inductive assumption } \}$
$(\exists a \colon a \in \mathcal{A}_\tau \wedge \mathcal{R}(\Sigma, IN, a) \colon =\!(\mathcal{A}) \models_{v \oplus \{x := a\}} P^{IN})$
$= \quad \{ \text{ definition of } \models \exists^{\Sigma}_{IN} \}$
$=\!(\mathcal{A}) \models_v \exists^{\Sigma}_{IN} x \colon \tau .\ P^{IN}$
$= \quad \{ \text{ definition of } \varphi^{IN} \}$
$=\!(\mathcal{A}) \models_v (\exists x \colon \tau .\ P)^{IN}$
$\square$

Hence, by the principle of structural induction, we conclude that, for any formula $\varphi \in WFF(IN, OUT)$, and valuation $v \in \mathbf{Val}(\mathcal{A}, \varphi)$ such that all existential quantification in $NNF(\varphi)$ is over sorts in IN,

$$\mathcal{A} \models_v \varphi \ \Rightarrow \ =\!(\mathcal{A}) \models_v \varphi^{IN}$$

It follows that, for any observational axiom ax such that all existential quantification in $NNF(ax)$ is over sorts in IN,

$$\mathcal{A} \models ax \ \Rightarrow \ =\!(\mathcal{A}) \models ax^{IN}$$

and so, for every model $\mathcal{A}$ of SP, $\equiv\!(\mathcal{A})$ is a model of SP'. Hence result.

End Lemma.

We can now state and prove the major result of this thesis.

Theorem 4.10 (semantic effect of ultraloose transformation)

Let Σ be a signature, IN and OUT subsets of the sorts of Σ, Ax a set of Σ-axioms and SP the flat Σ-specification $\langle \Sigma, Ax \rangle$.

If all existential quantification in $NNF(\llparenthesis Ax \rrparenthesis)$ is over sorts in IN and all inequations in $NNF(\llparenthesis Ax \rrparenthesis)$ is over sorts in OUT, then

$$\mathbf{Mod}(SP^{IN}_{OUT}) = \mathbf{Mod}^{IN}_{OUT}(SP)$$

Proof

$\mathbf{Mod}(SP^{IN}_{OUT}) = \mathbf{Mod}^{IN}_{OUT}(SP)$
$=$ { theorem 4.7 }
$\mathbf{Mod}^{IN}_{OUT}(SP^{IN}_{OUT}) = \mathbf{Mod}^{IN}_{OUT}(SP)$
$=$ { set theory }
$\mathbf{Mod}^{IN}_{OUT}(SP) \subseteq \mathbf{Mod}^{IN}_{OUT}(SP^{IN}_{OUT}) \wedge \mathbf{Mod}^{IN}_{OUT}(SP^{IN}_{OUT}) \subseteq \mathbf{Mod}^{IN}_{OUT}(SP)$
$\Leftarrow$ { closure properties }
$\mathbf{Mod}(SP) \subseteq \mathbf{Mod}(SP^{IN}_{OUT}) \wedge \mathbf{Mod}^{IN}_{OUT}(SP^{IN}_{OUT}) \subseteq \mathbf{Mod}^{IN}_{OUT}(SP)$
$=$ { lemma 4.9 }
$\mathbf{Mod}^{IN}_{OUT}(SP^{IN}_{OUT}) \subseteq \mathbf{Mod}^{IN}_{OUT}(SP)$
$=$ { theorem 4.7 }
$\mathbf{Mod}(SP^{IN}_{OUT}) \subseteq \mathbf{Mod}^{IN}_{OUT}(SP)$
$=$ { lemma 4.8 }
true
□

End Theorem.

Having seen counterexamples to show that

$$\mathbf{Mod}(SP^{IN}_{OUT}) = \mathbf{Cl}_{\underset{OUT}{\overset{IN}{\longleftrightarrow}}}(\mathbf{Mod}(SP))$$

does not hold if *SP* contains problematic axioms (axioms containing inequations or existential quantification) we showed that it does hold in the absence of such problematic axioms.

We have therefore succeeded in precisely characterising the semantic effect of the ultraloose specification transformation. No such characterisation has been published before although we have been informed that Reichel [23] presents a similar result to the above. We have not been able to obtain this paper but base the following comparision on Reichel's book [24 chapter 5] published two years later (which we assume presents essentially the same work).

We note three significant differences between Reichel's framework and our own:

1. The most obvious difference is that Reichel does not discuss ultraloose specifications!

   However, we believe Reichel's work is comparable since he obtains a similar effect by replacing the normal notion of satisfaction of axioms by a notion of behavioural satisfaction of axioms. This notion is based (in the obvious way) on a notion of behavioural equivalence of elements of an algebra which serves a similar rôle to the congruence $\equiv$ in ultraloose specifications.

That said, we believe that a very considerable amount of work would be required to apply Reichel's work to ultraloose specifications. (The principal problem is that models of ultraloose specifications can vary in their interpretation of $\equiv$ whereas, in Reichel's framework, the corresponding notion of behavioural equivalence of elements is completely determined by the other parts of the algebra.)

2. Reichel uses the following (less general) notion of behavioural equivalence.

   For a signature Σ with sorts T and a distinguished subset $I \subseteq T$, two Σ-algebras $\mathcal{A}$ and $\mathcal{B}$ are I-equivalent (written $\mathcal{A} \equiv \mathcal{B} \mod I$) if there is a Σ-algebra $\mathcal{F}$ such that $\mathcal{F} \underset{I}{\overset{T}{\rightarrow}} \mathcal{A}$ and $\mathcal{F} \underset{I}{\overset{T}{\rightarrow}} \mathcal{B}$.

   We believe that this definition is equivalent to $\underset{OUT}{\overset{IN}{\longleftrightarrow}}$ with $IN = \mathbf{Tp}(\Sigma)$ and $OUT = I$. The restriction that $IN = \mathbf{Tp}(\Sigma)$ avoids the need to use reachable quantification since, with $IN = \mathbf{Tp}(\Sigma)$, $\forall^{\Sigma'}_{IN} x{:}\, \tau.\ P$ and $\forall x{:}\, \tau.\ P$ are equivalent.

3. (In order to guarantee the existence of initial models) Reichel restricts axioms to be conditional equations of the form

   $$\forall xs{:}\, \tau s.\ l1 =_{\tau 1} r1 \wedge \ldots lm =_{\tau m} rm \Rightarrow l =_{\tau} r$$

   The relative simplicity of axioms of this form avoids the need to resort to negation normal form to detect problematic uses of inequality and prevents the problem of existential quantification from arising.

To summarise: Though there are similarities to Reichel's work, our result applies directly to ultraloose specifications and is considerably more general both in the form of axioms allowed in specifications and (most significantly) in the form of behavioural equivalence used.

There are two immediate corollaries to theorem 4.10.

Corollary 4.11 ($\mathit{Counter}^{\{Bool\}}_{\{Bool\}}$ is behaviourally closed)

The specification $\mathit{Counter}^{\{Bool\}}_{\{Bool\}}$ is a $(\{Bool\}, \{Bool\})$-behaviourally closed specification of counters.

End Corollary.

This seems to directly contradict Schoett's "impossibility theorem" (theorem 3.21) that one cannot specify a behaviourally closed set of counter-like algebras using only a *finite* set of axioms.

In fact, there is no contradiction: Schoett's result applies to flat specifications whereas the specification $\mathit{Counter}^{\{Bool\}}_{\{Bool\}}$ consists not just of axioms but also the hiding operation **export** _ **from** _. Since the late 1970's [16,38] it has been known that some finite specifications with hidden operations cannot be finitely written without hidden operations. It is therefore not too surprising that allowing the hidden operation $\equiv_{Stack}$ allows a finite specification.

The second corollary to theorem 4.10 is that the ultraloose transformation has precisely the same effect as the behavioural abstraction operator.

Corollary 4.12 ($SP^{IN}_{OUT} =$ **behaviour** SP **wrt** (IN, OUT))

Let Σ be a signature, IN and OUT subsets of the sorts of Σ, Ax a set of Σ-axioms and SP the flat Σ-specification $\langle \Sigma, Ax \rangle$.

If all existential quantification in $\mathit{NNF}(\!| Ax |\!)$ is over sorts in IN and all inequations in $\mathit{NNF}(\!| Ax |\!)$ is over sorts in OUT, then

$$SP^{IN}_{OUT} \quad = \quad \textbf{behaviour } SP \textbf{ wrt } (IN, OUT)$$

End Corollary.

Thus, not only have we succeeded in precisely characterising the semantic effect of the ultraloose specification transformation; we have also shown how flat ASL specifications can be transformed into equivalent USL specifications.

4.5 Summary

The introduction to this thesis argues that behavioural closure is an important property for specifications and notes that the "ultraloose specification style" used by Wirsing and Broy [42] appears to produce behaviourally closed specifications but that no-one has stated (or proved) what the precise effect of this style is.

Based on the notions of behavioural equivalence developed in the previous chapter, this chapter provides three results which characterise the effect of the ultraloose style with increasing precision.

The most important of these results is the last one (theorem 4.10) which shows that the semantic effect of the ultraloose transformation is to close a specification under behavioural equivalence (provided certain purely-syntactic side-conditions are met). The value of this result is threefold:

1. It provides a precise characterisation of the semantic effect of the ultraloose transformation;
2. It shows how ASL and USL are related to each other; and
3. It shows that the meaning of Schoett's "impossibility theorem" is *not* that one cannot write useful behaviourally closed specifications but that one must sometimes use hidden operations when writing them.

Chapter 5

Ease of Proofs in ASL and USL

The previous chapter compares ASL with USL from a specifier's point of view: showing when specifications written in each language have the same *meaning*. This chapter compares ASL with USL from an implementor's point of view: considering how easy it is to prove properties of (equivalent) behaviourally closed specifications written in ASL and in USL.

In particular, we consider the (apparently straightforward) task of showing that the specification *MCounter* (figure 5.1) (and an equivalent ultraloose specification) satisfy the following axiom[1]

$$\forall^r n, m{:}\, Nat, c{:}\, Ctr.\ n < m \Rightarrow isZero(mdec(n, minc(m, c))) =_{Bool} \text{False} \qquad (5.1)$$

This is typical of the kind of results one might wish to show about such a specification.

Section 5.1 repeats Schoett's argument [37] that the only known technique for proving the result requires an infinite proof; and section 5.2 shows that a finite proof is possible using the equivalent ultraloose specification. (Equivalence follows from

[1] Except in specifications (where we make quantification explicit) we use $\forall^r$ as an abbreviation for $\forall^{\{zero,inc,dec\}}_{\{Nat\}}$ throughout this chapter.

```
MCounter ≝
enrich Nat + Counter
by sign  minc, mdec: Nat × Ctr → Ctr
axioms
   (MC1)  ∀c: Ctr. minc(0, c) = c
   (MC2)  ∀n: Nat, c: Ctr. minc(succ(n), c) = minc(n, inc(c))
   (MC3)  ∀c: Ctr. mdec(0, c) = c
   (MC4)  ∀n: Nat, c: Ctr. mdec(succ(n), c) = mdec(n, dec(c))
end


Counter ≝
behaviour enrich Bool
          by sign  Ctr: type
                   zero: → Ctr
                   inc, dec: Ctr → Ctr
                   isZero: Ctr → Bool
          axioms
             (C1)  dec(zero) = zero
             (C2)  ∀c: Ctr. dec(inc(c)) = c
             (C3)  isZero(zero) = True
             (C4)  ∀c: Ctr. isZero(inc(c)) = False
          end
wrt ({Bool}, {Bool})
```

Figure 5.1: Multiple Counter — ASL

theorem 4.10 and the fact that *MCounter* does not contain inequations or existential quantification.)

Though useful to users of *MCounter*, the main interest of this result lies not in whether *MCounter* satisfies axiom 5.1 but in the fact that the result is difficult or impossible in ASL but straightforward in USL. That is, this chapter demonstrates a substantial difference in the ease of proving results about specifications written in ASL and USL.

5.1 Difficulty of Proofs in ASL

This section discusses how one might prove that the specification *MCounter* satisfies axiom 5.1. The discussion is divided between a repeat of Schoett's argument from [37 section 5] that a finite proof is impossible using a technique suggested by Sannella and Tarlecki and a demonstration that an alternative approach is unsound.

In essence, the task is to show a result of the form

$$(\textbf{behaviour}\ \mathit{Counter}\ \textbf{wrt}\ (\mathit{IN}, \mathit{OUT})) + \mathit{SP} \models \mathit{ax}$$

Since we behaviourally close *Counter*, the proof should only make use of behavioural properties of *Counter*. That is, the proof should only involve observational axioms satisfied by *Counter*. (Sannella and Tarlecki [27 section 4] give a slightly weaker inference rule — the difference is the restriction to "uniformly quantified" axioms discussed in section 3.5 and is not significant in the following.)

Schoett [37 p. 619] argues as follows (we have substituted our notation and references):

> "In a finite proof about the counter data type specified by **behaviour** *Counter* **wrt** $(\{\mathit{Bool}\}, \{\mathit{Bool}\})$, the proof rule of Sannella and Tarlecki can only be applied a finite number of times to yield a finite number of axioms in $\mathbf{Axm}(\{\mathit{Bool}\}, \{\mathit{Bool}\})$. Theorem 3.21 tells us that this set of axioms has a model $\mathcal{B}$ with a number n such that for all terms c composed of *zero*, *inc* and *dec*, we have $\mathit{isZero}(\mathit{dec}^n(c))_{\mathcal{B}} = \mathrm{True}_{\mathcal{B}}$. Putting $c = \mathit{inc}^m(\mathit{zero})$, we obtain
>
> $$\begin{aligned}&\mathit{isZero}(\mathit{mdec}(n, \mathit{minc}(m, \mathit{zero})))\\ &=\\ &\mathit{isZero}(\mathit{dec}^n(\mathit{inc}^m(\mathit{zero})))\\ &=\\ &\mathrm{True}\end{aligned}$$
>
> Since this holds for all m, the law desired by the user is false in $\mathcal{B}$. The law therefore cannot be a consequence of a finite set of formulas in

Axm$(\{Bool\}, \{Bool\})$ and thus cannot be finitely proved with the proof rule of Sannella and Tarlecki."

So, by Schoett's argument, we cannot show the result if we treat *Counter* as a "black box" of which we can only observe (a finite set of) observational axioms. However, it is clear that we can prove a similar result if we "ignore" the behavioural abstraction. That is, we could show a result of the form

$$Counter + SP \models ax$$

Unfortunately, it is not generally sound to conclude from this that

$$(\textbf{behaviour}\ Counter\ \textbf{wrt}\ (IN, OUT)) + SP \models ax$$

as the following lemma shows.

Lemma 5.1 (ignoring behavioural abstraction is unsound)

There exist specifications *SP1* and *SP2* and an observational axiom *ax* such that

$$SP1 + SP2 \models ax \quad \not\Rightarrow \quad \textbf{behaviour}\ SP1\ \textbf{wrt}\ (IN, OUT) + SP2 \models ax$$

Proof

Consider the following specifications

$$SP1 \overset{\text{def}}{=} \langle \tau\ \textbf{:type}\ c :\rightarrow \tau\ , \forall x{:}\tau.\ x = c \rangle$$

$$SP2 \overset{\text{def}}{=} \langle \tau\ \textbf{:type}\ c :\rightarrow \tau\ , \neg\forall x{:}\tau.\ x = c \rangle$$

To see that $(\textbf{behaviour}\ SP1\ \textbf{wrt}\ (\{\}, \{\tau\})) + SP2$ is consistent consider the model $\mathcal{A} \overset{\text{def}}{=} \langle \tau = \{1, 2\}, c = 1 \rangle$. Since $\mathcal{A}{:}\, SP2$ and $\mathcal{A}$'s reachable subalgebra is a model of *SP1* and behaviourally equivalent to *SP2*, $\mathcal{A}$ is a model of $(\textbf{behaviour}\ SP1\ \textbf{wrt}\ (\{\}, \{\tau\})) + SP2$.

Since (**behaviour** *SP1* **wrt** $(\{\},\{\tau\})) + SP2$ is consistent, we have that

$$(\textbf{behaviour}\ \mathit{SP1}\ \textbf{wrt}\ (\{\},\{\tau\})) + \mathit{SP2} \not\models \mathit{false}$$

Whereas, *SP1* + *SP2* is obviously inconsistent and therefore

$$\mathit{SP1} + \mathit{SP2} \models \mathit{false}$$

It follows that

$$\mathit{SP1} + \mathit{SP2} \models ax \quad \not\Rightarrow \quad \textbf{behaviour}\ \mathit{SP1}\ \textbf{wrt}\ (\{\},\{\tau\}) + \mathit{SP2} \models ax$$

End Lemma.

We conclude that the two obvious ways of showing the result cannot be used: the first is not finitely complete and the second is not sound. This *seems to suggest* that, since the obvious techniques cannot be applied, it is at least "difficult" and at worst "impossible" to prove the result for the ASL specification. We return to this topic in section 5.3.

5.2 Ease of Proofs in USL

This section discusses how one might prove that the ultraloose specification *MCounter2* (figure 5.2) satisfies axiom 5.1. Though typical of the kind of result about *MCounter2* one might wish to show, the principal interest lies not in the result but in the fact that it is perfectly straightforward to prove — in marked contrast to our experience in section 5.1 for the equivalent ASL specification.

As with the ASL specification, it is obvious that the result is true. It seems that we can prove the result by proving that *MCounter2* satisfies some simple properties such as

$$\forall^r n, m\colon Nat, c\colon Ctr.\ n \leq m \Rightarrow mdec(n, minc(m, c)) \equiv minc(m - n, c)$$
$$\forall^r m\colon Nat, c\colon Ctr.\ 0 < m \Rightarrow isZero(minc(m, c)) = \text{False}$$

$MCounter2 \overset{\text{def}}{=}$
enrich $Nat + Counter2$
by sign $minc, mdec\colon Nat \times Ctr \to Ctr$
axioms
$(MC1')$ $\forall c\colon Ctr.\ minc(0, c) = c$
$(MC2')$ $\forall n\colon Nat, c\colon Ctr.\ minc(succ(n), c) = minc(n, inc(c))$
$(MC3')$ $\forall c\colon Ctr.\ mdec(0, c) = c$
$(MC4')$ $\forall n\colon Nat, c\colon Ctr.\ mdec(succ(n), c) = mdec(n, dec(c))$
end

$Counter2 \overset{\text{def}}{=}$
hide $\equiv$ **in**
enrich $Bool$
by sign Ctr: **type**
$zero\colon \to Ctr$
$inc, dec\colon Ctr \to Ctr$
$isZero\colon Ctr \to Bool$
$\equiv\colon Ctr \times Ctr \to Bool$
IN $= Bool$
axioms
$(C1')$ $dec(zero) \equiv zero$
$(C2')$ $\forall^r c\colon Ctr.\ dec(inc(c)) \equiv c$
$(C3')$ $isZero(zero) = \text{True}$
$(C4')$ $\forall^r c\colon Ctr.\ isZero(inc(c)) = \text{False}$

$\forall c\colon Ctr.\ c \equiv c$
$\forall c1, c2\colon Ctr.\ c1 \equiv c2 \Leftrightarrow c2 \equiv c1$
$\forall c1, c2, c3\colon Ctr.\ c1 \equiv c2 \land c2 \equiv c3 \Rightarrow c1 \equiv c3$

$\forall c1, c2\colon Ctr.\ c1 \equiv c2 \Rightarrow inc(c1) \equiv inc(c2)$
$\forall c1, c2\colon Ctr.\ c1 \equiv c2 \Rightarrow dec(c1) \equiv dec(c2)$
$\forall c1, c2\colon Ctr.\ c1 \equiv c2 \Rightarrow isZero(c1) = isZero(c2)$
end

Figure 5.2: Multiple Counter — USL

from which the result easily follows.

We cannot show these results directly because the operation $\equiv$ is hidden in *MCounter2*. However, it is straightforward to transform *MCounter2* into an equivalent specification and to show that these results hold in this equivalent specification. (This ability to transform the specification into a more convenient form is the principal difference from the ASL case!)

Expanding *Counter2*, the specification in figure 5.2 is of the form

```
enrich  Nat +
            hide ≡
            in enrich BoolBase
               by sign CtrΣ
                    axioms CtrAx
                 end
by sign mdec, minc : Nat × Ctr → Ctr
     axioms MCtrAx
  end
```

By moving the **hide** operation out and rearranging the use of + and **enrich**, we obtain an equivalent specification of the form[2]

```
hide ≡
in enrich Nat + BoolBase
   by sign mdec, minc : Nat × Ctr → Ctr
              CtrΣ
        axioms CtrAx
                   MCtrAx
     end
```

shown in figure 5.3.

It is now straightforward, if a little tedious, to show the result. We shall use the following two lemmas in the proof.

Lemma 5.2

$$\textit{Counter3} \models \forall^r m, n\colon Nat, c\colon Ctr.\ minc(m+n, c) \equiv minc(m, minc(n, c))$$

[2]See [40,41] for rules which can be used to demonstrate that these specifications are equivalent.

$MCounter3 \stackrel{\text{def}}{=}$ **hide** $\equiv$ **in** $Counter3$

$Counter3 \stackrel{\text{def}}{=}$
enrich $Bool + Nat$
by sign $minc, mdec\colon Nat \times Ctr \to Ctr$

Ctr: **type**
$zero\colon \to Ctr$
$inc, dec\colon Ctr \to Ctr$
$isZero\colon Ctr \to Bool$
$\equiv\colon Ctr \times Ctr \to Bool$

axioms

($MC1'$) $\forall c\colon Ctr.\ minc(0, c) = c$
($MC2'$) $\forall n\colon Nat, c\colon Ctr.\ minc(succ(n), c) = minc(n, inc(c))$
($MC3'$) $\forall c\colon Ctr.\ mdec(0, c) = c$
($MC4'$) $\forall n\colon Nat, c\colon Ctr.\ mdec(succ(n), c) = mdec(n, dec(c))$

($C1'$) $dec(zero) \equiv zero$
($C2'$) $\forall_{\emptyset}^{\{zero,inc,dec\}} c\colon Ctr.\ dec(inc(c)) \equiv c$
($C3'$) $isZero(zero) = \text{True}$
($C4'$) $\forall_{\emptyset}^{\{zero,inc,dec\}} c\colon Ctr.\ isZero(inc(c)) = \text{False}$

$\forall c\colon Ctr.\ c \equiv c$
$\forall c1, c2\colon Ctr.\ c1 \equiv c2 \Leftrightarrow c2 \equiv c1$
$\forall c1, c2, c3\colon Ctr.\ c1 \equiv c2 \wedge c2 \equiv c3 \Rightarrow c1 \equiv c3$

$\forall c1, c2\colon Ctr.\ c1 \equiv c2 \Rightarrow inc(c1) \equiv inc(c2)$
$\forall c1, c2\colon Ctr.\ c1 \equiv c2 \Rightarrow dec(c1) \equiv dec(c2)$
$\forall c1, c2\colon Ctr.\ c1 \equiv c2 \Rightarrow isZero(c1) = isZero(c2)$

end

Figure 5.3: Restructured Multiple Counter

Lemma 5.3

$$Counter3 \models \forall^{r} m\colon Nat, c\colon Ctr.\ mdec(m, minc(m, c)) \equiv c$$

Throughout these proofs we silently make use of the reflexivity of $\equiv$ so that whenever we establish that $c1 = c2$, we can conclude that $c1 \equiv c2$.

Lemma 5.2 $(minc(m+n, c) \equiv minc(m, minc(n, c)))$

$$Counter3 \models \forall^r m, n\colon Nat, c\colon Ctr.\ minc(m+n, c) \equiv minc(m, minc(n, c))$$

Proof

Since $Counter3 \models \forall n\colon Nat.\ n \in \{0, succ\}(\emptyset)$, we can use structural induction with respect to *0* and *succ* to prove the result.

Base case: $n = 0$

$minc(m, minc(0, c))$
$= \quad \{ (MC1') \}$
$minc(m, c)$
$= \quad \{ m + 0 = m \}$
$minc(m + 0, c)$
$\square$

Inductive Step: $n = succ(n')$

Suppose that $minc(m, minc(n', c)) \equiv minc(m + n', c)$. Then

$minc(m, minc(succ(n'), c))$
$= \quad \{ (MC2') \}$
$minc(m, minc(n', inc(c)))$
$\equiv \quad \{$ Inductive Assumption $\}$
$minc(m + n', inc(c))$
$= \quad \{ (MC2') \}$
$minc(succ(m + n'), c)$
$= \quad \{ succ(m + n') = m + succ(n') \}$
$minc(m + succ(n'), c)$
$\square$

Since $minc(m, minc(0, c)) = minc(m+0, c)$ and $minc(m, minc(n', c)) \equiv minc(m+n', c) \Rightarrow minc(m, minc(succ(n'), c)) \equiv minc(m + succ(n'), c)$, we conclude that

$$Counter3 \models \forall^r m, n\colon Nat, c\colon Ctr.\ minc(m+n, c) \equiv minc(m, minc(n, c))$$

End Lemma.

Lemma 5.3 ($mdec(n, minc(n, c)) \equiv c$)

$$Counter3 \models \forall^r n\colon Nat,\ c\colon Ctr.\ mdec(n, minc(n, c)) \equiv c$$

Proof

Since $Counter3 \models \forall n\colon Nat.\ n \in \{0, succ\}(\emptyset)$, we can use structural induction with respect to 0 and $succ$ to prove the result.

Base case: $n = 0$

$mdec(0, minc(0, c))$
$=$ $\{ (MC1') \}$
$mdec(0, c)$
$=$ $\{ (MC3') \}$
c
$\square$

Inductive Step: $n = succ(m)$

Suppose that $mdec(m, minc(m, c)) \equiv c$. Then

$mdec(succ(m), minc(succ(m), c))$
$=$ $\{ succ(m) = 1 + m$, lemma 5.2 $\}$
$mdec(succ(m), minc(1, minc(m, c)))$
$=$ $\{ 1 = succ(0), (MC2'), (MC1') \}$
$mdec(succ(m), inc(minc(m, c)))$
$=$ $\{ (MC4') \}$
$mdec(m, dec(inc(minc(m, c))))$
$\equiv$ $\{ (C2') \}$
$mdec(m, minc(m, c))$
$\equiv$ $\{$ Inductive assumption $\}$
c
$\square$

Since $mdec(0, minc(0, c)) = c$ and $mdec(m, minc(m, c)) \equiv c \Rightarrow mdec(succ(m), minc(succ(m), c)) \equiv c$, we conclude that

$$Counter3 \models \forall^r n\colon Nat,\ c\colon Ctr.\ mdec(n, minc(n, c)) \equiv c$$

End Lemma.

We can now prove the main result of this section: that *MCounter3* (and, hence, *MCounter2*) satisfies axiom 5.1.

Theorem 5.4 $(n < m \Rightarrow \neg isZero(mdec(n, minc(m, c))))$

$MCounter3 \models \forall^r n, m\colon Nat, c\colon Ctr.\ n < m \Rightarrow isZero(mdec(n, minc(m, c))) =_{Bool}$ False

Proof

$\forall^r n, m\colon Nat, c\colon Ctr.\ n < m \Rightarrow isZero(mdec(n, minc(m, c))) =_{Bool}$ False
$=$ { arithmetic }
$\forall^r n, m\colon Nat, c\colon Ctr.\ n < m \Rightarrow isZero(mdec(n, minc(n + (m - n), c))) =_{Bool}$ False
$=$ { lemma 5.2 }
$\forall^r n, m\colon Nat, c\colon Ctr.\ n < m \Rightarrow isZero(mdec(n, minc(n, minc(m - n, c)))) =_{Bool}$ False
$=$ { lemma 5.3 }
$\forall^r n, m\colon Nat, c\colon Ctr.\ n < m \Rightarrow isZero(minc(m - n, c)) =_{Bool}$ False
$=$ { arithmetic }
$\forall^r n\colon Nat, c\colon Ctr.\ n > 0 \Rightarrow isZero(minc(n, c)) =_{Bool}$ False
$=$ { arithmetic }
$\forall^r n\colon Nat, c\colon Ctr.\ n \geq 0 \Rightarrow isZero(minc(1 + n, c)) =_{Bool}$ False
$=$ { lemma 5.2, $1 = succ(0)$, $(MC2')$, $(MC1')$ }
$\forall^r n\colon Nat, c\colon Ctr.\ n \geq 0 \Rightarrow isZero(inc(minc(n, c))) =_{Bool}$ False
$=$ { $(C3')$ }
$\forall^r n\colon Nat, c\colon Ctr.\ n \geq 0 \Rightarrow$ False $=_{Bool}$ False
$=$ { predicate calculus }
true
□

End Theorem.

5.3 Comparision

The previous two sections demonstrate a substantial difference in the ease of proving results about specifications written in ASL and USL. We have shown that it is hard to prove that the ASL specification *MCounter* satisfies the axiom

$$\forall^r n, m\colon Nat, c\colon Ctr.\ n < m \Rightarrow isZero(mdec(n, minc(m, c))) =_{Bool} \text{False}$$

but that it is straightforward to prove that the equivalent ultraloose specification *MCounter2* satisfies this axiom.

We considered two techniques for proving the result for the ASL specification:

1. To maintain soundness, we might restrict ourselves to using observational axioms in the proof as suggested in [27 section 4].

   Unfortunately, Schoett has shown that no finite proof exists under this restriction.

2. To achieve a finite proof, we might just "ignore" the behavioural abstraction.

   Unfortunately, this is easily shown to be unsound. That is, even for an observational axiom ax, we cannot conclude that

   $$\textbf{behaviour } \mathit{SP1} \textbf{ wrt } (\mathit{IN}, \mathit{OUT}) + \mathit{SP2} \models ax$$

   if

   $$\mathit{SP1} + \mathit{SP2} \models ax$$

   holds.

These problems with behavioural abstraction in ASL seem to support Wirsing and Broy's suggestion [42 paragraph 4] that the behavioural abstraction operator is "mathematically difficult." They also suggest an alternative use of the ultraloose transformation: one might develop specifications using ASL and then transform them into the equivalent (but somewhat longer) ultraloose specification to eliminate the use of behavioural abstraction and, hence, simplify proofs.

However, a recent idea of Sannella and Tarlecki [32 section 6] suggests a way round the problems in ASL based on Schoett's idea of stability which we briefly describe.

In Sannella and Tarlecki's framework for program development (as described in [29,30]), any function $\kappa\colon \mathbf{Alg}(\Sigma) \to \mathbf{Alg}(\Sigma')$ gives rise to a specification building operator $\bar{\kappa}\colon \mathbf{Spec}(\Sigma) \to \mathbf{Spec}(\Sigma')$ defined by

$$\mathbf{Mod}(\bar{\kappa}(\mathit{SP})) \overset{\text{def}}{=} \{\mathcal{A}\colon \mathcal{A} \in \mathbf{Mod}(\mathit{SP})\colon \kappa(\mathcal{A})\}$$

Such specification building operators are known as "constructors." For example, the specification building operators **derive**, **quotient** and **extend _ to _via _** are constructors.

The notion of constructors is important because some constructors can be easily implemented using programming language constructs and so a useful step in developing a program is to refine a specification SP to a specification of the form $\bar{\kappa}(SP')$ — replacing part of SP by an easily implemented constructor $\bar{\kappa}$. (Considerably more detail may be found in [29].)

A constructor $\bar{\kappa}$ is said to be "stable" (with respect to an equivalence relation $\equiv$) if, for any Σ-algebras $\mathcal{A}$ and $\mathcal{B}$,

$$\mathcal{A} \equiv \mathcal{B} \;\Rightarrow\; \kappa(\mathcal{A}) \equiv \kappa(\mathcal{B})$$

That is, a constructor is stable if the function on which it is based doesn't introduce differences between equivalent algebras.

The practical consequence of these ideas is the following:

> If all specification building operators provided by a specification language are stable (with respect to behavioural equivalence), then the straightforward proof technique of "ignoring" behavioural abstraction is valid.[3]

In other words, if we are willing to take the (entirely reasonable) step of restricting ourselves to a stable subset of ASL, the problems encountered in section 5.1 disappear.

5.4 Summary

This chapter considers the question of how ultraloose specifications compare with equivalent ASL specifications from the implementors point of view. In particular, it considers what one can prove about a a behaviourally closed specification.

[3]The reason for the problems in lemma 5.1 above is that + is not stable.

We found that, when the behaviourally closed specification is written using the behavioural abstraction operator **behaviour**, the two obvious ways of proving that the specification satisfies an axiom could not be used. This supports Wirsing and Broy's suggestion [42 paragraph 4] that the behavioural abstraction operator is "mathematically difficult" although we note that these problems disappear if we restrict ourselves to "stable" specification building operators.

Chapter 6

Summary and Conclusions

This thesis is concerned with how one might avoid overspecification when writing specifications. In particular, it examines two alternative approaches to writing behaviourally closed specifications: using a "behavioural abstraction operator" as in ASL; or using reachable quantification and a very stylised form of specification as in USL. The main questions asked in the introduction were:

- **Under what circumstances are USL specifications behaviourally closed?**
- **Under what circumstances do these two approaches give the same result?**
- **For which approach is it easiest to prove properties of the resulting specifications?**

The first two questions were tackled in chapter 4 which showed the following results (for flat specifications).

- Theorem 4.7 shows that flat ultraloose specifications are closed under $\underset{OUT}{\overset{IN}{\longleftrightarrow}}$ provided they contain no inequations.

- Theorem 4.10 uses theorem 4.7 to demonstrate that any flat ultraloose specification SP^{IN}_{OUT} is semantically equivalent to a corresponding ASL specification **behaviour** SP **wrt** (IN, OUT) if SP contains no inequations or existential quantification.

Chapter 5 tackled the third question. Using an argument due to Schoett, we saw that it was surprisingly hard to show that a behaviourally closed ASL specification satisfies a given axiom. Replacing the behaviourally closed specification with an equivalent ultraloose specification, we saw that the proof was quite straightforward. We concluded that, at least in this case, the more explicit style of the ultraloose specification was an advantage. However, ultraloose specifications are somewhat more verbose than their corresponding ASL specifications and so we suggested that it might be convenient to write an ASL specification initially and only transform into the ultraloose style before doing any proofs. (The soundness of this approach follows immediately from theorem 4.10.) An alternative approach would be to work in a language such as Extended ML [26] which restricts the specifier to "stable" specification building operators which allows straightforward proofs even in the presence of behavioural abstraction.

The focus of this thesis is very much on *theoretical* results rather than on their *practical* application. It is therefore worth emphasising the practical consequences of the above results.

- We believe that behavioural closure is an essential property of specifications. An easy way of ensuring that (flat) specifications are behaviourally closed is to use reachable quantification instead of the normal quantification and to use a congruence instead of equality in the way formalised in our ultraloose transformation. The only requirement is that the specification should not contain inequations (theorem 4.7).

- Modern approaches to formal program development emphasise the *gradual* refinement of specifications in a series of *small* steps. We believe (but have not attempted to demonstrate) that the more explicit specification of behavioural equivalence in ultraloose specifications allows specifications to be refined in considerably smaller (and hence, simpler) steps than is possible in ASL. Our demonstration of the relationship between (flat) ASL specifications and USL specifications provides the theoretical justification for transforming ASL specifications into USL specifications in preparation for such transformation. The only requirement is that the specification should not contain inequations or existential quantification (theorem 4.10).

Incidental Results

Answering the above questions required us to generalise Meseguer and Goguen's definition of behavioural equivalence and to define the "ultraloose transformation." We also came across the following interesting incidental results:

- Theorem 3.16 demonstrates that the satisfaction of "observational axioms" is invariant under behavioural equivalence.

  This result is stronger than a similar result by Sannella and Tarlecki [27 Fact 18] for their (weaker) notion of behavioural equivalence. We find this a convincing argument for the use of (our generalisation of) Meseguer and Goguen's definition of behavioural equivalence rather than that of Sannella and Tarlecki.

- Theorem 4.5 shows that specifications written in the ultraloose style ("ultraloose specifications") are downward closed under $\underset{OUT}{\overset{IN}{\rightarrow}}$.

  This suggests a link with the implementation notions of the ADJ group and Ehrig etc al. [4,10]which we did not pursue.

Further Work

Finally, we note the following areas for further work:

- The above results are for *flat* specifications only. It should be straightforward to extend the ultraloose transformation and associated results to handle structured specifications although some care is required with the **derive** operator since it does not preserve closure under isomorphism (counterexample 2.7).

- There are several alternative ways of defining an equivalence $\underset{OUT}{\overset{IN}{\longleftrightarrow}}$ like our behavioural equivalence. To our knowledge, no-one has answered the question of whether or not these any of these equivalences are equivalent to $\equiv_{\mathbf{Axm}(IN,OUT)}$. That is, if two algebras satisfy exactly the same set of observational axioms, are they behaviourally equivalent and vice-versa.

  We know of two partial answers:

  1. For $IN = \emptyset$, it is straightforward to show that the above holds.
  2. A result by Sannella and Tarlecki [27 Fact 16] suggests that this may be true for our definition of behavioural equivalence using *infinitary* observational axioms. We *believe* (but have not tried to prove) that this is true using *finitary* observational axioms in the presence of at most countably many unreachable elements.

- It is common to base the semantics of specification languages on partial algebras (that is algebras which allow partial functions as interpretations of function symbols) rather than total algebras. Partiality allows non-terminating computations and errors to be modelled directly.

  The major problem in trying to apply the results in this thesis to specifications allowing partial functions is that the standard framework for partial algebras requires functions to be strict. This complicates the interpretation of the function $\equiv_\tau : \tau \times \tau \rightarrow Bool$ as relation since it is not clear how to interpret an undefined result from $\equiv_\tau$.

  The most direct line of attack might be to define a logical framework which allowed non-strict functions (or just non-strict predicates). A less radical approach might be to consider a framework in which the inbuilt (non-strict) predicate $=$ is allowed to denote an arbitrary congruence rather than equality.

- Our motivation for considering behavioural equivalence was the desire to capture when it was possible to replace one module by another without changing the overall result of a program.

However, in some circumstances, the definition of behavioural equivalence we adopt (and most other definitions in the literature) is too strong. Consider a program that uses a stack module but for which we are able to show that it never generates stacks with more than 100 elements. In this case, it would be possible to use either an unbounded stack (like those considered in this thesis) or a bounded stack of size 100 or more. However, despite the fact that we can substitute one for the other in this program, bounded stacks and unbounded stacks are not behaviourally equivalent.

(It might be argued that allowing such bounded implementations introduces an undesirable degree of coupling between the module and the rest of the program. But, from a practical point of view, such implementations are very common and it is desirable to be able to be able to prove their correctness with respect to an appropriate specification. Furthermore, from a theoretical point of view, no real computers have an unbounded amount of storage and so it is impossible to implement unbounded data structures.)

An important area for further work is extending both the definition of behavioural equivalence and the ultraloose specification style to handle such cases.

Hennicker [11,12] describes an approach where one (axiomatically) specifies a family of predicates $Obs_{\tau}\colon \tau \to Bool$ which identifies those values which are considered observable. This approach would provide the flexibility required though it is not clear how this approach would interact with the ultraloose specification style.

References

[1] Manfred Broy, B. Möller, Peter Pepper & Martin Wirsing, "Algebraic Implementations Preserve Program Correctness," *Sci. Comput. Programming* 7 (1986).

[2] Manfred Broy & Martin Wirsing, "Ultraloose Algebraic Specification," *Bull. European Assoc. Theoret. Comput. Sci.* 35 (June 1988), 117–127.

[3] Chang & Keisler, *Model Theory*, Studies in Logic and the Foundations of Mathematics #73, North-Holland, 1973.

[4] Hartmut Ehrig, H.-J. Kreowski, Bernd Mahr & P. Padawitz, "Compound algebraic implementations: an approach to stepwise refinement of software systems.," *Lect. Notes in Comp. Sci.* 88 (Sept. 1–5, 1980), 231–245.

[5] Hartmut Ehrig & Bernd Mahr, *Fundamentals of Algebraic Specification 1 — Equations and Initial Semantics*, EATCS Monographs on Theoretical Computer Science #6, Springer-Verlag, New York–Heidelberg–Berlin, 1985.

[6] Marie-Claude Gaudel, "Structuring and Modularising Algebraic Specifications: the PLUSS Specification Language, Evolution and Perspectives," *Lect. Notes in Comp. Sci.* 577 (1992), 3–21.

[7] V. Giarratana, F. Gimona & U. Montanari, "Observability concepts in abstract data type specifications," *Lect. Notes in Comp. Sci.* 45 (1976), 567–578.

[8] Joseph A. Goguen & José Meseguer, "Universal Realization, Persistent Interconnection and Implementation of Abstract Modules," *Lect. Notes in Comp. Sci.* 134 (1982).

[9] Joseph A. Goguen, James W. Thatcher & Eric G. Wagner, "An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types," in *Advances in Computer Programming*, Yeh, ed. #4, 1978, 80–149 .

[10] Joseph A. Goguen, James W. Thatcher, Eric G. Wagner & Jesse B. Wright, "Initial Algebra Semantics and Continuous Algebras," *J. Assoc. Comput. Mach.* 24, 1 (Jan. 1977), 68–95.

[11] R. Hennicker, "Observational Implementations," *Lect. Notes in Comp. Sci.* 349 (1988), 59–71.

[12] R. Hennicker, "Implementation of Parameterised Observational Specifications," *Lect. Notes in Comp. Sci.* 351 (1989), 290–305.

[13] C. A. R. Hoare, "Proof of Correctness of Data Representations," *Acta Inform.* 1 (1972), 272–281.

[14] T. S. E. Maibaum, M. R. Sadler & P. A. S. Veloso, "Logical Implementation," Department of Computing, Imperial College, London, Technical Report, August 1983.

[15] T. S. E. Maibaum, P. A. S. Veloso & M. R. Sadler, "A theory of abstract data types for program development: bridging the gap?," in *Mathematical Foundations of Software Development '85* #186, Springer-Verlag, New York–Heidelberg–Berlin, 1985, 214–230.

[16] M. E. Majster, "Limits of the "algebraic" specification of abstract data types," *ACM SIGPLAN Notices* 12 (1977), 37–42.

[17] José Meseguer & Joseph A. Goguen, "Initiality, induction and computability," in *Algebraic Methods in Semantics*, M. Nivat & J. Reynolds, eds., Cambridge Univ. Press, New York, NY, 1983, 460–541.

[18] Carroll C. Morgan, "Data Refinement by Miracles," *Information Processing Letters* 26 (1987/88), 243–246.

[19] Joseph M. Morris, "Laws of Data Refinement," *Acta Inform.* 26 (1989), 287–308.

[20] P. Nivela & F. Orejas, "Initial behaviour semantics for algebraic specifications," *Lect. Notes in Comp. Sci.* 332 (1987), 184–207.

[21] Axel Poigné, "Partial Algebras, Subsorting and Dependent Types — Prerequisites of Error Handling in Algebraic Specifications," *Lect. Notes in Comp. Sci.* 332 (1987), 208–234.

[22] A. Ramsay, *Formal methods in Artificial Intelligence*, Cambridge Tracts in Theoretical Computer Science #6, Cambridge Univ. Press, New York, NY, 1988.

[23] Horst Reichel, "Initial restrictions of behaviour," *Proceedings The Role of Abstract Models in Information Processing* (1985).

[24] Horst Reichel, *Initial computability, algebraic specifications and partial algebras*, The International series of monographs on Computer Science #2, Clarendon Press, Oxford, 1987.

[25] Donald T. Sannella, "Formal Specification of ML Programs," LFCS, Univ. of Edinburgh, Research Report, 1985.

[26] Donald T. Sannella & Andrzej Tarlecki, "Extended ML: an Institution-independent Framework for Formal Program Development," LFCS, University of Edinburgh, Research Report, 1985.

[27] Donald T. Sannella & Andrzej Tarlecki, "On Observational Equivalence and Algebraic Specification," *J. Comput. System Sci.* 34 (1987), 150–178.

[28] Donald T. Sannella & Andrzej Tarlecki, "Specifications in an Arbitrary Institution," *Information and Computation* 76 (1988), 165–210.

[29] Donald T. Sannella & Andrzej Tarlecki, "Toward Formal Development of Programs from Algebraic Specifications: Implementations Revisited," *Acta Inform.* 25 (1988), 233-281.

[30] Donald T. Sannella & Andrzej Tarlecki, *A kernel specification formalism with higher-order parameterisation*, Dept. of Computer Science, Univ. of Edinburgh, 1992.

[31] Donald T. Sannella & Andrzej Tarlecki, "Program Specification and Development in Standard ML," *Proceedings 12th ACM Symposium on Principles of Programming Languages*, New Orleans (Jan. 1985).

[32] Donald T. Sannella & Andrzej Tarlecki, "Toward formal development of programs from algebraic specifications: model-theoretic foundations," Dept. of Computer Science, Univ. of Edinburgh, ECS-LFCS-92-204, Mar. 1992.

[33] Donald T. Sannella, Andrzej Tarlecki & Stefan Sokolowski, "Toward formal development of programs from algebraic specifications: parameterisation revisited," FB3 — Mathematik/Informatik, Universität Bremen, Draft report, Feb. 16, 1990.

[34] Donald T. Sannella & Martin Wirsing, "A kernel language for algebraic specification and implementation," *Lect. Notes in Comp. Sci.* 158 (1983), 413–427.

[35] Oliver Schoett, "Data Abstraction and the correctness of modular programming," Univ of Edinburgh, CST-42-87, 1986, Ph.D. thesis .

[36] Oliver Schoett, "An observational subset of first-order logic cannot specify the behaviour of a counter," *Lect. Notes in Comp. Sci.* 480 (1991), 499–510, extended abstract.

[37] Oliver Schoett, "Two Impossibility Theorems on Behaviour Specification of Abstract Data Types," *Acta Inform.* 29 (1992), 595–621.

[38] J. W. Thatcher, E. G. Wagner & J. B. Wright, "Data Type Specification: Parameterisation and the Power of Specification Techniques," *ACM Transactions on Programming Languages and Systems* 4, 4 (Oct. 1982), 711–732, (abbreviated version presented at 10th Annual Symposium on Theory of Computing, San Diego, California, May 1–3, 1978.).

[39] M. Wand, "Final algebra semantics and data type extensions," *Journal of Computer and System Sciences* 19 (1981), 27–44.

[40] Martin Wirsing, "Structured Algebraic Specification: a kernel language," *Theoretical Computer Science* 42 (1986), 123–249.

[41] Martin Wirsing, "Algebraic Specifications," in *Handbook of Theoretical Computer Science, Formal Models and Semantics, Volume B*, Elsevier, Amsterdam–New York, 1990, 677–788, Chapter 13.

[42] Martin Wirsing & Manfred Broy, "A Modular Framework for Specification and Implementation," *Lect. Notes in Comp. Sci.* 351 (1989), 42–73.