LFCS

Laboratory for Foundations of Computer Science
Department of Computer Science - University of Edinburgh

Towards a Formal Framework for Evaluation of Operational Semantics Specifications

by

Fabio da Silva

LFCS Report Series ECS-LFCS-90-126

LFCS
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ

December 1990

Towards a Formal Framework for Evaluation of Operational Semantics Specifications

Fabio da Silva
LFCS - Department of Computer Science,
University of Edinburgh,
fabio@uk.ac.ed.lfcs

Abstract

This paper describes the theoretical basis of a framework for writing and executing operational semantics descriptions of various aspects of programming languages. We consider a such framework as a fundamental step towards an environment for semantic based formal program manipulation in a broader sense. The framework described here share some common characteristics with other work in semantic based evaluation, e.g. the Centaur System [BCD$^+$87], and the Animator Generator [Ber90]. Our approach differs from these works mainly in (1) the formal characterisation of an operational semantics, and (2) the meta-language of semantics specification, and its evaluation model. Regarding (1), we present a notion of operational semantics in the framework of first order signatures and first order structures. Our objective is to present operational semantics specification in a form that is more amenable to formal manipulation of the kind done by semi-automatic theorem provers. As far as (2) is concerned, we define a meta-language in which only a class of *deterministic specifications* can be expressed. For this meta-language we define an evaluation model that describes how deterministic specifications can be evaluated. In this work we present our characterisation of operational semantics, define the meta-language, and describe its evaluation model. We then describe how another sub-class of semantics specifications, called *determinate semantics*, can be transformed into deterministic ones. This transformation shows how the evaluation model can be applied to a wider class of specifications without changing its complexity.

Contents

1 Introduction

This paper summarises the work we have been doing for the last year towards the definition of a framework for writing and executing operational semantics specifications. Similar frameworks have been proposed, and some of them have received actual implementations. One of the motivation for such frameworks is that the formal manipulation of real size semantics specifications (e.g. the definition of Standard ML [HMT89]) need some kind of machine assistance.

The Centaur System [BCD$^+$87] is one step towards this automatic assistance, which has facilities for helping the writing and testing of operational semantics specifications. In this system, the meta-language TYPOL [Des88] is an implementation of the Natural Semantics formalism [Kah87], in which semantic specifications can be written, and executed. Berry's Animator Generator [Ber90], although designed with other objectives in mind, is another example of a system that has facilities for the prototyping of programming languages, including a specification meta-language in which a restricted form of structural operational semantics can be written.

In both systems, an evaluation model is defined to execute the semantic specifications written in their meta-languages. TYPOL specifications are compiled into MU-Prolog [Nai83], and then the Prolog program is evaluated according to the usual depth-first left-to-right evaluation strategy. The Animator Generator has its own evaluation model for the LSL (Language Specification Language), which does not rely on any other evaluation strategy.

One central problem with the Centaur approach is that it creates a gap between the underlying theory of the semantics formalism, and the behaviour of the evaluation model as defined by the search strategy used by the Prolog compiler. For instance, although the theory of Natural Semantics regard the order in which semantic rules appear as insignificant, in fact, since TYPOL is compiled into MU-Prolog, the order of the rules has the same meaning as in the MU-Prolog evaluation strategy. The problem then is that changing the order of the rules in a specification may lead to non-termination of its evaluation. In the Animator Generator, on the other hand, the evaluation model of the meta-language is considered as the underlying theory of the semantics formalism. Therefore, no semantic gap exists, since both concepts are made to coincide, although rather artificially.

We do not agree with both approaches because in some way most of the clarity, abstraction, and formality that comes with the semantics formalism is lost, which is unfortunate. Moreover, the semi-formal treatment of issues like side conditions in semantic rules given in both approaches, makes difficult any formal reasoning about the semantic specifications.

We follow yet another approach when designing and presenting our framework. We start by defining a formal notion of *operational semantics*, in which inductive systems [Acz77] play the central rôle in the semantic specifications. A *proof search strategy* is then defined that specifies how to evaluate such semantic specifications. This strategy can be proved correct according to the underlying theory, but it fails to have an efficient implementation.

Therefore, the second step in the definition of the framework is to present an efficient evaluation model, even though some expressiveness power in the semantic specifications has to be sacrificed in the process. This evaluation model is defined three stages, which are outlined below.

First, the points in the proof search strategy in which, for various reasons (see section 4), a non-deterministic choice has to be made during the evaluation are identified. Then a *specification meta-language* is defined in which only a subset of the theory can be expressed, namely the class of *deterministic specifications.* The motivation for the definition of the meta-language is that for each non-deterministic choice identified in the previous stage, a restriction on the meta-language is defined that rules out the possibility of the occurrence of such a choice during the evaluation. Finally, we define an *evaluation model* for the meta-language, which, together with the definition of the underlying theory, constitutes the basis of a formal framework for the evaluation of a restricted form of operational semantics specifications.

We believe that this is a good way develop a framework for operational semantics for three main reasons. First, the definition of the underlying theory opens the possibility of using the framework, together with other tools like theorem provers, for formal manipulation of semantic specifications. Second, the way the meta-language was designed emphasizes which kinds of semantic specifications are ruled out, i.e., which subset of the underlying theory is being addressed when using this particular evaluation model. Finally, it leaves open the possibility of other evaluation models, perhaps to address other subsets of the theory, to be defined without any alteration to the underlying theory itself.

Although the presentation of the framework is completed within the definition of the evaluation model, it is sometimes very tedious to present some semantics in the form of deterministic specifications. However, a large number of semantics can be expressed in a form called *determinate specification,* which allows more readable and concise specifications. Therefore, the final step in the description of the framework for the evaluation of operational semantics is to show how determinate specifications can be automatically converted into semantically equivalent deterministic specifications by a process called *factorisation.* This is an interesting result for (1) it shows that determinate specifications are not more general than deterministic ones w.r.t the class of semantics that can be expressed in them, (2) it demonstrates the suitability of the theoretical framework in the formal manipulation of semantic specifications, and (3) it extends the applicability of the evaluation model.

This paper is organised as follows: section 2 presents a formalisation of operational semantics. In section 3 a proof search strategy for inductive definitions is given. Section 4 defines a formal meta-language which only the class of deterministics semantic specifications can be expressed in. Section 5 defines an evaluation model for the class of deterministic specifications. In section 6, we discuss how determinate semantics specifications can be factorised into deterministic equivalent ones, and give some examples of factorisations of common semantics specifications. The factorisation algorithm is presented in section 7, and in section 8 we prove its correctness. Finally section 9 shows presents some conclusions, and areas for future research.

2 A Formal Characterisation of Operational Semantics

In this section a notion of *relational semantics* is defined, and then specialised to obtain a definition of *operational semantics*, where relations are inductively defined. The objective of this characterisation is twofold: (1) to have a form of semantic specifications that is suitable for uniform formal treatment and manipulation of the kind done by theorem provers or proof editors; (2) to define the basic underlying theory of a practical framework in which interpreters, compilers, and debuggers for programming languages can be semi-automatically generated from operational semantics specifications.

Although there is a large number of definitions of operational semantics in the literature ([Plo81], [Kah87], and [Ast89], to give just a few examples), there are two main aspects of these definitions that make then unsuitable for the kind of framework proposed in this work. First, even though it is possible to formally reason about particular semantic specifications when using some of the existent definitions cited above, the formal manipulation that is achieved does not provide an uniform treatment that could be applied generically to a whole class of specifications. Moreover, if we want to define generic functions to manipulate any operational semantic specification of some type, those functions must be defined over a rigorously defined abstract object, and not over a particular specification.

The second aspect is that although the underlying theory of operational semantics is considered by many as being very simple, fairly often what is considered as a formal semantic specification has components that do not have formal definitions. Since our objective is to use the specifications with tools like theorem provers, each component of the semantic specifications must have been formally defined. For instance, in most of the descriptions of an inference rule style of operational semantics, which includes [Ast89] and [Kah87] among others, the issue of side conditions is not formalised

The two aspects discussed above justify the level of formalisation we want to achieve in our characterisation of operational semantics. In order to motivate the formal definitions we informally present one small example of a semantic specification, which presents the dynamic semantics of a small language of expressions as a set of inference rules. The expression language has constant values denoted by n, variables x whose values are in an environment E, and one prefix operator called *plus*. The rules below should be read as natural deduction style rules, with premisses above the horizontal line and a conclusion underneath the line. The meaning of a such rule is that if we can prove each of its premisses we can infer its conclusion.

$$\frac{}{\mathrm{E} \vdash n \rightarrow n}$$

$$\frac{\mathrm{E}(x) = v}{\mathrm{E} \vdash x \rightarrow v} \qquad \text{if } x \in \mathrm{dom\ E}$$

$$\frac{\mathrm{E} \vdash e_1 \rightarrow v_1 \qquad \mathrm{E} \vdash e_2 \rightarrow v_2}{\mathrm{E} \vdash \mathit{plus}(e_1, e_2) \rightarrow v_1 + v_2}$$

where: n is a constant expression, e.g. 3, 24 or 0; x is a variable and v is its value in the environment E; the side condition restricts the applicability of the second rule to the cases when x is defined in E; e_1 and e_2 are general expressions, with v_1 and v_2 as their values respectively; and $+$ is the plus operation on values.

Intuitively, the above rules are defining a ternary relation $\vdash\!\!\rightarrow$ among *environment*, *expressions* and *values*, which is a set of triples like (E, e, v) such that $(\mathrm{E}, e, v) \in \vdash\!\!\rightarrow$ iff v is the value of e in an environment E, e.g. $(\emptyset, \mathit{plus}(3,4), 7) \in \vdash\!\!\rightarrow$ and $(\emptyset, \mathit{plus}(5,10), 48) \notin \vdash\!\!\rightarrow$ (where $\emptyset$ denotes the empty environment). As can be easily seen in the above example, it is not obvious why this particular style of presenting a semantics should be called operational. Moreover, the meanings of the side condition, the use of the environment in the premiss of the second rule, and the application of the $+$ operator in the thir rule are not very clear. In the next sections these points are clarified when this notion of operational semantics is formalised.

2.1 Preliminary Definitions

We first give some definitions and notation that will be used throughout this section. The set theoretical notions can be found on any good book in axiomatic set theory, while most of the definitions on inductive systems are borrowed from [Acz77].

Let A and B be sets; *Pow*(A) denotes the set of all subsets of A; *FinPow*(A) denotes the set of all finite subsets of A; $[\mathrm{A} \rightarrow \mathrm{B}]$ denotes the set of all functions from A to B; and $[n]$ denotes the finite subset $\{1, \ldots, n\}$ of the natural numbers.

Let S be a set (of sorts); then $\{\mathrm{A}_s\}_{s \in \mathrm{S}}$ is the family A of sets sorted, or indexed, by S. For A an S-sorted family, $\bigcup \mathrm{A}_s$ is the union of all sets in the family A (and $\bigcap \mathrm{A}_s$ is the intersection), and we often use A to ambiguously denote $\bigcup \mathrm{A}_s$. If $w \in \mathrm{S}^*$ is a string $s1 \ldots sn$ of sorts in S, and A is an S-sorted family of sets then $\mathrm{A}^w = \mathrm{A}_{s1} \times \ldots \times \mathrm{A}_{sn}$.

For this presentation we assume we have an *universe* $\mathcal{U}$ of objects of interest. A *rule* is an object (X, x), where X is a set (contained in $\mathcal{U}$), called the premisses, and x is an object (in $\mathcal{U}$), called the conclusion. An *inductive system* is a set of rules, usually denote by ϕ. If ϕ is a set of rules, and A is some subset of $\mathcal{U}$, we define A to be ϕ-closed if each rule in ϕ whose premisses are in A also has its conclusion in A. More precisely, $\mathrm{A} \subseteq \mathcal{U}$ is ϕ-closed if for each $(\mathrm{X}, \mathrm{x}) \in \phi$, $\mathrm{X} \subseteq \mathrm{A}$ implies $\mathrm{x} \in \mathrm{A}$.

If ϕ is a set of rules, then $\mathrm{I}(\phi)$, the set inductively defined by ϕ, is defined by:

$$\mathrm{I}(\phi) = \bigcap \{\mathrm{A} \subseteq \mathcal{U} \mid \mathrm{A} \text{ is } \phi\text{-closed}\}$$

Let S be a set (of sorts); an S-sorted rule is an object $(\mathrm{X}, \mathrm{x}_s)$, where X is an S-sorted set and x_s is an object of sort s, such that $s \in \mathrm{S}$. All definitions above extend trivially to sets of S-sorted rules. Moreover, if ϕ is an S-sorted set of rules, then $\mathrm{I}(\phi)$ is the S-sorted set inductively defined by ϕ. Rules are the special case of S-sorted rules where S is singleton. Hereafter, we use the sole name rules for both single sorted and many sorted rules.

When a set of rules ϕ is finitary, i.e., X is finite for each rule (X, x) in ϕ, then there exists a notion of finite length proof, defined below:

Definition 2.1 (Finite Length Proof) *Given a finitary set of rules ϕ, a sequence $a_0, \ldots, a_n$ is a finite length ϕ-proof of an object b if:*

1. *$a_n = b$, and*
2. *for all $m \leq n$ there is an $\mathrm{X} \subseteq \{a_i | i < m\}$ such that $(\mathrm{X}, a_m) \in \phi$*

The next proposition relates ϕ-proofs to the set $\mathrm{I}(\phi)$ inductively defined by ϕ. We omit the proof here, which can be found in Aczel's paper [Acz77].

Proposition 2.1 *For a finitary set of rules ϕ:*

$$\mathrm{I}(\phi) = \{a | a \textit{ has a } \phi\textit{-proof}\}$$

This proposition will be central in the definition of a proof search strategy in section 3.

2.2 First Order Signatures and Relation Semantics

In this section a formalisation of *relation semantics* in the framework of first structures and related models is presented. The objective is to achieve a better understanding of the nature of the semantic domains and objects involved in a relation semantics. This formalisation presents relation semantics as a very general and precisely defined mathematical object, which is then specialised in section 2.3 to the notion of operational semantics. The notation and some definitions on many sorted universal algebras are those of [TWW78]. Most of the notation and some definitions on many sorted first order signatures and related models are borrowed from [GB90].

Definition 2.2 (First Order Signature) *An* S*-sorted first order signature Ω, called an Ω-signature, is a triple $\langle \mathrm{S}, \Sigma, \Pi \rangle$, where* S *is a set of sorts, Σ is an $\mathrm{S}^* \times \mathrm{S}$-sorted family of sets (of operator or function symbols), and Π is an S^*-sorted family of sets (of predicate or relation symbols).*

For all Ω-signatures $\langle \mathrm{S}, \Sigma, \Pi \rangle$ we require that: (1) $\bigcap \Pi_w = \emptyset$, i.e., there is no overloading of relation symbols[1]; (2) Σ satisfies the condition that $\Sigma_{w,s} \cap \Sigma_{w',s} = \emptyset$ for all $s \in \mathrm{S}$ and $w, w' \in \mathrm{S}^*$, such that $w \neq w'$, i.e., if there is overloading of operations symbols then the result sort must be different whenever the input sorts are different, for each overloaded symbol in Σ.

To illustrate the presentation a small example of signature is given. In order to simplify the reading of the example, suppose we are working with an Ω-signature $\langle \mathrm{S}, \Sigma, \Pi \rangle$. Whenever a string $w \in \mathrm{S}^*$, $w = s1 \ldots sn$, is the sort of some relation symbol in Π, this sort is written as the sequence $s1, \ldots, sn$. If w is as above and $s \in \mathrm{S}$ is some sort, then if w, s is the sort of an operation symbol in Σ, this sort is written as $s1, \ldots, sn \rightarrow s$.

[1]Although this restriction is not strictly necessary its relaxation makes the notation rather obscure. Therefore, we preferred the more restricted version in this presentation.

Moreover, if w is the empty string of sorts (denoted ε), the sort ε, s is written as simply s.

Consider the signature Bin $= \langle \mathrm{S}, \Sigma, \Pi \rangle$ that could be used in the relation semantics of binary numerals, where:

$$\begin{aligned}
\mathrm{S} &= \{\mathrm{bin}, \mathrm{nat}\} \\
\Sigma &= \{\{\mathit{0}, \mathit{1}\}_{\mathrm{bin}}, \{\mathit{NextOdd}, \mathit{NextEven}\}_{\mathrm{bin}\to\mathrm{bin}}, \{0\}_{\mathrm{nat}}, \{\mathit{Succ}\}_{\mathrm{nat}\to\mathrm{nat}}, \{\times\}_{\mathrm{nat},\mathrm{nat}\to\mathrm{nat}}\} \\
\Pi &= \{\{\Rightarrow\}_{\mathrm{bin},\mathrm{nat}}\}
\end{aligned}$$

Intuitively, the way of assigning meaning to the symbols in the signatures, is by giving it a model. A definition of first order model, or structure, is given below.

Definition 2.3 (Ω-Model) *An Ω-model, for an Ω-signature $\langle \mathrm{S}, \Sigma, \Pi \rangle$, is a triple $\mathcal{M} = \langle |\mathrm{A}|, \alpha, \beta \rangle$, where $|\mathrm{A}|$ is an* S*-sorted family of non-empty sets $\{\mathrm{A}_s\}_{s\in \mathrm{S}}$, where A_s is called the carrier of sort s, α is an $\mathrm{S}^* \times \mathrm{S}$-sorted family of functions $\alpha_{w,s} : \Sigma_{w,s} \to [\mathrm{A}^w \to \mathrm{A}_s]$ assigning a function to each function symbol in Σ, and β is an S^*-sorted family of functions $\beta_w : \Pi_w \to Pow(\mathrm{A}^w)$ assigning a relation to each relation symbol in* Π.

A possible Ω-model for the Bin signature is given by ModelBin $= \langle \{\mathrm{A}_{\mathrm{bin}}, \mathrm{A}_{\mathrm{nat}}\}, \alpha, \beta \rangle$, where:

$$\begin{aligned}
\mathrm{A}_{\mathrm{bin}} &= \{\mathit{0}, \mathit{1}\} \\
\mathrm{A}_{\mathrm{nat}} &= \{0\}
\end{aligned}$$

The functions in the family α are the usual operations and constants on the natural numbers, and $\alpha_{\mathrm{bin}}(\mathit{0})$ is the constant function $\mathit{0}$ (similarly for $\alpha_{\mathrm{bin}}(\mathit{1})$), and $\alpha_{\mathrm{bin}\to\mathrm{bin}}(\mathit{NextOdd})$ and $\alpha_{\mathrm{bin}\to\mathrm{bin}}(\mathit{NextEven})$ are the obvious functions to construct the next odd and next even binary numerals respectively. The relation $\beta_{\mathrm{bin},\mathrm{nat}}(\Rightarrow)$ is the set of pairs like $(\mathit{0}, 0)$, or $(\mathit{NextEven}(\mathit{1}), \mathit{Succ}(\mathit{Succ}(0)))$, which defines the decimal base values of the binary numerals in terms of the natural numbers.

For $\pi \in \Pi_w$ with $w = s1, \ldots, sn$ and $a_i \in \mathrm{A}_{si}$, we say that "$\pi(a_1, \ldots, a_n)$ holds" in $\langle |\mathrm{A}|, \alpha, \beta \rangle$ iff $(a_1, \ldots, a_n) \in \beta(\pi)$. Usually an object $(a_1, \ldots, a_n)$ is called an n-tuple in a relation $\beta(\pi)$. In what follows, whenever π is a binary relation we write $a_1 \pi a_2$ for $\pi(a_1, a_2)$. In order to formalise a notion of relation semantics we first define how to write relation expressions, i.e., atomic formulae over an Ω-signature, with variables in an S-sorted set X. This is done in the usual way, by first defining the set of terms over an Ω-signature.

Let $\Omega = \langle \mathrm{S}, \Sigma, \Pi \rangle$ be a first order signature, and X be an S-sorted family $\{\mathrm{X}_s\}_{s\in\mathrm{S}}$ of sets of variables such that $\bigcap \mathrm{X}_s = \emptyset$, and X is disjoint from both Σ and Π. The S-sorted family $\mathrm{TERM}_{\mathrm{X}}(\Omega)$ of (Ω, X)-terms is defined to be the carriers of the the free Σ-algebra $\mathrm{T}_\Sigma(\mathrm{X})$, with generators X. In what follows we use x, possibly with subscripts and superscripts, as a generic element of X, and x_s when we want to emphasize its sort.

The S-sorted function $\mathcal{V}$, of variables in a term, i.e., $\mathcal{V}_s : \mathrm{TERM}_{\mathrm{X}}(\Omega) \to Pow(\mathrm{X}_s)$, for $s \in \mathrm{S}$, is defined inductively as follows:

1. $\mathcal{V}_s(x) = \{x\}$, for $x \in \mathrm{X}_s$, and $\{\ \}$ otherwise.
2. $\mathcal{V}_s(\sigma(t_1, \ldots, t_n)) = \bigcup_{i=1}^{n} \mathcal{V}_s(t_i)$, for $\sigma(t_1, \ldots, t_n) \in \mathrm{TERM}_{\mathrm{X}}(\Omega)$

For the Ω-signature Bin, and for a set X of variables sorted by S of bin, such that $\mathrm{X}_{\mathrm{bin}} = \{b_0, b_1, \ldots\}$, and $\mathrm{X}_{\mathrm{nat}} = \{n_0, n_1, \ldots\}$, a term in $\mathrm{TERM}_{\mathrm{X}}(\mathrm{Bin})$ is $\times(n_1, \mathit{Succ}(0))$, with the set of variables given by $\mathcal{V}_{\mathrm{nat}}(n_1 \times \mathit{Succ}(0)) = \{n_1\}$.

If $t \in \mathrm{TERM}_{\mathrm{X}}(\Omega)$, then a *substitution* θ of terms for variables in t is an S-sorted mapping such that each θ_s has type $\theta_s : \mathcal{V}_s(t) \rightarrow \mathrm{TERM}_{\mathrm{X}}(\Omega)_s$. It is often necessary to write a substitution explicitly, in this case we use [ ] to denote the empty substitution, and $[x_{s1}/t_{s1}, \ldots, x_{sn}/t_{sn}]$ to denote the substitution of the terms $t_{s1}, \ldots, t_{sn}$ for the variables $x_{s1}, \ldots, x_{sn}$, respectively. We write $\theta(t)$ to denote the application of a substitution on a term t.

An *α-conversion* is a one-to-one substitution $\theta : \mathcal{V}(t) \rightarrow \mathrm{X}$, and we write $t_1 \equiv_\alpha t_2$ when there exists an α-conversion for t_1 such that $\theta(t_1) = t_2$. A *renaming* of variables in a formula f is a one-to-one substitution $\rho : \mathcal{V}(f) \rightarrow \mathrm{X} \setminus \mathcal{V}(f)$, which has the effect of changing the names of each variable in f to other variables not appearing in f. Two terms t_1 and t_2 *unify* iff there exists a substitution θ such that $\theta(t_1) = \theta(t_2)$, and the two terms are *different*, written $t_1 \neq t_2$, iff they do not unify. The *composition* of two substitutions θ_1 and θ_2, is denoted by $\theta_1\theta_2$.

Then, for a given Ω-signatures $\Omega = \langle \mathrm{S}, \Sigma, \Pi \rangle$, the set of well-formed (Ω,X)-formulae is defined as the one-sorted set of expressions $\mathrm{WFF}_{\mathrm{X}}(\Omega)$ [2] given by:

$$\mathrm{WFF}_{\mathrm{X}}(\Omega) = \{\pi(t_1, \ldots, t_n) | \pi \in \Pi_w, w = s1, \ldots, sn, t_i \in \mathrm{TERM}_{\mathrm{X}}(\Omega)_{si}\}$$

Hereafter, the term formula is used meaning a formula in $\mathrm{WFF}_{\mathrm{X}}(\Omega)$. The definition of the S-sorted function $\mathcal{V}$ such that $\mathcal{V}_s : \mathrm{WFF}_{\mathrm{X}}(\Omega) \rightarrow \mathit{Pow}(\mathrm{X}_s)$, is extended to formulae in the following way: $\mathcal{V}_s(\pi(t_1, \ldots, t_n)) = \bigcup_{i=1}^{n} \mathcal{V}_s(t_i)$. The definitions of substitution, α-conversion, and unification are also extended to formulae in the trivial way.

An example of formula in $\mathrm{WFF}_{\mathrm{X}}(\mathrm{Bin})$ is $b_3 \Rightarrow \mathit{Succ}(n_0)$, for which the set of variables is given by $\mathcal{V}_{\mathrm{bin}}(b_3 \Rightarrow \mathit{Succ}(n_0)) = \{b_3\}$, and $\mathcal{V}_{\mathrm{nat}}(b_3 \Rightarrow \mathit{Succ}(n_0)) = \{n_0\}$.

Now, a satisfaction relation of formulae in $\mathrm{WFF}_{\mathrm{X}}(\Omega)$ by Ω-models is defined in the following way:

Definition 2.4 (Satisfaction Relation) *Let f be a formula $\pi(t_1, \ldots, t_n) \in \mathit{WFF}_{\mathrm{X}}(\Omega)$. An Ω-model $\mathcal{M} = \langle |\mathrm{A}|, \alpha, \beta \rangle$ satisfies f, or f holds in $\mathcal{M}$, written $\mathcal{M} \models f$, iff for all (S-sorted) valuation functions $v : \mathrm{X} \rightarrow |\mathrm{A}|$, $(v^{\#}(t_1), \ldots, v^{\#}(t_n)) \in \beta(\pi)$, where $v^{\#}(t)$ denotes the evaluation of the term t in the Σ-algebra $\langle |A|, \alpha \rangle$ part of $\mathcal{M}$ using initiality, with the values of the variables given by the valuation function v.*

[2]The set $\mathrm{WFF}_{\mathrm{X}}(\Omega)$ is not the usual definition of formulae over a first order signature, which may include formulae with quantifiers and logic connectives which are not necessary for our purposes.

In the example, ModelBin $\models$ *NextEven(1)* $\Rightarrow$ *Succ(Succ*(0)), but if we use $\not\models$ meaning "does not satisfy", ModelBin $\not\models$ *1* $\Rightarrow$ *Succ(Succ*(0)).

An Ω-sentence is defined as a ground (Ω,X)-formula, i.e., an (Ω,X)-formula f with $\mathcal{V}(f) = \emptyset$, and Sen($\Omega$) is used to denote the set of all Ω-sentences. Finally, relational semantics is defined in the following way:

Definition 2.5 (Relational Semantics) *Let* $\Omega = \langle \mathrm{S}, \Sigma, \Pi \rangle$ *be a first order signature, and* $\mathcal{S} \subseteq Sen(\Omega)$ *be a nonempty set of sentences, then a relational semantics for* $\mathcal{S}$ *is an* Ω*-model* $\mathcal{M}$ *if* $\mathcal{M} \models S$ *iff* $S \in \mathcal{S}$.

In the example, ModelBin is a relation semantics for exactly the set of pairs (b, n) in which n is the decimal base value of the binary numeral b.

2.3 Inductive Definitions and Operational Semantics

In this section *operational semantics* is defined as a particular case of relation semantics where the relations can be inductively defined. The justification for giving the name operational to inductive definitions can be informally explained as follows: (1) for the particular definition of *inductive definitions* given below, a notion of ϕ-proofs similar to the definition 2.1 exists, and a *proof search strategy* can be defined that uses the inductive rules to find ϕ-proofs of objects; (2) by examining the way proofs are constructed by the search strategy, it is possible to identify the individual steps of a particular proof, and then define a general notion of *proof steps*; (3) the notion of *directed relations* of section 4.1 allows relations to be intuitively interpreted as taking some inputs, performing some operations on them, and delivering some outputs. Then, whenever a semantics of some operation is given by an inductively defined directed relation, the proof steps will describe exactly how such operation acts on its inputs to produce its outputs, and this is the operational notion we want to characterise.

We start by describing how to define sets of relations inductively. Let $\mathcal{M} = \langle |\mathrm{A}|, \alpha, \beta \rangle$ be an Ω-model (for an Ω-signature $\Omega = \langle \mathrm{S}, \Sigma, \Pi \rangle$), in which $|\mathrm{A}|$ and α are already defined, and β is to be inductively defined. We first introduce a notion of *inductive definitions* over an Ω-signature with variables in a set X, and then show how to go from an inductive definition to the inductive system it defines. Finally, if $\phi^{\#}$ denotes the inductive system obtained from an inductive definition over Ω, then $\mathrm{I}(\phi^{\#})$ is the set of relations necessary in order to define β. This notion is formalised below.

An inductive (Ω,X)-rule is an object (Pre, c), where Pre is a finite set of (Ω,X)-formulae, called the *premisses*, and c is an (Ω,X)-formula, called the *conclusion*. All variables in the formulae in a rule are implicitly bound to the rule. An *inductive definition* is a set of inductive (Ω, X)-rules, and ϕ_{X} is used to denote generic inductive definitions with variables in X. When the Ω-signature and the set of variables X can be inferred from the context, the term *inductive rules* is used meaning inductive (Ω, X)-rules, and *rules* to denote the rules of an inductive system, as in section 2.1.

An example of how to inductively define $\Rightarrow$ of signature Bin would be the following set of inductive rules, in which we write the rules in a more readable form of natural deduction style rules:

$$\frac{}{\mathit{0} \Rightarrow 0}$$

$$\frac{}{\mathit{1} \Rightarrow \mathit{Succ}(0)}$$

$$\frac{b_0 \Rightarrow n_0}{\mathit{NextEven}(b_0) \Rightarrow \times(\mathit{Succ}(\mathit{Succ}(0)), n_0)}$$

$$\frac{b_0 \Rightarrow n_0}{\mathit{NextOdd}(b_0) \Rightarrow \mathit{Succ}(\times(\mathit{Succ}(\mathit{Succ}(0)), n_0))}$$

We show how to go from an inductive definition to the inductive system it specifies, by first showing how to obtain an n-tuple from a formula. Suppose f is a formula $\pi(t_1, \ldots, t_n)$, for some $\pi \in \Pi_w$. Then for a (S-sorted) valuation function $v : \mathrm{X} \longrightarrow |\mathrm{A}|$, the n-tuple derived from f is $(v^{\#}(t_1), \ldots, v^{\#}(t_n))$, where $v^{\#}(t)$ denotes the evaluation of a term t in the Σ-algebra $\langle |\mathrm{A}|, \alpha\rangle$ part of $\mathcal{M}$ using initiality, with the values of the variables given by the valuation v.

As an example, suppose f is a formula $b_1 \Rightarrow \times(\mathit{Succ}(\mathit{Succ}(0)), n_0)$. For an assignment $[b_1/\mathit{NextEven}(\mathit{1}), n_0/\mathit{Succ}(0)]$ the tuple derived from f is $(\mathit{NextEven}(\mathit{1}), \mathit{Succ}(\mathit{Succ}(0)))$.

This example emphasizes what we want to achieve with the construction of tuples from formulae. Intuitively, for objects like the natural numbers, we can define the operations like $\times$ straight in the Σ-algebra part of the semantics. For more complex objects, like the binary numerals, we give their meaning in terms of the natural numbers, by defining the relation $\Rightarrow$. Therefore, evaluation by initiality on these objects is just the identity function on $\mathrm{TERM}_{\mathrm{X}}(\Omega)$.

Once it has been defined how tuples are obtained from formulae, we show how rules can be derived from inductive rules. Let $(\mathrm{Pre}, \mathrm{c}) \in \phi_{\mathrm{X}}$ be an inductive rule; for a given (S-sorted) valuation function $v : \mathrm{X} \longrightarrow |\mathrm{A}|$, the Π-sorted rule $(\mathrm{X}, \mathrm{x}_\pi)$ derived from $(\mathrm{Pre}, \mathrm{c})$ is defined as follows:

- X is a Π-sorted family of sets $\{\mathrm{X}_\pi\}_{\pi \in \Pi_w}$, such that:

$$\mathrm{X}_\pi = \{(v^{\#}(t_1), \ldots, v^{\#}(t_n)) | \pi(t_1, \ldots, t_n) \in \mathrm{Pre}\}$$

- x_π is a Π-sorted object:

$$\mathrm{x}_\pi = (v^{\#}(t_1), \ldots, v^{\#}(t_n)), \qquad \text{for } \mathrm{c} = \pi(t_1, \ldots, t_n)$$

where the tuples are obtained by the construction given above.

Then the inductive system specified by ϕ_{X} is the union of the sets of rules derived from each rule in ϕ_{X} for all possible valuations v, and is denoted by $\phi^{\#}$. Therefore, $\phi^{\#}$ is the inductive system uniquely determined by an inductive definition ϕ_X. In what follows we use $I(\phi_X)$ for $I(\phi^\#)$, and a ϕ_X-proof for a formula f is just a $\phi^\#$-proof for some evaluation $v^\#(f)$ of f, for a given valuation function v.

The concept of an inductive Ω-model is defined as a first order Ω-model in which the family of functions β is replaced by an inductive definition ϕ_X of a family of relations.

Definition 2.6 (Inductive Ω-Models) *An inductive Ω-model $\mathcal{O}$ for a signature Ω is a triple $\langle|A|, \alpha, \phi_X\rangle$ where $|A|$ and α are as in the first order Ω-model, and ϕ_X is a set of $(\Omega$,X$)$-rules.*

Then the satisfaction relation of definition 2.4 can be generalised to inductive Ω-models. Consider that an inductive Ω-model $\mathcal{O} = \langle|A|, \alpha, \phi_X\rangle$ is a way of defining a first order Ω-model $\langle|A|, \alpha, \beta\rangle$ such that β is the S^*-sorted family of functions $\beta_w : \Pi_w \rightarrow I(\phi_X)$, given by $\beta_w(\pi) = I(\phi_X)_\pi$. Therefore, if f is a formula $\pi(t_1, \ldots, t_n) \in \mathrm{WFF}_X(\Omega)$, then $\mathcal{O} \models f$ iff for all valuation functions v, $v^\#(f) \in I(\phi_X)_\pi$.

And a definition of operational semantics can be formally presented as follows:

Definition 2.7 (Operational Semantics) *Let $\Omega = \langle S, \Sigma, \Pi\rangle$ be a first order signature, then an operational semantics for a non-empty set of sentences $\mathcal{S} \in Sen(\Omega)$ is an inductive Ω-model $\mathcal{O} = \langle|A|, \alpha, \phi_X\rangle$ if $\mathcal{O} \models S$ iff $S \in \mathcal{S}$.*

From the definitions given above an Ω-model $\mathcal{O} = \langle|A|, \alpha, \phi_X\rangle$ is an operational semantics for exactly the Π-sorted set of relations $I(\phi_X)$. Therefore, hereafter $\mathcal{O}$ will be called an operational semantics without specifying for which set of sentences, since this can be easily deduced from $\mathcal{O}$.

Finally, an *operational semantics specification* is a pair $\mathcal{D} = \langle\Omega, \mathcal{O}\rangle$, where Ω is a first order signature and $\mathcal{O}$ is an Ω-model such that $\mathcal{O}$ is also an operational semantics. Since the set of variables X can often be inferred from the context, in what follows ϕ will be used for ϕ_X.

2.3.1 The Treatment of Side Conditions

Before the characterisation of operational semantics is finished, it is necessary to show how the concept of side conditions fits in the framework. In most of the work on operational semantics, side conditions are treated in one of two ways: (1) by considering them as normal premisses, for which a pre-defined inductive definition exists; (2) by treating them as special objects, whose meaning is somehow defined in an ad hoc way, e.g. by calling Prolog predicates, as in TYPOL.

Both treatment have problems which make the side conditions in semantic specifications objects that are difficult to formally reason about. In (1) there is no nice way of inductively define some side conditions, basically because some objects (e.g. real numbers), and some relations (e.g. inequality) cannot be inductively defined. The problem in (2) is that the meaning of side conditions somehow "depends on the implementation", and therefore any formal reasoning about them become difficult, if possible.

The approach proposed here is formalised in three stages: (1) for each operational semantics specification $\langle \Omega, \mathcal{O} \rangle$, a new sort **bool** is included in S of Ω, together with a new carrier $\{true, false\}_{\mathbf{bool}}$ in $|A|$ of $\mathcal{O}$. Then, (2) the predicates that are to appear in side conditions are defined in $\Sigma_{w,\mathbf{bool}}$, and their implementations are defined as some $\alpha_{w,\mathbf{bool}}$ functions, for some w. Finally, (3) a new relation name, say π, is included in $\Pi_{\mathbf{bool}}$, and the following rule is added to ϕ of $\mathcal{O}$:

$$\frac{}{\pi_{\mathbf{bool}}(true)}$$

This has exactly the desired effect for the side conditions, since each term in $T_\Sigma(X)_{w,\mathbf{bool}}$ will be evaluated to either *true* or *false* when initiality is applied, and only the ones resulting in *true* will be given a proof.

The reason why this formalisation was chosen is that: first, it allows side conditions to be treated as formulae, and therefore appearing as normal premisses in the rules. Second, the precise meaning of side conditions is given as the evaluation of a term in the Σ-algebra part of the semantics. Finally, it does not preclude the use of pre-defined inductive definitions of predicates as discussed above.

In this way the only special treatment for side conditions are the points discussed above. No special treatment is necessary in the definition of a proof search strategy or an evaluation model, since the terms in the inductive rules are already being evaluated using initiality. This discussion finishes the definition of the characterisation of operational semantics. It remains to show how this notion can be used to automatically check whether an object is defined by some semantics. This is done in the definition of a proof search strategy of next section.

3 A Proof Search Strategy

Recall that proposition 2.1 states a relationship between a particular definition of proofs in an inductive system and its associated inductively defined set. In the context of inductive definitions presented in section 2.3 that proposition can be restated as: if ϕ is an inductive definition, and f is a formula $\pi(t_1, \ldots, t_2)$, then for a given valuation v, $v^\#(f) \in I(\phi)_\pi$ iff $v^\#(f)$ has a ϕ-proof.

With a given operational semantics specification $\langle \Omega, \mathcal{O} \rangle$, there are many useful questions one can ask. For instance, whether a sentence $s \in \mathrm{Sen}(\Omega)$ holds in $\mathcal{O}$, i.e., is it the case that $\mathcal{O} \models s$? Another, more interesting, question could be: for a given formulae f, does there exist a valuation v, such that $\mathcal{O} \models v^\#(f)$? From the discussion of the last paragraph, this question is equivalent to ask whether $v^\#(f)$ has a ϕ-proof.

Thus, to check whether an operational semantics satisfies some formula f means to use the inductive definition ϕ to find a ϕ-proof for $v^\#(f)$, for some valuation v. In the process of searching for this proof, variables in f get instantiated appropriately such that if a ϕ-proof exits the final instantiation of f belongs to $I(\phi)_\pi$. In other words, the search for a ϕ-proof for f also finds a valuation v such that $v^\#(f) \in I(\phi)_\pi$.

Back to the example of section 2, the sentence $\mathit{NextEven}(1) \Rightarrow \mathit{Succ}(\mathit{Succ}(0))$ can be given a ϕ-proof in the rules of section 2.3. On the other hand, the search for a ϕ-proof for the formula $\mathit{NextEven}(1) \Rightarrow n$, should find a valuation $v = [n/\mathit{Succ}(\mathit{Succ}(0))]$ such that $v^{\#}(\mathit{NextEven}(1) \Rightarrow n)$ has a ϕ-proof.

In this way the operational semantics can be used not only to check whether a sentence is satisfied by some semantics specification, but also to query about the values of certain variables in a formula (like n in the above example). Usually these variables are in the place of results of programs, and the proof becomes a way of evaluating the programs according to the operational semantics.

In what follows a *proof search strategy* for inductive definitions is presented, which can be proved sound and complete according to the definition of ϕ-proofs. Let $\mathcal{D} = \langle \Omega, \mathcal{O} \rangle$ be an operational semantics specification, such that $\Omega = \langle \mathrm{S}, \Sigma, \Pi \rangle$, and $\mathcal{O} = \langle |\mathrm{A}|, \alpha, \phi \rangle$. A closure is an object $(\theta : \Phi)$, where θ is a syntactical representation of an S-sorted family of substitutions, and Φ is a finite set of (Ω, X)-formulae.

The basic idea is to present for each set of inductive rules ϕ, a non-deterministic terminal transition system $\mathrm{PSS}_\phi = \langle \Rightarrow_\phi, S, T \rangle$, where S is a set of closures, also called states, $T \subset S$ is a set of closures of the form $(\theta : \emptyset)$, and $\Rightarrow_\phi \subseteq S \times S$ is a transition relation defined by the rewriting rule below:

$$(\theta : \Phi) \Rightarrow_\phi (\mathit{Ext}(\theta\theta') : (\Phi \setminus \{f\}) \cup \rho(\mathrm{Pre}))$$

where: $f \in \Phi$, $(\mathrm{Pre}, \mathrm{c}) \in \phi$, θ' is the most general unifier of $\theta(\rho(c))$ and $\theta(f)$. ρ is a renaming of variables in the formulae to new variables not appearing in the left-side of the arrow[3]. The operation *Ext* applied to the substitution $\theta\theta'$ has the effect of extending the substitution by the following construction: for all variables x in the domain of $\theta\theta'$, if $\theta\theta'(x)$ is a ground term t (i.e., a term without variables), then obtain the evaluation of t by using initiality in the Σ-algebra part of the semantics. Finally, extend $\theta\theta'$ with the binding $[x/\theta\theta'^{\#}(t)]$, where $\theta\theta'^{\#}(t)$ denotes the evaluation of t.

Then, a non-deterministic acceptor for PSS_ϕ can be used to check for proofs of formulae. The correctness of PSS_ϕ can be stated as follows: for all formulae f, $([\,] : \{f\}) \Rightarrow_\phi^* (\theta : \emptyset)$ iff $\theta(f)$ has a $\phi - \mathit{proof}$, where $\Rightarrow_\Phi^*$ denotes the transitive closure of $\Rightarrow_\phi$. The proof, although trivial, will not be given here due to shortage of space.

With the proof search strategy above the underlying theory of the framework for operational semantics is actually completed. What remains to be done is to show how the proof search strategy can be used in a practical tool for the execution of semantics specifications. This means that an implementation for the PSS_ϕ must be given. In particular we are interested in an efficient implementation in the sense that no backtracking strategy is used.

The design of an efficient implementation for PSS_ϕ sums up to point out where non-determinis-tic choices occur in PSS_ϕ, and then give these choices a deterministic implemen-

[3]The renaming ρ is technically necessary to assure that variable names do not get mixed up when either the same variable name appears in more than one rule, or there is a recursive use of some rule during a proof.

tation which does not involve backtracking. What then becomes the problem is that it is not always possible to achieve this efficient implementation for all specifications that can be expressed in the theory. Therefore, we take the approach of first defining a semantics specification meta-language, in which only a subset of the theory can be expressed. An evaluation model for the meta-language is then defined, which can be given an efficient implementation.

4 A Semantic Specification Meta-Language

In this section a *specification meta-language* is designed, in which only a class of deterministic semantics specification can be expressed. This design is done in two steps: (1) a definition of a notion of well formed rules; (2) the characterisation of the class of deterministic semantics specifications. Both steps are necessary to avoid two sources of non-determinism in the search strategy, which are pointed out below.

Considering (1), the problem is how in a state $(\theta : \Phi)$ the next formula to be proved is chosen. Since Φ is an unordered set, we have a choice of picking up any element of it to be the next formula to be proved. However, some formulae in Φ may have *non-complete* terms, i.e. terms whose some of the variables are non-instantiated. It can be problematic to chose such formulae to continue a proof, because for some possible proof strategies backtracking may be necessary to recover from a failure state in some proofs.

However, the problem of picking up the next formulae in a state can be solved without the need for backtracking, if Φ can be ordered such that its smallest element is always the correct next formula to be chosen at any state. This solution involves to know what are the possible correct orders in which the premisses of rules must be tried. If we know such ordering on the premisses of each rule in a semantic specification, then Φ can be treated as a sequence of formulae. Moreover, the operations for insertion and deletion of formulae in Φ can be defined consistently with the ordering of the premisses (this is further discussed in section 5). Therefore, an evaluation model that uses sequences of formulae with insertion and deletion operations as discussed above would be deterministic and work without the need for backtracking. The definition of well formed rules has the objective of allowing a natural definition of an ordering relation on sets of premisses of a rule.

As far as (2) is concerned, we want to treat another source of non-determinism in PSS_ϕ which is related to the problem of choice of rules. By restricting the semantics specifications to be deterministic we want to exclude a class of inductive rules, which one example of is the rules for conditional expressions, given below:

$$\mathcal{R} : \frac{e_1 \Rightarrow \textbf{true} \qquad e_2 \Rightarrow r}{\textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3 \Rightarrow r}$$

$$\mathcal{R}' : \frac{e_1 \Rightarrow \textbf{false} \qquad e_3 \Rightarrow r}{\textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3 \Rightarrow r}$$

Whenever rules like $\mathcal{R}$ and $\mathcal{R}'$ are allowed, it is not possible to choose which one to use to follow a proof until some of the premisses have been tried. This leads to the necessity of some kind of backtracking strategy, which we want to avoid for efficiency reasons as already discussed above.

4.1 Well Formed Rules and Definitions

In this section a notion of well formed rules is presented. The defined conditions are imposed on each rule in an inductive definition.

Let $\mathcal{D} = \langle \Omega, \mathcal{O} \rangle$ be an operational semantics specification with $\Omega = \langle \mathrm{S}, \Sigma, \Pi \rangle$ and $\mathcal{O} = \langle |\mathrm{A}|, \alpha, \phi \rangle$. A *direction* in a relation symbol $\pi \subseteq \Pi_w$, for $w = s_1, \ldots, s_n$ is a partition $\mathrm{D} = \{\mathrm{P}, \mathrm{R}\}$ on $[n]$. For this presentation it is assumed that each Ω-signature is given an unique Π-sorted set of directions $\{\mathrm{D}_\pi\}_{\pi \in \Pi}$, one direction for each relation symbol in Π. A relation symbol with a direction is called a *directed relation.* The definition of directed relation presented here is an extension to the definition given in [Chi89] for the case of binary relations.

If π is a directed relation with $\mathrm{D}_\pi = \{\mathrm{P}_\pi, \mathrm{R}_\pi\}$, a formula $\pi(t_1, \ldots, t_n) \in \mathrm{WFF}_\mathrm{X}(\Omega)$ is written $\pi(t_{\mathrm{P}_\pi}, t_{\mathrm{R}_\pi})$, or even $\pi(t_\mathrm{P}, t_\mathrm{R})$ when the subscript π can be determined from the form of the formula. Notice that this is not an extension to $\mathrm{WFF}_\mathrm{X}(\Omega)$ but only a "syntactic sugar" for writing formulae in a directed relation. Therefore, when we write $\pi(t_\mathrm{P}, t_\mathrm{R}) \in \mathrm{WFF}_\mathrm{X}(\Omega)$ we mean $\pi(t_1, \ldots, t_n) \in \mathrm{WFF}_\mathrm{X}(\Omega)$, where the formula $\pi(t_1, \ldots, t_n)$ is the "desugared" form of $\pi(t_\mathrm{P}, t_\mathrm{R})$. The terms in t_P are called *phrases* and those in t_R are the *results* of a directed relation π.

For the relation $\Rightarrow_{\mathrm{bin,nat}}$ of section 2, a possible direction can be such that for a tuple $b_3 \Rightarrow \mathit{Succ}(0)$, $t_{\mathrm{P}_\Rightarrow} = b_3$ and $t_{\mathrm{R}_\Rightarrow} = \mathit{Succ}(0)$.

The function $\mathcal{V}$ and the definitions of substitution, α-conversion, and unification extend to the object t_P (also to t_R) by considering their effect on each term t in t_P (also in t_R).

We follow by defining a dependency relation $\prec$ in $\mathrm{WFF}_\mathrm{X}(\Omega)$. The intuition behind this relation is that if a formula f' depends on another formula f, then some terms in the results of f have variables that also appear in some terms in the phrases of f'. This indicates that f is "producing" the instantiation of some variables which will be "consumed" by f'. Therefore, in a proof f must be tried first.

The relation $\prec$ is defined as follows: for $f = \pi(t_{\mathrm{P}_\pi}, t_{\mathrm{R}_\pi})$ and $f' = \pi'(t'_{\mathrm{P}_{\pi'}}, t'_{\mathrm{R}_{\pi'}})$, $f \prec f'$ if $\mathcal{V}(t_{\mathrm{R}_\pi}) \cap \mathcal{V}(t'_{\mathrm{P}_{\pi'}}) \neq \emptyset$. Then let a relation $\sqsubseteq$ to be the transitive and reflexive closure of $\prec$. The relation $\sqsubseteq$ is used in the definition of well formed rules given below.

An inductive rule (Pre, c), where Pre $\in$ *FinPow*($\mathrm{WFF}_\mathrm{X}(\Omega)$) and c $\in \mathrm{WFF}_\mathrm{X}(\Omega)$, is defined to be *well formed* iff for c $= \pi(t_\mathrm{P}, t_\mathrm{R})$, both conditions below hold:

1. $\sqsubseteq$ is antisymmetric on the elements of Pre, i.e. $\sqsubseteq$ is a partial order on the set of premisses.

2. For each term t in t_{R}, either $\mathcal{V}(t) = \emptyset$, or for each variable $v \in \mathcal{V}(t)$, either $v \in \mathcal{V}(t_{\mathrm{P}})$, or there exists a formula $f \in \mathrm{Pre}$, such that $v \in \mathcal{V}(f)$.

Hereafter,inductive rules are written in the more common form of natural deduction style rules, as below:

$$\frac{\mathrm{Pre} \in \mathit{FinPow}(\mathrm{WFF}_{\mathrm{X}}(\Omega))}{\mathrm{c} \in \mathrm{WFF}_{\mathrm{X}}(\Omega)}$$

For a given set of premisses Pre, and $f \in \mathrm{Pre}$, the notation $s(f)$ denotes the formulae in Pre that are strictly smaller than the formula f according to $\sqsubseteq$, called the *strict initial segment* determined by f in Pre. The definitions of the function $\mathcal{V}$, substitution, α-conversion, and unification generalises to the strict initial segment of a formula in a set of formulae by considering its effect in each formula in the segment.

The partial ordering $\sqsubseteq$ is the one in which a proof search strategy should try the premisses in order to guarantee that instantiations of variables in the premisses are done in an appropriate sequence. If *produced variables* are defined as those variables that appear in some result term of some formula in Pre, the argument for using this particular partial order is that whenever any premiss is about to be tried, all its produced variables are instantiated. Moreover, no further instantiation of variables will invalidate the previous instantiations because $\sqsubseteq$ is a partial order, and therefore Pre is cycle free.

Although it is possible to define a partial order on sets of premisses as discussed above, an actual implementation of an evaluation model would have to chose a particular total order in which to carry on the proofs. Nevertheless, this total order only need to be consistent with $\sqsubseteq$, which gives a great deal of flexibility in the definition of the evaluation model. However, in this presentation this flexibility is not exploited any further, which is a topic for future work.

Hereafter, we assume that all sets of premisses are totally ordered by an ordering relation that is consistent with $\sqsubseteq$. Whenever, a rule had its premisses ordered by a total order as discussed above, it is called an $\sqsubseteq$-ordered rule, and is written as follows:

$$\frac{\pi_1(t_{\mathrm{P}1}, t_{\mathrm{R}1}) \ldots \pi_n(t_{\mathrm{P}n}, t_{\mathrm{R}n})}{\pi(t_{\mathrm{P}}, t_{\mathrm{R}})}$$

where the indexing of the formulae indicates the total ordering.

4.2 Deterministic Semantic Specifications

A set of well formed inductive rules ϕ is defined to be *deterministic* if there is no pair of rules (Pre, c) and (Pre$'$, c$'$) in ϕ, such that whenever $\mathrm{c} = \pi(t_{\mathrm{P}}, t_{\mathrm{R}})$ and $\mathrm{c}' = \pi(t'_{\mathrm{P}}, t'_{\mathrm{R}})$, t_{P} and t'_{P} unify.

Well formed deterministic sets of inductive rules form a class of semantics specification, which is a subset of all possible specification in the theory defined earlier. In the next section an efficient evaluation model is defined for this class of specifications. In the next few paragraphs yet another subset of semantic specifications, called *determinate specifications*, is defined and compared with the deterministic ones The definition of determinate specification is an extension to the definition given in [Ber90], where a more restricted form of rules is used.

A set of inductive rules ϕ is *determinate* iff for each pair of rules (Pre, c) and (Pre$'$, c$'$) in ϕ, whenever the conclusions are such that $\mathrm{c} = \pi(t_{\mathrm{P}}, t_{\mathrm{R}})$ and $\mathrm{c}' = \pi(t'_{\mathrm{P}}, t'_{\mathrm{R}})$, either:

- $t_{\mathrm{P}} \neq t'_{\mathrm{P}}$, i.e. the phrases of both conclusions do not unify, or
- $t_{\mathrm{P}} \equiv_\alpha t'_{\mathrm{P}}$, and there exits $\pi_i(t_{\mathrm{P}i}, t_{\mathrm{R}i}) \in \mathrm{Pre}$ and $\pi_i(t'_{\mathrm{P}i}, t'_{\mathrm{R}i}) \in \mathrm{Pre}'$, such that $t_{\mathrm{P}i} \equiv_\alpha t'_{\mathrm{P}i}$, $t_{\mathrm{R}i} \neq t'_{\mathrm{R}i}$, and $s(\pi_i(t_{\mathrm{P}i}, t_{\mathrm{R}i})) \equiv_\alpha s(\pi_i(t'_{\mathrm{P}i}, t'_{\mathrm{R}i}))$.

where the pair of premisses $\pi_i(t_{\mathrm{P}i}, t_{\mathrm{R}i})$ in Pre, and $\pi_i(t'_{\mathrm{P}i}, t'_{\mathrm{R}i})$ in Pre$'$, is called a *distinction pair.* When n rules clash as defined above, the concept of distinction pair generalises to *distinction n-tuple* in the trivial way.

Determinate sets of inductive rules only allow deterministic programming languages to have their operational semantics specified. This can be seen informally by considering that whenever two rules have conclusions which unify with a given formula, at most one will be used in the proof of the formula, for at most one of the components of the distinction pair can be proved.

The subset of determinate specifications is not more powerful than the subset of deterministic specifications, in the sense of the class of semantics that can be expressed in them. However, determinate specifications allow more flexibility in writing the semantics, leading to more readable and concise sets of rules. Moreover, there exists an algorithm that converts determinate specifications into deterministic ones. This is desirable since semantic specifications can be written in the more flexible form of determinate specifications, converted into deterministic equivalent specifications, and the resulting specifications can be evaluated in the *evaluation model* define in section 5. This issue is further discussed in section 6.

5 An Evaluation Model

The objective is to define an *evaluation model* for the class of deterministic semantic specifications defined in the previous section. This is done by defining another terminal transition system which uses the fact that premisses in the inductive rules are ordered. This allows the new transition system to treat the sets Φ in a state $(\theta : \Phi)$ of PSS_ϕ as sequences of formulae. Therefore, we must define operations to manipulate sequences, like insertion and deletion of elements.

At this stage we are faced with various options on how to define the operations to manipulate sequences of formulae. The only requirement for these operations is that the insertion and deletion on sequences must be somehow consistent with the partial order $\sqsubseteq$. Intuitively, this means that the insertion of two formulae f and f', such that $f \sqsubseteq f'$ but $f' \not\sqsubseteq f$, in any sequence must be done such that f is always deleted from the sequence before f'. Each of the possible options for the definition of the operations somehow characterises how the evaluation model will work. For instance, treating sequences as queues results in a search strategy that works breadth-first, whereas treating sequences as stacks would make the strategy to work depth-first.

It is not obvious which particular implementation to chose for an evaluation model, mainly because these choices somehow depend on how and why the evaluation model will be used in practice. In the process of defining our evaluation model various factors were taken into account, but a deep discussion on those factors is out of the scope of this paper. A depth-first search strategy was chosen, with the order in which premisses are tried given by a total order consistent with $\sqsubseteq$, as discussed in section 4.1. The transition system PSS_ϕ is redefined to make explicit use of sequences, and treat them as stacks of formulae with the left-most element of the sequence being the top of the stack. The new transition systems is called EM_ϕ for *evaluation model* of ϕ. The transition relation is defined below, where f, δ denotes a stack of formulae with top f:

$$(\theta : f, \delta) \Rightarrow_\phi (Ext(\theta\theta') : \rho(\pi_1(t_{\mathrm{P}1}, t_{\mathrm{R}1})), \ldots, \rho(\pi_n(t_{\mathrm{P}n}, t_{\mathrm{R}n})), \delta)$$

where: $\frac{\pi_1(t_{\mathrm{P}1},t_{\mathrm{R}1})\ldots\pi_n(t_{\mathrm{P}n},t_{\mathrm{R}n})}{\pi(t_{\mathrm{P}},t_{\mathrm{R}})} \in \phi$, θ is the most general unifier of $\theta(f)$ and $\theta(\rho(\pi(t_{\mathrm{P}}, t_{\mathrm{R}})))$, and ρ is a renaming for variables in the formulae to new variables not appearing in the left side of the arrow. The operation *Ext* is the same extension on the substitutions defined in section 3.

Therefore, a deterministic acceptor for EM_ϕ can be used to evaluate deterministic semantics specifications. This deterministic acceptor can be easily implemented in much the same way as a Prolog interpreter, but without the need for any backtracking mechanism. At this point we already have defined a formal framework in which (1) a large class of semantic specifications can be written in a form that is suitable for formal reasoning and manipulation, (2) within the whole class of semantic specifications expressible in the formalism, a subclass of deterministic well formed specifications can be efficiently evaluated.

6 Evaluating Determinate Semantics

Although the definition of a framework for evaluation of operational semantics was completed in the last section, to work in the class of deterministic semantics is sometimes very tedious. Nevertheless we do not want to change the evaluation model, in order to cope with a wider class of semantics specifications. Therefore, we investigated how another class of specifications, namely that of determinate specifications, could be converted into deterministic ones.

This investigation turned out to be very useful for two reason: (1) we proved that determinate specifications and deterministic specifications express the same class of semantics; (2) we designed an algorithm to convert from determinate specifications to deterministic ones, which allows the use of EM_ϕ with determinate specifications.

Regarding (1), the way the proof was carried out gave us confidence that the formal framework of sections 2 and 3 actually provides the level of formality required for the semantics specifications. As far as (2) is concerned, semantics specification can now be written in the more concise and easier to write form of determinate specification, while the transformed deterministic specification is evaluated in EM_ϕ.

In what follow we comment on how the transformation algorithm was designed. But before some new definitions used below are given. An inductive rule $(\mathrm{Pre}, \mathrm{c})$ is *candidate* for f in a state $(\theta : f, \delta)$ if $\theta(f)$ and $\theta(\mathrm{c})$ unify. A candidate rule $(\mathrm{Pre}, \mathrm{c})$ is *applicable* for f in $(\theta : f, \delta)$ if the transition sequence starting at this state and using the rule $(\mathrm{Pre}, \mathrm{c})$ to try the proof for f eventually reaches a state $(\theta : \varepsilon)$ (where ε denotes the empty sequence of formulae). We say informally that two inductive rules *clash* in a state $(\theta : f, \delta)$ if both rules are candidates for f in this state.

Then, notice that, whenever determinate specification are being considered, there is at most one applicable rule at each state of EM_ϕ, but there may be more than one candidate. The problem is that to find the applicable rule in any state is not in general possible without trying at least some premisses of each candidate rule, in order to discharge the non applicable rules. However, from the definition of determinate rules, whenever two rules $\mathcal{R}$ and $\mathcal{R}'$ are candidates for a formula f in a state $(\theta : f, \delta)$, these rules have a very special form:

$$\mathcal{R} : \frac{\pi_1(t_{\mathrm{P}1}, t_{\mathrm{R}1}), \ldots, \pi_i(t_{\mathrm{P}i}, t_{\mathrm{R}i}), \ldots, \pi_n(t_{\mathrm{P}n}, t_{\mathrm{R}n})}{\pi(t_{\mathrm{P}}, t_{\mathrm{R}})}$$

$$\mathcal{R}' : \frac{\pi_1(t'_{\mathrm{P}1}, t'_{\mathrm{R}1}), \ldots, \pi_i(t'_{\mathrm{P}i}, t'_{\mathrm{R}i}), \ldots, \pi'_n(t'_{\mathrm{P}n}, t'_{\mathrm{R}n})}{\pi(t'_{\mathrm{P}}, t'_{\mathrm{R}})}$$

such that: $\pi_k(t_{\mathrm{P}_{\pi_k}}, t_{\mathrm{R}_{\pi_k}}) \equiv_\alpha \pi_k(t'_{\mathrm{P}_{\pi_k}}, t'_{\mathrm{R}_{\pi_k}})$, $1 \leq k < i$, $t_{\mathrm{P}i} \equiv_\alpha t'_{\mathrm{P}i}$, and $t_{\mathrm{R}i} \neq t'_{\mathrm{R}i}$.

This looks very similar to the case for context free grammars, when two productions for the same non-terminal symbol have a common initial segment. For this case there exists an algorithm which factorises the common segment, such that the productions in the factorised grammar can be chosen by looking only at the first symbol (see [ASU86]). Moreover, this factorisation preserves the meaning of the original grammar, in the sense that the original and the factorised grammars generate the same language.

The way the transformation algorithm works is by factoring clashing rules in a similar way as grammars are factorised. Therefore, no alteration must be done on the evaluation model for it to be applied to a wider class of specifications

In the next section, we exemplify the factorisation of sets of rules as discussed above. We give the first example in full, explaining it step by step in order to illustrate the algorithm for the factorisation, which is presented in section 7. In the examples we write $t_{\mathrm{P}} \Rightarrow t_{\mathrm{R}}$ instead of $\Rightarrow (t_{\mathrm{P}}, t_{\mathrm{R}})$, and use concrete syntax instead of the less readable abstract syntax of terms.

6.1 Conditional Expressions

The rules $\mathcal{R}$ and $\mathcal{R}'$ given below are part of a dynamic semantics for a simple expression language, in which the binary relation $\Rightarrow$ intuitively means: $e \Rightarrow r$ iff r is the result of the evaluation of the expression e. $\mathcal{R}$ and $\mathcal{R}'$ define the meaning of the conditional expression, and are written as follows:

$$\mathcal{R} : \frac{e_1 \Rightarrow \textbf{true} \qquad e_2 \Rightarrow r}{\textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3 \Rightarrow r}$$

$$\mathcal{R}' : \frac{e_1 \Rightarrow \textbf{false} \qquad e_3 \Rightarrow r}{\textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3 \Rightarrow r}$$

Clearly, the set $\phi = \{\mathcal{R}, \mathcal{R}'\}$ is determinate, since the formulae $e_1 \Rightarrow \textbf{true}$ and $e_1 \Rightarrow \textbf{false}$ form a distinction pair, which means that in a state $(\theta : \textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3 \Rightarrow r, \alpha)$, of EM_ϕ, $\mathcal{R}$ or $\mathcal{R}'$ are candidate but at most one is applicable. We will show how to factorise ϕ into another set ϕ', such that ϕ' has only one candidate at each state in EM_ϕ.

Step 1: Find a substitution $\mathcal{S}$ which α-converts the phrase of the conclusion of $\mathcal{R}'$ (in this case, **if** e_1 **then** e_2 **else** e_3) into the phrase of the conclusion of $\mathcal{R}$ (**if** e_1 **then** e_2 **else** e_3). Apply $\mathcal{S}$ to $\mathcal{R}'$ obtaining a rule $\mathcal{R}''$. This step is necessary in order to get the same name for corresponding variables in both rules. In this case, $\mathcal{S}$ is the identity mapping, therefore $\mathcal{R}'' = \mathcal{R}'$. Here it is assumed that α-conversion does not alter the meaning of a rule. This is valid since variables are bound to the rules they appear in.

Step 2: Substitute the result, r, of the conclusion of $\mathcal{R}$ for a fresh variable. In this case, change r to r', getting **if** e_1 **then** e_2 **else** $e_3 \Rightarrow r'$

Step 3: From one of the elements of the distinction pair (say, $e_1 \Rightarrow \textbf{true}$), substitutes the result by fresh variables, creating a formula $e_1 \Rightarrow r''$ (where r'' is new).

Step 4: Create a new formula $\mathcal{K}(r'', e_2, e_3, r')$, where $\mathcal{K}$ is a fresh relation's name, and the phrase of $\mathcal{K}$ is r'', e_2, e_3 and the result of $\mathcal{K}$ is r' . Combine the three formulae constructed before into a new rule:

$$\mathcal{R}_1 : \frac{e_1 \Rightarrow r'' \qquad \mathcal{K}(r'', e_2, e_3, r')}{\textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3 \Rightarrow r'}$$

The intuition behind the new relation $\mathcal{K}$ is that it is postponing the decision between the rules $\mathcal{R}$ and $\mathcal{R}'$, by "getting" the result r'', which was "calculated" by the first premiss, and "passing" its value together with the variables e_2, and e_3, to some other rules, while it is also "receiving" the result r' from these rules. The variables e_2, and e_3 are those necessary in the phrases of the remaining premisses $e_2 \Rightarrow r$ of $\mathcal{R}$, and $e_3 \Rightarrow r$ of $\mathcal{R}'$.

Step 5: Build two new formulae $\mathcal{K}(\mathbf{true}, e_2, e_3, r)$ and $\mathcal{K}(\mathbf{false}, e_2, e_3, r)$, and with them construct the two new rules below:

$$\mathcal{R}_2 : \frac{e_2 \Rightarrow r}{\mathcal{K}(\mathbf{true}, e_2, e_3, r)}$$

$$\mathcal{R}_3 : \frac{e_3 \Rightarrow r}{\mathcal{K}(\mathbf{false}, e_2, e_3, r)}$$

And here, the intuition is that $\mathcal{R}_2$ and $\mathcal{R}_3$ will receive the result from the distinction pair, "calculate" in rule $\mathcal{R}_1$, and since both conclusions do not unify, at most one rule can be candidate at any state in a transition of EM_ϕ.

Step 6: The set $\phi' = \{\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3\}$ is the factorised set of rules from ϕ. Since there is no clashing in ϕ' we can stop applying the factorisation.

It is not difficult to show that ϕ and ϕ' are equivalent in the sense that a formula has a ϕ-proof iff it has a ϕ'-proof. Moreover, ϕ' is trivially deterministics, and therefore our goal was achieved.

6.2 Function Application

Consider now a fragment of a semantics for a simple functional programming language. The rules of the specification describe a ternary relation $\vdash \Rightarrow$, with the intuitive meaning that: $(E, e, r) \in \vdash \Rightarrow$ iff r is the result of evaluating e in the environment E. The rules $\mathcal{R}$ and $\mathcal{R}'$ describe a usual semantics for function application:

$$\mathcal{R} : \frac{\begin{array}{c} E \vdash e_1 \Rightarrow [x, e, E'] \qquad E \vdash e_2 \Rightarrow r \\ E' + \{x \mapsto r\} \vdash e \Rightarrow r' \end{array}}{E \vdash e_1 e_2 \Rightarrow r'}$$

$$\mathcal{R}' : \frac{E \vdash e_1 \Rightarrow \mathit{const} \qquad E \vdash e_2 \Rightarrow r \qquad \text{APPLY}\langle \mathit{const}, r\rangle = r'}{E \vdash e_1 e_2 \Rightarrow r'}$$

where $(\langle E, e_1\rangle \Rightarrow [x, e, E'], \langle E, e_1\rangle \Rightarrow \mathit{const})$ is a distinction pair, and $s(\langle E, e_1\rangle \Rightarrow [x, e, E']) = \langle E, e_2\rangle \Rightarrow r$ and $s(\langle E, e_1\rangle \Rightarrow \mathit{const}) = \langle E, e_2\rangle \Rightarrow r$ are trivially α-convertible. We will not go through all the steps of the factorisation, but only show the final factorised set of rules ϕ' below:

$$\mathcal{R}_1 : \frac{E \vdash e_2 \Rightarrow r \qquad E \vdash e_1 \Rightarrow r''' \qquad \mathcal{K}(r''', r, r'')}{E \vdash e_1 e_2 \Rightarrow r''}$$

$$\mathcal{R}_2 : \frac{E' + \{x \mapsto r\} \vdash e \Rightarrow r'}{\mathcal{K}([x, e, E'], r, r')}$$

$$\mathcal{R}_3 : \frac{\text{APLLY}\langle \mathit{const}, r\rangle \Rightarrow r'}{\mathcal{K}(\mathit{const}, r, r')}$$

6.3 Operator Overloading

The last example is a fragment of a static semantics for an expression language in which the operator $+$ is overloaded. The relation $\vdash$: should be understood as follows: $(\mathrm{TE}, e, \tau) \in \vdash$: iff e has type τ in the type environment TE(which maps variables to their types).The rules $\mathcal{R}$ and $\mathcal{R}'$ describe how to type $+$:

$$\mathcal{R} : \frac{\mathrm{TE} \vdash e_1 : \mathbf{int} \qquad \mathrm{TE} \vdash e_2 : \mathbf{int}}{\mathrm{TE} \vdash e_1 + e_2 : \mathbf{int}}$$

$$\mathcal{R}' : \frac{\mathrm{TE} \vdash e_1 : \mathbf{real} \qquad \mathrm{TE} \vdash e_2 : \mathbf{real}}{\mathrm{TE} \vdash e_1 + e_2 : \mathbf{real}}$$

where $(\mathrm{TE} \vdash e_1 : \mathbf{int}, \mathrm{TE} \vdash e_1 : \mathbf{real})$ is a distinction pair (not unique in this case). The factorisation of the set $\phi = \{\mathcal{R}, \mathcal{R}'\}$ produce the following three rules:

$$\mathcal{R}_1 : \frac{\mathrm{TE} \vdash e_1 : \tau \qquad \mathcal{K}(\tau, \mathrm{TE}, e_2, \tau')}{\mathrm{TE} \vdash e_1 + e_2 : \tau'}$$

$$\mathcal{R}_2 : \frac{\mathrm{TE} \vdash e_2 : \mathbf{int}}{\mathcal{K}(\mathbf{int}, \mathrm{TE}, e_2, \mathbf{int})}$$

$$\mathcal{R}_3 : \frac{\mathrm{TE} \vdash e_2 : \mathbf{real}}{\mathcal{K}(\mathbf{real}, \mathrm{TE}, e_2, \mathbf{real})}$$

7 The Factorisation Algorithm

Initially, we state the problem we want to solve by factorisation of rules, and then present a factorisation algorithm called FAIR, for *Factorisation Algorithm for Inductive Rules.* For an operational semantics definition $\mathcal{D} = \langle \Omega, \mathcal{O} \rangle$ where $\Omega = \langle \mathrm{S}, \Sigma, \Pi \rangle$ and $\mathcal{O} = \langle |\mathrm{A}|, \alpha, \phi \rangle$, the factorisation problem is to transform ϕ into another set ϕ', possibly adding new relation names in Π making it into Π', such that

1. ϕ' is deterministic, and
2. for all formulae $f \in \mathrm{WFF}_{\mathrm{X}}(\Omega)$, f has a ϕ-proof iff f has a ϕ'-proof.

In order to make the presentation of the algorithm clearer, we define the *candidate class* of an inductive definition ϕ as a partition $\mathcal{CC}_\phi$ of ϕ such that two rules (Pre, c) and (Pre$'$, c$'$) belong to the same element of $\mathcal{CC}_\phi$ iff $\mathrm{c} = \pi(t_{\mathrm{P}}, t_{\mathrm{R}})$ and $\mathrm{c} = \pi(t'_{\mathrm{P}}, t'_{\mathrm{R}})$, and $t_{\mathrm{P}} \equiv_\alpha t'_{\mathrm{P}}$, i.e. the rules clash. Therefore, for any formula $f \in \mathrm{WFF}_{\mathrm{X}}(\Omega)$ each candidate rule of f is in at most one set in $\mathcal{CC}_\phi$.

We present the algorithm FAIR for the special case when the candidate class $\mathcal{CC}_\phi$ has each element with at most two rules in it. The generalisation is straightforward and we omit it here for clarity.

FAIR:

Given $\langle |\mathrm{A}|, \alpha, \phi \rangle$, and $\psi \in \mathcal{CC}_\phi$ such that $\psi = \{\mathcal{R}, \mathcal{R}'\}$:

$$\mathcal{R} : \frac{\pi_1(t_{\mathrm{P}1}, t_{\mathrm{R}1}), \ldots, \pi_n(t_{\mathrm{P}n}, t_{\mathrm{R}n})}{\pi(t_{\mathrm{P}}, t_{\mathrm{R}})}$$

$$\mathcal{R}' : \frac{\pi'_1(t'_{\mathrm{P}1}, t'_{\mathrm{R}1}), \ldots, \pi'_m(t'_{\mathrm{P}m}, t'_{\mathrm{R}m})}{\pi'(t'_{\mathrm{P}}, t'_{\mathrm{R}})}$$

where: (1) $t_{\mathrm{P}} \equiv_\alpha t'_{\mathrm{P}}$; (2) $\pi_i(t_{\mathrm{P}i}, t_{\mathrm{R}i})$, and $\pi'_j(t'_{\mathrm{P}j}, t'_{\mathrm{R}j})$ form a distinction pair; (3) and $s(\pi_i(t_{\mathrm{P}i}, t_{\mathrm{R}i})) \equiv_\alpha s(\pi'_j(t'_{\mathrm{P}j}, t'_{\mathrm{R}j}))$. Factorise ψ using the steps below:

Step 1: Determine one substitution θ, from $\mathcal{V}(t'_{\mathrm{P}})$ to $\mathcal{V}(t_{\mathrm{P}})$, such that $\theta(t'_{\mathrm{P}}) = t_{\mathrm{P}}$. Apply θ to $\mathcal{R}'$ to get a rule $\mathcal{R}'' = \mathcal{S}(\mathcal{R}')$. This step is necessary to ensure that corresponding variables in $\mathcal{R}$ and $\mathcal{R}''$ are the same. And it is valid since variables are bound to the rule they appear in.

Step 2: Generate a new formula $\pi(t_{\mathrm{P}}, v_{\mathrm{R}})$, $v_{\mathrm{R}} = \mathcal{S}'(t_{\mathrm{R}})$, where $\mathcal{S}'$ is a mapping replacing the term t_k by a fresh variable v_k for each term in t_{R}, such that $\mathcal{S}'$ preserves the sorts of the terms in the new variables.

Step 3: From $\pi_i(t_{\mathrm{P}i}, t_{\mathrm{R}i})$, build a new formula $\pi_i(t_{\mathrm{P}i}, v_{\mathrm{R}i})$, $v_{\mathrm{R}i} = \mathcal{S}''(t_{\mathrm{R}i})$, where $\mathcal{S}''$ is mapping replacing the term t_{ik} by a fresh variable v_{ik} for each term in $t_{\mathrm{R}i}$, such that $\mathcal{S}''$ preserves the sorts of the terms in the new variables.

Step 4: Create a new formula $\mathcal{K}(v_{\mathrm{R}i}, \mathrm{Z}, v_{\mathrm{R}})$, where $\mathcal{K}$ is a new relation's name, v_{R} was defined in Step 2, $v_{\mathrm{R}i}$ was defined in step 3, and Z is a sequence of variables obtained from a set of variables given by the formulae:

$$\mathrm{V} = \left(\left(\bigcup_{k=1}^{i-1} \mathcal{V}(t_{\mathrm{R}k}) \cup \mathcal{V}(t_{\mathrm{P}}) \right) \cap \bigcup_{k=i+1}^{n} \mathcal{V}(t_{\mathrm{P}k}) \right) \cup \tag{1}$$

$$\left(\left(\bigcup_{k=1}^{j-1} \mathcal{V}(t''_{\mathrm{R}k}) \cup \mathcal{V}(t''_{\mathrm{P}}) \right) \cap \bigcup_{k=j+1}^{m} \mathcal{V}(t''_{\mathrm{P}k}) \right) \cup \tag{2}$$

$$\left(\bigcup_{k=1}^{i-1} \mathcal{V}(\pi_k(t_{\mathrm{P}k}, t_{\mathrm{R}k})) \cap \mathcal{V}(t_{\mathrm{R}}) \right) \cup \tag{3}$$

$$\left(\bigcup_{k=1}^{j-1} \mathcal{V}(\pi''_k(t''_{\mathrm{P}k}, t''_{\mathrm{R}k}) \cap \mathcal{V}(t''_{\mathrm{R}}) \right) \tag{4}$$

such that $\mathrm{Z} = v_{s1}, \ldots, v_{sn}$, whenever $\mathrm{V} = \{v_{s1}, \ldots, v_{sn}\}$. Then the direction of $\mathcal{K}$ is obtained by getting the terms in v_{R_i} and Z as the phrases, and the ones in v_{R} as the results of $\mathcal{K}$.

Step 5: Create two new formulae:

$$\mathcal{K}(t_{\mathrm{R}i}, \mathrm{Z}, t_{\mathrm{R}})$$

$$\mathcal{K}(t''_{\mathrm{R}j}, \mathrm{Z}, t''_{\mathrm{R}})$$

Step 6: Build $\psi' = \{\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3\}$ in the following way:

$$\mathcal{R}_1 : \frac{\overline{s}(\pi(t_{\mathrm{R}i}, t_{\mathrm{P}i})) \qquad \pi_i(t_{\mathrm{P}i}, v_{\mathrm{R}i}) \qquad \mathcal{K}(v_{\mathrm{R}i}, \mathrm{Z}, v_{\mathrm{R}})}{\pi(t_{\mathrm{P}}, v_{\mathrm{R}})}$$

$$\mathcal{R}_2 : \frac{\pi_{i+1}(t_{\mathrm{P}i+1}, t_{\mathrm{R}i+1}), \ldots, \pi_n(t_{\mathrm{P}n}, t_{\mathrm{R}n})}{\mathcal{K}(t_{\mathrm{R}i}, \mathrm{Z}, t_{\mathrm{R}})}$$

$$\mathcal{R}_3 : \frac{\pi''_{j+1}(t''_{\mathrm{P}j+1}, t''_{\mathrm{R}j+1}) \ldots \pi''_m(t''_{\mathrm{P}m}, t''_{\mathrm{R}m})}{\mathcal{K}(t''_{\mathrm{R}j}, \mathrm{Z}, t''_{\mathrm{R}})}$$

The factorised set of rules is given by: $\phi' = (\phi \setminus \psi) \cup \psi'$, with the appropriated extension to Π to introduce the new relation $\mathcal{K}$.

Step 7: Reapply FAIR to ϕ' until $\mathcal{CC}_{\phi'}$ has only singleton elements.

8 The Correctness Proof of FAIR

In this section we present a proof that FAIR is correct according to its specification, given in Section 7. We assume that FAIR only produces sets of well formed rules, which can be easily checked from the form of the generated rules, and will not be proved here.

It is necessary to show that for a semantics specification $\mathcal{D} = \langle \Omega, \mathcal{O} \rangle$, where $\Omega = \langle \mathrm{S}, \Sigma, \Pi \rangle$ and $\mathcal{O} = \langle |\mathrm{A}|, \alpha, \phi \rangle$, FAIR applied to ϕ produces a ϕ' such that:

1. ϕ' is deterministic, and
2. for all formulae $f \in \mathrm{WFF}_{\mathrm{X}}(\Omega)$, f has a ϕ-proof iff f has a ϕ'-proof.

The proof is given for the case where each element of $\mathcal{CC}_\phi$ has at most two elements. We assume that a generic element $\psi \in \mathcal{CC}_\phi$ is $\psi = \{\mathcal{R}, \mathcal{R}'\}$, as in section 7, and that FAIR applied to ψ produces $\psi' = \{\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3\}$ (also as in section 7). The proof is split in two cases regarding the items 1 and 2 above.

Case 1: ϕ' is deterministic

To show that ϕ' is deterministic is equivalent, by the definition of *candidate class*, as to show that the candidate class $\mathcal{CC}_{\phi'}$ has only singleton elements. Suppose $\phi' = (\phi \setminus \psi) \cup \psi'$ is the result of the first step of the application of FAIR on ϕ, as described above. We first show that $\mathcal{CC}_{\phi'}$ has exactly one non-singleton element less than $\mathcal{CC}_{\phi}$. This is the same as to show that the factorisation gets rid of one non-singleton set $\psi \in \mathcal{CC}_{\phi}$ and does not introduce any other new non-singleton set in $\mathcal{CC}_{\phi'}$ by adding the set of rules ψ'.

The first argument is trivial since $\psi \not\subseteq \phi'$. For the second argument suppose that there exists a new ψ'' non-singleton set in $\mathcal{CC}_{\phi'}$, after applying FAIR. For this to be the case at least one rule in ψ'' has to be one of the rules in ψ', otherwise $\psi'' \in \mathcal{CC}_{\phi}$ and therefore would not be new in $\mathcal{CC}_{\phi'}$.

Suppose that the rule $\mathcal{R}_1 \in \psi'$ is one rule in ψ'', then any other rule in ψ'' has to be of the form $\pi(t'_{\mathrm{P}}, t'_{\mathrm{R}})$ such that $t_{\mathrm{P}} \equiv_{\alpha} t'_{\mathrm{P}}$. Suppose there exists such a rule, and call it $\mathcal{R}''$. But $\mathcal{R}''$ cannot be neither $\mathcal{R}_2$ nor $\mathcal{R}_3$, since $\pi \neq \mathcal{K}$. If $\mathcal{R}''$ is in ϕ' then it has to be in $\phi \setminus \psi$, but then it should be in ψ in the first place and therefore cannot be in ϕ', therefore $\mathcal{R}''$ does not exist and ψ'' is singleton.

Suppose that $\mathcal{R}_2$ is one rule in ψ'', then any other rule in ψ'' has to be $\mathcal{R}_3$ for the relation $\mathcal{K}$ is different from any other relation in ϕ'. But $t_{\mathrm{R}i} \neq t''_{\mathrm{R}j}$ therefore $\mathcal{R}_3$ cannot be in ψ'', therefore ψ'' is singleton.

The same argument applies if $\mathcal{R}_3$ is one element of ψ''. Therefore FAIR does not introduce any new non-singleton set of rules in $\mathcal{CC}_{\phi'}$.

To complete the proof, it is easy to see that if we iterate FAIR on the set ϕ it will eventually stops and the resulting set of rules will be such that its candidate class has only singleton elements.

Case 2: Semantic Equivalence between ϕ and ϕ'

Intuitively, two inductive definitions ϕ and ϕ' are equivalent w.r.t. the set of sentences $\mathrm{Sen}(\Omega)$ iff they prove exactly the same subset of $\mathrm{Sen}(\Omega)$. This definition of semantic equivalence in rather different from the one that imposes that $\mathrm{I}(\phi) = \mathrm{I}(\phi')$, this one being too restrictive for the case of the factorisation algorithm.

This is so because when factoring ϕ producing ϕ', the algorithm introduces new relations and formulae making the above equality false in general, although proofs of objects in $\mathrm{Sen}(\Omega)$ are still preserved.

Therefore, we want to prove that for all sentences $\overline{f} \in \mathrm{Sen}(\Omega)$, $\overline{f}$ has a ϕ-proof iff $\overline{f}$ has a ϕ'-proof. Since a formula $f \in \mathrm{WFF}_{\mathrm{X}}(\Omega)$ is an abbreviation for a set of sentences in $\mathrm{Sen}(\Omega)$, this would in turn give a proof that for all f, f has a ϕ-proof iff f has a ϕ'-proof.

Soundness: *If $\overline{f}$ has a ϕ'-proof then $\overline{f}$ has a ϕ-proof.*

We want to assure that ϕ' cannot prove more Ω-sentences than ϕ can do. The proof is by induction on the length of the ϕ'-proof for $\overline{f}$.

Base case: A proof of length 1 for $\overline{f}$ must be of the form of an instance $\theta(\emptyset, \mathrm{c})$ for some substitution θ, and rule $(\emptyset, \mathrm{c}) \in \phi'$. But FAIR does not introduce any new rule of the form $(\emptyset, \mathrm{c})$ in ϕ' that was not already in ϕ. Therefore, ϕ'-proofs of length 1 are also ϕ-proofs of length 1.

Induction step: Suppose $\overline{f_0}, \ldots, \overline{f_N}$ is a ϕ'-proof of length N for $\overline{f}$, such that $\overline{f_N} = \overline{f}$ and each $\overline{f_I}$, $0 \leq I \leq N$, is a sentence in $\mathrm{Sen}(\Omega)$. We want to show that $\overline{f_0}, \ldots, \overline{f_N}$ is also a ϕ-proof for $\overline{f}$.

From the definition of ϕ-proofs we must show that:

- $\overline{f_N} = \overline{f}$, but this is trivial because it is the initial assumption.
- for each $\overline{f_I}$, there exists an $\mathrm{X}_I \subseteq \{\overline{f_K} | K < I\}$ such that $(\mathrm{X}_I, \overline{f_I})$ is an instance of some rule in ϕ.

We first show how to obtain $(\mathrm{X}_N, \overline{f_N})$, and then apply the induction hypothesis to get the remaining instances. The only case that $(\mathrm{X}_N, \overline{f_N})$ is not an instance of some rule in ϕ is when it is an instance of $\mathcal{R}_1$, for this rule is not in ϕ. Moreover, $(\mathrm{X}_N, \overline{f_N})$ cannot be an instance of $\mathcal{R}_2$ or $\mathcal{R}_3$ because instantiations of the conclusions of those rules are not in $\mathrm{Sen}(\Omega)$ (remember that $\mathcal{K} \not\in \Pi$). But if $(\mathrm{X}_N, \overline{f_N})$ is an instance of $\mathcal{R}$, then X_N must be:

$$\{\theta(\pi_1(t_{\mathrm{P}1}, t_{\mathrm{R}})), \ldots, \theta(\pi_i(t_{\mathrm{P}i}, v_{\mathrm{R}i})), \theta(\mathcal{K}(v_{\mathrm{R}i}, \mathrm{Z}, v_{\mathrm{R}}))\}$$

and $\overline{f_N} = \theta(\pi(t_{\mathrm{P}}, v_{\mathrm{R}}))$ for some instantiation θ. From the definition of ϕ'-proofs there exists a set $\mathrm{X} \subseteq \{\overline{f_K} | K < N-1\}$, such that $(\mathrm{X}, \theta(\mathcal{K}(v_{\mathrm{R}i}, \mathrm{Z}, v_{\mathrm{R}})))$ is an instance of either rule $\mathcal{R}_2$ or $\mathcal{R}_3$ in ϕ'. Suppose it is rule $\mathcal{R}_2$, then:

$$\mathrm{X} = \{\theta'(\pi_{i+1}(t_{\mathrm{P}i+1}, t_{\mathrm{R}i+1})), \ldots, \theta'(\pi_n(t_{\mathrm{P}n}, t_{\mathrm{R}n}))\}$$

and $\theta(\mathcal{K}(v_{\mathrm{R}i}, \mathrm{Z}, v_{\mathrm{R}})) = \theta'(\mathcal{K}(t_{\mathrm{R}i}, \mathrm{Z}, t_{\mathrm{R}}))$ for some substitution θ'. This last condition becomes $\theta(v_{\mathrm{R}i}) = \theta'(t_{\mathrm{R}i})$, $\theta(\mathrm{Z}) = \theta'(\mathrm{Z})$, and $\theta(v_{\mathrm{R}}) = \theta'(t_{\mathrm{R}})$. But then the rule $(\mathrm{X}', \overline{f_N})$, where:

$$\mathrm{X}' = \{\theta(s(\pi_i(t_{\mathrm{P}i}, t_{\mathrm{R}i}))), \theta\theta'(\pi_i(t_{\mathrm{P}i}, t_{\mathrm{R}i})), \theta'(\pi_{i+1}(t_{\mathrm{P}i+1}, t_{\mathrm{R}i+1})), \ldots, \theta'(\pi_n(t_{\mathrm{P}n}, t_{\mathrm{R}n}))\}$$

is an instance of $\mathcal{R}$ in ϕ. Moreover, because of the condition that $\theta(v_{\mathrm{R}i}) = \theta'(t_{\mathrm{R}i})$, we have that $\theta\theta'(\pi_i(t_{\mathrm{P}i}, t_{\mathrm{R}i})) = \theta(\pi_i(t_{\mathrm{P}i}, v_{\mathrm{R}i}))$. The only problem is that θ and θ' may map the same variable to different terms, and therefore the composition $\theta\theta'$ may not be consistent. But when a variable v is in dom θ and also in dom θ' it is in Z, as can be easily checked from equations 1 to 4 in section 7. Therefore, the condition that $\theta(\mathrm{Z}) = \theta'(\mathrm{Z})$ assures that $\theta(v) = \theta'(v)$, for each $v \in \mathrm{Z}$.

Now, each element $\overline{f'} \in \mathrm{X}'$ is some sentence in $\overline{f_0}, \ldots, \overline{f_N}$. Therefore, there exists an instance $(\mathrm{X}'', \overline{f'})$ of some rule in ϕ', and a ϕ'-proof for $\overline{f'}$ of length less then N. By the induction hypothesis there exists a ϕ-proof for each $\overline{f'}$, therefore, there exists an instance

$(\mathrm{X}''', \overline{f'})$ of a rule in ϕ for each $\overline{f'}$. These ϕ-proofs together with the instance $(\mathrm{X}', \overline{f_N})$ makes $\overline{f_0}, \ldots, \overline{f_N}$ also a ϕ-proof for $\overline{f}$, since the condition that $\theta(v_{\mathrm{R}}) = \theta'(t_{\mathrm{R}})$ makes $\theta(\pi(t_{\mathrm{P}}, v_{\mathrm{R}})) = \overline{f}$.

The same argument applies when we chose rule $\mathcal{R}_3$ instead of $\mathcal{R}_2$. Therefore, soundness holds.

Completeness: *If $\overline{f}$ has a ϕ-proof then $\overline{f}$ has a ϕ'-proof.*

This proof is by induction on the length of a ϕ-proof for $\overline{f}$.

Base case: The proofs of length 1 are not changed under the application of FAIR by a similar argument to that used in the proof of soundness.

Induction step: Suppose $\overline{f_0}, \ldots, \overline{f_N}$ is a ϕ-proof of length N for $\overline{f}$, such that $\overline{f_N} = \overline{f}$ and each $\overline{f_I}$, $0 \leq I \leq N$ is a sentence in $\mathrm{Sen}(\Omega)$. We want to show that $\overline{f_0}, \ldots, \overline{f_N}$ is also a ϕ'-proof for $\overline{f}$.

Basically, we must do the same steps as in the proof of the soundness. First, we will obtain an instance $(\mathrm{X}_N, \overline{f_N})$, of some rule in ϕ' and then apply the induction hypothesis to get a ϕ'-proof for $\overline{f}$. The only case that $(\mathrm{X}_N, \overline{f_N})$, for $\mathrm{X}_N \subseteq \{\overline{f_K} | K < N\}$, is not an instance of some rule in ϕ' is when it is an instance of either $\mathcal{R}$ or $\mathcal{R}'$, for these rules are not in ϕ'. Suppose it is of $\mathcal{R}$, then:

$$\mathrm{X}_N = \{\theta(\pi_1(t_{\mathrm{P}1}, t_{\mathrm{R}})), \ldots, \theta(\pi_n(t_{\mathrm{P}n}, t_{\mathrm{R}n}))\}$$

and $\overline{f_N} = \theta(\pi(t_{\mathrm{P}}, t_{\mathrm{R}}))$ for some instantiation θ. But now we can get an instantiation $(\mathrm{X}, \overline{f_N})$ of $\mathcal{R}_1$ and $(\mathrm{X}', \overline{f'})$ of $\mathcal{R}_2$ where:

$$\mathrm{X} = \{\theta(s(\pi_i(t_{\mathrm{P}i}, t_{\mathrm{R}i}))), \theta'(\pi_i(t_{\mathrm{P}i}, v_{\mathrm{R}i})), \theta'(\mathcal{K}(v_{\mathrm{R}i}, \mathrm{Z}, v_{\mathrm{R}}))\}$$

$$\mathrm{X}' = \{\theta(\pi_{i+1}(t_{\mathrm{P}i+1}, t_{\mathrm{R}i+1})), \ldots, \theta(\pi_n(t_{\mathrm{P}n}, t_{\mathrm{R}n}))\}$$

and $\overline{f'} = \theta(\mathcal{K}(t_{\mathrm{R}i}, \mathrm{Z}, t_{\mathrm{R}}))$, with the condition that $\theta'(v_{\mathrm{R}i}) = \theta(t_{\mathrm{R}i})$, $\theta'(\mathrm{Z}) = \theta(\mathrm{Z})$, and $\theta'(v_{\mathrm{R}}) = \theta(t_{\mathrm{R}})$.

The only remaining task is to find a proper instantiation θ'. But this is easily done by first composing θ with $[\mathcal{S}'(t_{\mathrm{R}})/t_{\mathrm{R}}]$, where $\mathcal{S}'$ is the mapping defined in step 3 of FAIR. Then compose $\theta[\mathcal{S}'(t_{\mathrm{R}})/t_{\mathrm{R}}]$ with $[\mathcal{S}''(t_{\mathrm{R}i})/t_{\mathrm{R}i}]$, where $\mathcal{S}''$ is the mapping defined in step 4 of FAIR, getting $\theta' = \theta[\mathcal{S}'(t_{\mathrm{R}})/t_{\mathrm{R}}][\mathcal{S}''(t_{\mathrm{R}i})/t_{\mathrm{R}i}]$.

Building θ' in this way also assures that $\overline{f_N} = \theta'(\pi(t_{\mathrm{P}}, v_{\mathrm{R}}))$ which is a condition for $(\mathrm{X}, \overline{f_N})$ to be an instance of $\mathcal{R}_1$. Moreover, because of the condition that $\theta'(v_{\mathrm{R}i}) = \theta(t_{\mathrm{R}i})$, it is trivial that $\theta'(\pi_i(t_{\mathrm{P}i}, v_{\mathrm{R}i})) = \theta(\pi_i(t_{\mathrm{P}i}, t_{\mathrm{R}i}))$ holds.

Therefore, each sentence $\overline{f''}$ in X or in X', that is not the sentence $\theta(\mathcal{K}(v_{\mathrm{R}i}, \mathrm{Z}, v_{\mathrm{R}}))$, is some sentence in X_N, therefore is also in $\overline{f_0}, \ldots, \overline{f_N}$. Therefore, there exists an instance $(\mathrm{X}'', \overline{f''})$ of some rule in ϕ, and a ϕ-proof for $\overline{f''}$ of length less then N. By the induction hypothesis there exists a ϕ'-proof for $\overline{f''}$, therefore there exists an instance $(\mathrm{X}''', \overline{f''})$ of a rule in ϕ'. Therefore, $\overline{f_0}, \ldots, \overline{f_N}$ is also a ϕ'-proof for $\overline{f}$.

The same argument is valid when the rule $\mathcal{R}'$ is chosen. Therefore, completeness holds, finishing the proof of the semantic equivalence between ϕ and ϕ'. Which also finishes the proof that FAIR is correct.

$\square$

9 Conclusions

In this work a characterisation of operational semantics was presented as inductive definitions of sets of relations. For this characterisation a notion of finite length proof can be used to check whether objects are in some relation. Therefore, the same notion of proofs can be used to check the meaning of objects according to the semantics. In order to complete what is called the meta-semantics of the characterisation of operational semantics, a proof search strategy for inductive definitions, called PSS_ϕ, was defined. This search strategy can be proved sound and complete according to the definition of ϕ-proofs.

The next step in the definition of a framework for evaluation of operational semantics, was the definition of an evaluation model EM_ϕ for a particular sub-class of deterministic semantics specifications. The objective was that EM_ϕ could then be given an efficient implementation. In order to formalise the notion of deterministic specifications a ***specification meta-language*** was designed. At this stage we faced various design options, which were pointed out explicitly for two main reasons: first, it makes other possibilities explicit, creating space for further investigations on the discharged options. Second, it emphasizes that such decisions are not inherent from the semantics formalism, but are done only to achieve a more efficient evaluation.

For a particular set of design choices, the evaluation model EM_ϕ exploits a depth-first search strategy, with the order of the premisses given by a total order which is consistent with the partial order $\sqsubseteq$ of section 4.1. The use of the partial order has similar effect as to use *delays* in Prolog programs [Nai85]. One difference being that the partial order is statically determined whereas the delaying mechanism is done at run time. On the other hand, some delays can be automatically generated. Although the partial order can be automatically determined we did not investigate the possibility of automatically defining the directions on the relations. It would be interesting to do some investigation on this problem, and it is being considered for future extensions on the evaluation model.

The last work that has been done on the evaluation model is the definition of an algorithm to convert from determinate specifications into deterministic ones. This work is important for three main reasons: (1) it shows that determinate specifications are not more general than deterministic ones w.r.t the class of semantics that can be expressed in them, (2) it demonstrates the suitability of the theoretical framework in the formal manipulation of semantic specifications, and (3) it extends the applicability of the evaluation model.

A possible improvement on the evaluation model is to further exploit the use of the partial order on the proofs of premisses. This would involve to define a search strategy similar to that used in parallel logic programming languages. The basic idea is that if two formulae f and f' are such that $f \sqsubseteq f'$ and $f' \sqsubseteq f$, then they can be tried in parallel, or rather, in any order among themselves. The partial order has two advantages when using the evaluation model in a real programming tool. First, a more efficient evaluation could be achieved if the parallelism was actually implemented. Second, for tools like debuggers, the partial order could be used to allow more flexible and powerful debugging functions.

Although some work on EM_ϕ has still to be done, it already provides a good framework for reasoning about semantics based evaluation of programs. The evaluation model together with the formal characterisation of operational semantics of section 2, and with the *meta-language*, form a complete framework in which semantics can be written, executed, and formally reason about.

Acknowledgements

The author would like to thanks his supervisor at Edinburgh, Kevin Mitchell, for many discussions on various topics of this work and for many suggestions on this paper. Thanks to Ed Kazmierczak for helpful discussions on first order structures and free algebras, and Michael Mendler for proof reading a previous version of this paper. Thanks also to Roberto Hexsel and Dave Berry for helpful comments on a previous version of the factorisation algorithm. The author is supported by a Brazilian government scholarship, CNPq process number 20.0459/88.0/CC.

References

[Acz77] Peter Aczel. An introduction to inductive definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, chapter C.7, pages 739–782. North-Holland Publishing Company, 1977.

[Ast89] Egidio Astesiano. Inductive semantics. In *Lecture Notes of the State of the Art Seminar on Formal Description of Programming Concepts, Petropolis, Brazil.* IFIP TC2 WG 2.2, 1989.

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: principles, techniques, and tools.* Addison-Wesley Publishing Company, 1986.

[BCD$^+$87] P. Borras, D. Clément, Th. Despeyroux, J. Incerpi, G. Kahn, B. Lang, and V. Pascual. CENTAUR: The system. Technical Report 777, INRIA, Rocquencourt, France, December 1987.

[Ber90] Dave Berry. *Generating Program Animators from Programming Language Semantics.* PhD thesis, LFCS, Department of Computer Science, University of Edinburgh, Edinburgh, EH9 3JZ, Scotland, 1990.

[Chi89] C. Chin. A support tool for operational semantics. Undergraduate Project Report, Department of Computer Science, University of Edinburgh, May 1989.

[Des88] T. Despeyroux. TYPOL: a formalism to implement Natural Semantics. Technical Report 94, INRIA, Sophia-Antipolis, France, March 1988.

[GB90] Joseph A. Goguen and Rod Burstall. Institutions: abstract model theory for specification and programming. Technical Report ECS-LFCS-90-106, LFCS, Department of Computer Science, University of Edinburgh, January 1990.

[HMT89] Robert Harper, Robin Milner, and Mads Tofte. The definition of Standard ML (version 3). Technical Report ECS-LFCS-89-81, LFCS, Department of Computer Science, University of Edinburgh, May 1989.

[Kah87] G. Kahn. Natural Semantics. Gipe project second annual review report, INRIA, Sophia-Antipolis, France, January 1987.

[Nai83] Lee Naish. Mu-prolog 3.0 reference manual. Technical report, Department of Computer Science, Melbourne University, July 1983.

[Nai85] Lee Naish. *Negation and Control in Prolog*, volume 238 of *Lecture Notes in Computer Science*. Springer Verlag, 1985.

[Plo81] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, Aarhus, Denmark, September 1981.

[TWW78] James W. Thatcher, Eric G. Wagner, and Jesse B. Wright. Notes on algebraic fundamentals for theoretical computer science. In *FAICS '78*, pages 22–28, May 1978.