Semantics for Interactive Higher-order Functional-logic Programming

Makoto Hamana

Doctoral Program in Engineering
University of Tsukuba

Contents

1 Introduction **5**
- 1.1 Background . . . 6
  - 1.1.1 Functional Programming Languages . . . 6
  - 1.1.2 Interactive Functional Programming Languages . . . 7
  - 1.1.3 Logic Programming Languages . . . 7
  - 1.1.4 Algebraic Specification Languages . . . 8
  - 1.1.5 First-order Functional-logic Programming Languages . 8
  - 1.1.6 Higher-order Functional-logic Programming Languages 10
- 1.2 Our Approach . . . 10
  - 1.2.1 First-order FLP . . . 10
  - 1.2.2 Interactive FLP . . . 12
  - 1.2.3 Higher-order FLP . . . 15

2 Many-sorted First-order FLP **17**
- 2.1 Many-sorted Conditional Equational Logic . . . 17
- 2.2 Many-sorted Term Rewriting and Narrowing . . . 22
- 2.3 Algebraic Semantics . . . 29
- 2.4 Many-sorted First-order FLP in Equational Logic Framework . 31
- 2.5 Least Complete Herbrand Model . . . 33
- 2.6 Equivalence of the Validity in Models . . . 37
- 2.7 Categorical Semantics . . . 43
- 2.8 Deriving Algebraic Semantics from Categorical Semantics . . . 47

3 Interactive First-order FLP **51**
- 3.1 Computational Metalanguage . . . 52
- 3.2 Categorical Semantics Based on Strong Monads . . . 54
- 3.3 Interactive First-order FLP . . . 57
- 3.4 The Strong Monad for Side-effects . . . 59
- 3.5 A Translation from Interactive FLPs to CTRSs . . . 60
- 3.6 Soundness of the Translation $_^{\circ}$ . . . 65
- 3.7 Interaction and Solving Equations . . . 71

3.8 Meaning Preservation of the Translation _° 74

4 Simply-typed Applicative FLP **77**
4.1 Simply-typed Applicative Conditional Equational Logic 77
4.2 Simply-typed Applicative FLP in Equational Logic Framework 79
4.3 Minimal Applicative Herbrand Model 82
4.4 Soundness and Completeness 85

5 Conclusion **89**

Chapter 1

Introduction

The purpose of this thesis is to provide axiomatic, operational, algebraic, and categorical semantics for interactive higher-order functional-logic programming.

These semantics are important for the following purposes:

- the operational semantics for execution,
- the algebraic semantics for a functional interpretation,
- the axiomatic semantics for a logical interpretation,
- the categorical semantics for the formalization of interaction.

Moreover, to ensure correctness of them there must be a correspondence between these semantics. To give correct semantics, we extend the following three classes of functional logic programming languages in a step-by-step manner and give semantics to these languages:

- many-sorted first-order functional-logic programming languages,
- interactive many-sorted first-order functional-logic languages,
- simply typed applicative functional-logic languages.

We formalize the syntactical and logical part of these classes of functional-logic languages as equational logic and give their semantics. In this thesis, we use the word "FLP" to mean "functional-logic program".

1.1 Background

The family of languages called declarative programming languages are distinguished by a mathematically clear semantics and have been the subject of extensive research efforts. Languages belonging to this family are functional programming languages, logic programming languages, algebraic specification languages[1], and functional-logic programming languages. The last family is the topic of this thesis. First, we briefly review the basic ideas and semantics of these languages. Afterwards, we discuss that a functional-logic language can be considered as an integration of a functional, logic and algebraic specification language.

1.1.1 Functional Programming Languages

Functional programming languages are programming languages that regard mathematical "functions", which compute an output value from a given input value, as "programs" and construct more complex programs by composing functions. A functional program is similar in style to the notation of function definitions in mathematics, e.g.

$$\mathsf{f}\ \mathsf{x} = \mathsf{x} + 1.$$

Execution of a functional program means to compute the value of a given expression. Operational semantics of functional languages are given by (some variants of) lambda calculus [Bar84], combinatory logic [CF58, CHS72], or term rewriting systems [DJ90, Klo92]. More mathematical semantics are given by denotational semantics using domain theory [Sco71, Gun92] or categorical semantics by cartesian closed categories [LS86, Cur93].

Modern functional languages like Scheme [Re91], ML, Haskell, Clean are *higher-order* functional languages where functions can be used as data. From the viewpoint of "programs as mathematical functions", functional languages are partitioned into two classes: *pure* and *impure* functional languages.

Impure functional languages like Scheme and ML have side-effects, such as assignments or I/O. Although these features are necessary in realistic programming, they are not compatible with the original idea of "programs as mathematical functions". For example, a function `read` which reads a

[1] Although these are not actually programming languages, they often have an evaluation mechanism, e.g. OBJ3 [GWMF96].

character from a file (if the file consists of mutually different characters) returns different values whenever it is called. Since mathematical functions should always map to a unique value for the same argument, `read` is not a mathematical function in the original sense.

1.1.2 Interactive Functional Programming Languages

In pure functional languages like Haskell or Clean, functions can be regarded as truly mathematical functions; hence, the lambda calculus, denotational semantics, etc. directly correspond to such functional languages. Due to such mathematical clarity and simplicity, pure functional languages have been shown to be successful, and many arguments have been provided for the advantages of doing without any side-effects. However, in order to write a realistic program, it must be possible for functional programs to interact with the outside world. A substantial amount of mechanisms have been proposed and implemented for interaction in functional languages without destroying the pureness of the language [Gor94]. The recent, most dominant styles of treating interaction in pure functional languages are

- the monadic style as in Haskell [Wad90], and
- the explicit state passing style as in Clean [AP95].

These ideas have influenced our approach for handling interactions in functional-logic programming. We discuss this subject further in Chapter 3.

1.1.3 Logic Programming Languages

Logic programming languages like PROLOG are based on Horn clause logic. A program consists of a set of definite clauses of the following form (here $\overrightarrow{x}$ denotes a sequence of variables).

$$\mathsf{P}(\overrightarrow{x}) \Leftarrow \mathsf{Q}_1(\overrightarrow{x_1}), \ldots, \mathsf{Q}_n(\overrightarrow{x_n}).$$

Execution in logic programming means to prove an existentially quantified predicate of the form

$$\exists \overrightarrow{x} \,.\, \mathsf{P}_1(\overrightarrow{x_1}), \ldots, \mathsf{P}_n(\overrightarrow{x_n})$$

under the given program. The proof method of SLD resolution [Kow74] can obtain all solutions constructively; so, these solutions can be considered as

output of the program. Operational semantics of SLD resolution and declarative semantics by fixed point construction are given and its correspondence is proved. A standard textbook of This semantic correspondence is treated in a standard textbook [Llo87].

1.1.4 Algebraic Specification Languages

Algebraic specification languages, the OBJ [GWMF96] family as a major representative, are based on many(or order)-sorted equational logic. Although algebraic specification languages are not programming languages, their ideas are very similar to functional and logic programming languages. A specification consists of a set of sorted universally quantified equations of the form

$$\forall x : \mathsf{nat} \,.\, f(x) = x + 1.$$

Under such a specification, a system of an algebraic specification language can evaluate a term as in functional programming and can prove properties expressed by universally quantified equations like

$$\forall x : \mathsf{nat} \,.\, f(x + 1) = f(x) + 1$$

by using term rewriting. Semantics are given by many(or order)-sorted algebras and, in the so-called initial algebra approach, the quotient term algebra is used as the standard semantics of specifications. If the specification is built from a confluent and terminating rewrite system, it is well-known that universally quantified equation are proved by term rewriting both terms into the same term.

1.1.5 First-order Functional-logic Programming Languages

Functional-logic programming languages are an integration of functional and logic programming. Examples of such languages include SLOG [Fri85],EQLOG [GM86b], K-LEAF [GLMP91], BABEL [MNRA92], ALF [Han90], Ev [HNN+94] and Curry [HKMN95]. For a recent overview of the field, see Hanus [Han94]. A first-order functional-logic program is given by equations of the same form as in functional programs, like

$$f(x) = x + 1.$$

Execution in functional-logic programming means to provide an equation called a *query* and to obtain values for variables contained in the query. For example, if we give the query

$$?-\mathsf{f}(\mathsf{x}) = 8$$

to a system of a functional-logic programming language, under the above program, the system returns the answer

$$\mathsf{x} \mapsto 7.$$

We stress that modern functional-logic languages are *not* just an ad-hoc combination of functional and logic programming languages like combining Lisp and PROLOG. Modern functional-logic languages have a unified operational treatment of the functional and logic part of the execution and a clear view of mathematical semantics. Next we discuss this in detail.

Two approaches towards the semantics of first-order functional-logic languages are known. One is based on term rewriting. In this approach, a program of a functional logic language is regarded as a conditional term rewriting system [Kap84, BK86] and a query is considered to be the question for convertibility between the two sides of equations in the query. To solve the query is to find substitutions that establish such convertibility. Conditional narrowing has been developed as a method for this purpose. Since conditional narrowing is a sound and complete procedure for establishing this convertibility [GM86a, MH94, IO94], the operational models of some existing functional logic languages are based on refined versions of conditional narrowing, e.g., ALF and Ev.

The other approach is to extend the declarative semantics of logic programming languages [Llo87]. This was first described by Levi et al. [LPB+87] for their logic-plus-functional language K-LEAF. In this method, a program defines partial functions on the complete Herbrand universe, which contains infinite data structures, and solving a query means finding a substitution that is a solution for the query in the least complete Herbrand model. The languages BABEL [MNRA92] and SFL [JMMGMA92] also have semantics in line with this approach. Sound and complete operational models for these languages are given in [MNRA92] and [JMMGMA92].

1.1.6 Higher-order Functional-logic Programming Languages

Recently, higher-order extensions of functional-logic languages are investigated. The language SFL [GMHGRA92, JMMGMA92], which is a higher-order extension of BABEL, has the form of applicative equations for a program. Two semantics for SFL were proposed in [GMHGRA92]. The first one is the "declarative semantics" of SFL, which is essentially a semantics along the declarative approach in the semantics of first-order functional-logic languages where a slight difference is that each element in the Herbrand universe is an applicative form. The other is the "denotational semantics" of SFL, which is essentially also a declarative approach, but the Herbrand universe is extended to the universe that contains function spaces, because functions are used as data in higher-order programming.

A rewriting approach to the semantics of higher-order functional-logic language is given in [NMI95]. In this approach, a program is an applicative term rewriting system and the presented narrowing calculus for higher-order functional-logic programming is shown to be sound and complete for proving convertibility generated from applicative term rewriting, which is actually first-order.

1.2 Our Approach

In this section, we describe our approach to the semantics of first-order, interactive first-order, and higher-order functional-logic languages by comparing existing approaches in functional, logic, and specification languages.

Our main viewpoint is conditional equational logic. We use conditional equational logic as a basic framework of semantics throughout this thesis. Conditional equational logic [GM87, Wec92] is a subset of first-order logic, where the equality symbol "=" is the only predicate symbol and formulae are conditional equations.

1.2.1 First-order FLP

Our approach to the semantics for first-order functional-logic programming is the following. A functional-logic program is a set of conditional equations interpreted as defining functions, and a query is an existentially quantified

equation. Solving the query, which is the execution of a functional-logic program means proving the validity of the existentially quantified equation by obtaining a witness with respect to the functional-logic program as the premise. Conditional narrowing is a complete proof method for valid queries. The two approaches to the semantics of functional-logic languages are formalized as two different models of a functional-logic program, which comprise a set of conditional equations; these two models are the quotient term model and the least complete Herbrand model. The validity of the query is shown to be equivalent in both models; in other words, the quotient term model and least complete Herbrand model for a functional-logic language are equivalent (Section 2.6).

We can also straightforwardly extend the approach to higher-order functional-logic programs. This view influences the semantics of other declarative languages, i.e. functional, logic, and algebraic specification languages. In addition, it will be relatively clear that this view actually include the semantics of existing semantics of declarative languages. To clarify it, in the rest of this subsection, we explain that our approach to semantics of functional-logic languages offers us to view functional-logic programming languages as an integration of functional, logic, and algebraic specification languages.

Functional-logic Programming as Functional Programming

A pure higher-order functional program and an evaluation of a expression can be considered as an applicative term rewriting system [Klo92, KKSdV96, vBSB93] and solving a restricted form of a query whose right-hand side is a existentially quantified variable, e.g.

$$\exists \mathsf{x} \,.\, \mathsf{f}\ 3 = \mathsf{x}.$$

Functional-logic Programming as Logic Programming

If predicates are regarded as boolean functions, a logic program is considered as a restricted form of a functional-logic program where all the equations are only definitions of boolean functions. Asking a query in logic programming can be done by asking an equational form of the query coinciding with the form of equations in functional-logic programming. Namely, a logic program and the query is expressed as the following functional-logic program

$$\mathsf{P_i}(\overrightarrow{x}) = \mathsf{true} \Leftarrow \mathsf{Q_1}(\overrightarrow{x_1}) = \mathsf{true}, \ldots, \mathsf{Q_n}(\overrightarrow{x_n}) = \mathsf{true}$$

and the query

$$\exists \overrightarrow{x_1}, \ldots, \overrightarrow{x_n} \,.\, \mathsf{P}_1(\overrightarrow{x_1}) = \mathsf{ture}, \ldots, \mathsf{P_n}(\overrightarrow{x_n}) = \mathsf{true}.$$

This view is not just an intuitive explanation, which is theoretically proved that functional-logic programs and their execution is exactly a superset of logic programs and their execution [SH97]. On the other hand, in a logical sense, conditional equational logic is a subset of Horn clause logic where the only predicate symbol is the equality symbol. As we will see above its inclusion between the two systems (conditional equational logic $\subseteq$ Horn clause logic) does not actually mean an inclusion of expressive power of two systems.

Functional-logic Programming as Algebraic Specification

Since a functional-logic program can be seen as a set of (restricted form of) universally quantified equations, it is just an algebraic specification. Therefore, we can apply initial algebra semantics to a functional-logic program. Although, usually, functional-logic programming is not equipped with the feature of proving a universally quantified equation, from the equational logic view, it is possible to prove the universally quantified equation by using term rewriting. Moreover in functional-logic programming, not only a universally quantified but also an *existentially* quantified equation like

$$\exists \mathsf{x} : \mathsf{nat} \,.\, \mathsf{f}(\mathsf{x}+1) = \mathsf{f}(\mathsf{x}) + \mathsf{x}$$

can be constructively proved by solving the query. So, we can regard this part as an extension of algebraic specification. Of course, since the form of algebraic specifications is wider than that of functional-logic programs (e.g. associativity like $\forall \mathsf{x}, \mathsf{y}, \mathsf{z} \,.\, \mathsf{x} + (\mathsf{y} + \mathsf{z}) = (\mathsf{x} + \mathsf{y}) + \mathsf{z}$ can be included in a specification), not all features of algebraic specification are contained in functional-logic programming.

1.2.2 Interactive FLP

As we saw in the preceding sections, functional-logic programming contain most features of *pure* functional, logic, and specification languages. However, they does not have the features like assignment or I/O provided by impure functional programming languages. In Chapter 3, we introduce such impure

features, which we call *interaction*, into first-order functional-logic programming. Since a functional-logic program is a pure functional program, one may attempt to introduce monadic or explicit state passing style into functional-logic programming as candidates for introduction of interaction. However we think that they do not directly fit to functional-logic programming due to the following reasons:

- Monadic style: The monad for side-effects [Mog88] for interaction requires higher-order types, namely lambda expressions are needed. But the functional-logic programming languages considered in this thesis do not have lambda expressions, even if they belong to higher-order functional-logic programming languages (Chapter 4). Hence, this monadic approach is not directly applicable tofunctional-logic programming.

- Explicit state passing style: This way is directly applicable to functional-logic programming languages. However, a type system which ensure the single thredness of the state passing parameter [BS93] is required. The relationship between narrowing and this kind of type systems has not been investigated yet. Moreover all functions having interaction must have an extra parameter for state and therefore programs become complicated.

We also note that both approaches of treating interaction are merely *programming styles*. There is no direct semantic characterization of the interaction part in both approaches. Actually, the semantics of a program having interaction is given by the traditional semantics of pure functional programs. These approaches indeed lead to the preservation of the properties of pure programs and the fact that the modification of the operational semantics is not needed for interaction. However, since the constructs expressing interaction are expanded into programs, they become complicated and understanding their meaning is difficult, especially in the monadic style.

The monadic style proposed by Wadler [Wad90] is based on Moggi's work on notions of computation in categorical framework using monads. But Wadler's approach does not exactly match the original idea of Moggi, i.e., in the monadic style, Moggi's semantics is *implemented* in pure functional programs. We think that this is the reason of some complication in the use of monads in functional programs.

Hence, in our approach, we proceed as follows: We go back to the original idea of Moggi's formalization of computations and follow it:

(i) We extend the syntax of the first-order functional-logic program language to the syntax of interactive first-order functional-logic programming language expressing sequential computations by introducing let-terms (Section 3.1).

(ii) We give a semantics of let-terms by using the monad for side-effects in a categorical framework (Section 3.2).

Moggi gave the deduction system of the computational metalanguage, which is an extension of many-sorted equational logic including let-terms, and showed the soundness and completeness with respect to the categorical semantics using monads. In our semantic formalization for interactive functional-logic programming, we also use this approach; here we use the computational metalanguage instead of many-sorted equational logic in first-order FLP.

Moggi did not give an operational semantics for the computational metalanguage. So our third step of introducing interaction into functional-logic programming is the following:

(iii) We give a translation from a set of axioms of the computational metalanguage into a conditional term rewriting system.

Then, we show its soundness by considering categorical semantics and proving the implication of the provability relations between the two systems.

By these steps we can give the following semantics of interactive functional-logic languages:

- axiomatic semantics – computational metalanguage,
- categorical semantics – monads, and
- rewriting semantics – conditional term rewriting systems (rewriting and narrowing).

Since an interactive functional-logic program is translated into a conditional term rewriting system and narrowing is a sound and complete solving method for the convertibility of conditional term rewriting systems, narrowing can be used as a sound and complete solving method for interactive functional-logic programming.

1.2.3 Higher-order FLP

The class of higher-order functional-logic programming languages treated in this thesis is called "simply-typed applicative functional-logic languages", which includes the higher-order functional-logic language SFL.

Our approach to the semantics for this class of higher-order functional-logic programming language is a further development of both SFL's "denotational" approach [GMHGRA92] and the applicative term rewriting approach [NMI95], which were discussed in Section 1.1.6. We treat a program as an applicative term rewriting system in the operational semantics, i.e., first-order conditional narrowing is used to solve a higher-order query. For the algebraic semantics, we present first the initial model in a class of all models of a program as the quotient of convertibility generated from applicative term rewriting. Next, we give another model, which we call "minimal applicative Herbrand model". The carrier of this model contains function spaces, which is similar to the "denotational semantics" of SFL. The important difference compared with their semantics is that our semantics is sound and complete for the operational model of narrowing, unlike SFL's "denotational semantics". This desirable result comes from the minimality of the applicative Herbrand model in our semantics defined in Section 4.4.

Chapter 2

Many-sorted First-order FLP

In this chapter, we give the following semantics of many-sorted first-order functional-logic programming language:

- axiomatic semantics [GM87],
- rewriting semantics [Hul80, SMI95]
- algebraic semantics:
  - the quotient term model [GTW75, GM87],
  - the least complete Herbrand model [LPB$^+$87],
- categorical semantics [Cro93].

These semantics, except for the least complete Herbrand model, are already proposed as semantics of equational logic. Since our approach is to define a functional-logic program as a particular form of axioms of equational logic, we point out that they also become semantics of FLP.

Then we discuss the relationship between models using the known results (axiomatic $\Leftrightarrow$ rewriting $\Leftrightarrow$ the quotient term model), prove a new result (the quotient term model $\Leftrightarrow$ the least complete Herbrand model) and point out the inclusion (categorical $\supseteq$ algebraic).

2.1 Many-sorted Conditional Equational Logic

First we give syntax and a deduction system of many-sorted conditional equational logic. The form of presentation of equational logic follows [Cro93].

Definition 2.1.1 (**Signature**)
Let S be a set of *sorts*. An S-sorted *signature* Σ is a set of *function symbols* together with an arity function assigning to each function symbol f a finite sequence $\alpha_1, \ldots, \alpha_n$ and β of elements of S. This is denoted by

$$f : \alpha_1, \ldots, \alpha_n \to \beta$$

where we say that f has *arity* n, the *source sorts* are $\alpha_1, \ldots, \alpha_n$ and the *target sort* is β. When $n = 0$, we write

$$f : \beta$$

and say that f is a *constant symbol*. A sort α is called *empty* if there is no constant symbol of the sort α in Σ.

Definition 2.1.2 (**Raw terms**)
The *raw terms* are given by the following grammar:

$$t ::= x \mid k \mid f(t_1, \ldots, t_n)$$

where x is a variable, k a constant symbol and f a function symbol of non-zero arity n in the signature Σ.

Definition 2.1.3 (**Typing**)
A *typing context*, usually written as $\Gamma, \Delta, \ldots$, is a finite sequence $(x_1{:}\alpha_1, \ldots, x_n{:}\alpha_n)$ of (variable,sort)-pairs, where x_i's are mutually different. If a typing context Γ is an empty sequence, we say the *null typing context*, denoted by $\varnothing$. We write $\Gamma, x{:}\alpha$ to indicate the result of extending Γ by assigning the sort α to a variable $x \notin \mathcal{V}(\Gamma)$. We say that a raw term t has a type α, denoted by $t : \alpha$, under a typing context Γ if $\Gamma \rhd t : \alpha$ is derivable from the following typing rules. Such a raw term t is called a *well-typed term* or simply a *term*, and the well-typed term under the typing context $\Gamma \rhd t : \alpha$ a *proved term*. We often use abbreviated notation "$s, t : \alpha$" to mean that $s : \alpha$ and $t : \alpha$. A term is *ground* if it does not contain any variable. A term is *linear* if there is no multiple occurrence of the same variable.

$$\text{(variable)} \quad \frac{}{\Gamma, x{:}\alpha \rhd x{:}\alpha}$$

$$\text{(constant symbol)} \quad \frac{}{\Gamma \rhd k{:}\alpha} \quad k{:}\alpha \in \Sigma$$

$$\text{(first-order}^1\text{ term)} \quad \frac{\Gamma \rhd t_1{:}\alpha_1, \ldots, \Gamma \rhd t_n{:}\alpha_n}{\Gamma \rhd f(t_1, \ldots, t_n) : \beta} \quad f : \alpha_1, \ldots, \alpha_n \to \beta \in \Sigma$$

Notation 2.1.4
The set of variables occurring in a syntactic object e is denoted by $\mathcal{V}(e)$. A sequence of syntactic objects $e_1, \ldots, e_n$ is often abbreviated as $\overrightarrow{e_i}$.

Remark 2.1.5
We use a slightly different treatment of terms from the usual treatment of terms in many-sorted (conditional) equational logic in the context of algebraic specification [GTW75, GTW76, GM85, GM87], namely, we do not assume that variables used in the language have pre-defined sorts. In other words, we do not assume a global variable set indexed by sorts. Instead, we always attach type information of variables (typing context) for terms, for example,

$$\mathsf{x} : \mathsf{Int}, \mathsf{y} : \mathsf{Int} \ \triangleright \ \mathsf{plus}(\mathsf{x},\mathsf{y}).$$

This style is widely used in the type theory [Mit96]. In a formula (equation), a typing contexts are also used as a *variable quantification*. This form of terms and equations in equational logic is suited for the following two reasons:

- to use standard categorical treatment of semantics of terms and equations [Cro93, Pit95], i.e. categorically this form is natural.
- to avoid the unsound deduction problem with respect to empty sort interpretation in many-sorted equational-logic [GM85].

Definition 2.1.6 (Σ-**algebra**)
Let S be a set of sorts and Σ an S-sorted signature. A Σ-*algebra* is a pair $(A, \Sigma_{\mathcal{A}})$, consisting of an S-indexed family $A = \{A^{\alpha} \mid \alpha \in S\}$ of carrier sets and a set $\Sigma_{\mathcal{A}} = \{f_{\mathcal{A}} : A^{\alpha_1} \times \ldots \times A^{\alpha_n} \to A^{\beta} \mid f : \alpha_1, \ldots, \alpha_n \to \beta \in \Sigma\}$ of operations.

Definition 2.1.7 (**Homomorphism**)
Let $\mathcal{A} = (A, \Sigma_{\mathcal{A}})$ and $\mathcal{B} = (B, \Sigma_{\mathcal{B}})$ be Σ-algebras. A homomorphism $\phi : \mathcal{A} \to \mathcal{B}$ is a S-indexed family of functions, $\phi^{\alpha} : A^{\alpha} \to B^{\alpha}$ for each sort α, such that for each function symbol $f : \alpha_1, \ldots, \alpha_n \to \alpha$

$$\phi(f_{\mathcal{A}}(a_1, \ldots, a_n)) = f_{\mathcal{B}}(\phi^{\alpha_1}(a_1), \ldots, \phi^{\alpha_n}(a_n)).$$

[1]The terminology "first-order" used in this thesis refers to first-order functions in the sense of functional programming, namely a function (symbol) which is over base types. Note that it is different from the terminology "first-order" in the sense of "first-order predicate logic". Later we introduce higher-order FLP, but equational logic used there is always first-order in logical sense, i.e. disallowing predicates on predicates. (Actually, "=" is the only predicate symbol used in this thesis).

Definition 2.1.8
The *term algebra* $\mathcal{T}_\Sigma(\Gamma)$ under the typing context Γ is defined as follows:
carrier:

$$\mathcal{T}_\Sigma(\Gamma)^\alpha = \{t \mid \Gamma \triangleright t : \alpha\} \quad \text{for each sort } \alpha \in S.$$

operation:

$$f_{\mathcal{T}_\Sigma(\Gamma)}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$$

for each $f : \alpha_1, \ldots, \alpha_n \to \beta \in \Sigma$ where $\Gamma \triangleright t_i : \alpha_i$ for each $i = 1, \ldots, n$. If Γ is the null typing context, we write $\mathcal{T}_\Sigma$.

It is well-known that the ground well-typed term algebra has the following important property, called *initiality* in all Σ-algebras [GTW75, GTW76]. This result induces the uniqueness of interpretation of a term in a Σ-algebra.

Theorem 2.1.9
The ground well-typed term algebra $\mathcal{T}_\Sigma$ is an *initial algebra* in the class of all Σ-algebras, i.e. for any Σ-algebra $\mathcal{A}$, there exists a unique homomorphism from $\mathcal{T}_\Sigma$ to $\mathcal{A}$, denoted by $\mathcal{A}[\![_]\!] : \mathcal{T}_\Sigma \to \mathcal{A}$.

Definition 2.1.10 (Assignment)
Let $\mathcal{A} = (A, \Sigma_\mathcal{A})$ be a Σ-algebra and Γ a typing context $(x_1 : \alpha_1, \ldots, x_n : \alpha_n)$. An *assignment* θ from Γ to $\mathcal{A}$, denoted by $\theta : \Gamma \to \mathcal{A}$, is an S-indexed family $\{\theta^\alpha : X^\alpha \to A^\alpha \mid \alpha \in S\}$ of functions where X^α is a set of all variables of a sort α occurring in Γ. Notice that if some carrier set A^α is the empty set, θ^α is undefined as a function. In this case, we say that the assignment θ is *undefined*. The assignment θ is uniquely extended to a homomorphism $\theta^\# : \mathcal{T}_\Sigma(\Gamma) \to \mathcal{A}$ as follows:

$$\begin{aligned} \theta^\#(x) &= \theta(x), \\ \theta^\#(k) &= k_\mathcal{A}, \\ \theta^\#(f(t_1, \ldots, t_n)) &= f_\mathcal{A}(\theta^\#(t_1), \ldots, \theta^\#(t_n)). \end{aligned}$$

A *substitution* is an assignment $\theta : \Gamma \to \mathcal{T}_\Sigma(\Gamma')$ to a term algebra $\mathcal{T}_\Sigma(\Gamma')$. We often write $t\theta$ instead of $\theta^\#(t)$ for the application of substitutions. A substitution σ is often presented as the set $\{x_1 \mapsto t_1, .\,., x_n \mapsto t_n\}$, where $\sigma(x_i) = t_i$ for $i = 1, .\,., n$. Let $\sigma, \tau : \Gamma \to \mathcal{T}_\Sigma(\Gamma')$. We write $\sigma \preccurlyeq \tau[\Gamma]$ if there exists a substitution $\rho : \Gamma' \to \mathcal{T}_\Sigma(\Gamma'')$ such that $\rho^\# \circ \sigma = \tau$.

Definition 2.1.11 (**Equations**)
Let $s{:}\alpha$, $t{:}\alpha$, $s_1{:}\beta_1$, $\cdots$, $s_n{:}\beta_n$, $t_1{:}\beta_1$, $t_n{:}\beta_n$ be terms under a typing context Γ. A formula of the form

$$\forall\Gamma(s = t : \alpha \Leftarrow s_1 = t_1{:}\beta_1, \cdots , s_n = t_n{:}\beta_n)$$

is called a *universally quantified conditional equation* or simply *conditional equation*. If $n = 0$, we simply write the formula as $\forall\Gamma(s = t : \alpha)$. If Γ is a null typing context, we call it a *ground (conditional) equation*, and we denote it by $\forall\varnothing(s = t \Leftarrow s_1 = t_1, \cdots , s_n = t_n)$ or simply omit the quantification as $(s = t \Leftarrow s_1 = t_1, \cdots , s_n = t_n)$. A formula of the form

$$\exists\Delta(s_1 = t_1 : \beta_1, \cdots , s_n = t_n : \beta_n)$$

is called an *existentially quantified equation*. The type information like "$:\alpha$" in equations an proved terms may be omitted if it is not important. We may write $\forall\Gamma$. e or $\exists\Delta$. e for quantified equations.

Definition 2.1.12 (**Conditional Equational Logic**)
Conditional equational logic is a deduction system consisting of the following rules. This system is the same as that described in [GM87]. Let $\mathcal{R}$ be a set of conditional equations called *axioms*. A *theorem* is an equation derived by

these rules.

$$\text{(axiom)}\quad \frac{\forall\Gamma(s_1\theta = t_1\theta : \alpha_1)\cdots\forall\Gamma(s_n\theta = t_n\theta : \alpha_n)}{\forall\Gamma(s\theta = t\theta : \alpha)}$$

for any axiom $\forall\Gamma'(s = t : \alpha \Leftarrow s_1 = t_1 : \alpha_1, \cdots, s_n = t_n : \alpha_n) \in \mathcal{R}$
and for any substitution $\theta : \Gamma' \to \mathcal{T}_\Sigma(\Gamma)$

$$\text{(reflexivity)}\quad \frac{\Gamma \ \triangleright \ t : \alpha}{\forall\Gamma(t = t : \alpha)}$$

$$\text{(symmetry)}\quad \frac{\forall\Gamma(s = t : \alpha)}{\forall\Gamma(t = s : \alpha)}$$

$$\text{(transitivity)}\quad \frac{\forall\Gamma(s = t : \alpha)\quad \forall\Gamma(t = u : \alpha)}{\forall\Gamma(s = u : \alpha)}$$

$$\text{(permutation)}\quad \frac{\forall\Gamma(s = t : \alpha)}{\forall\Gamma'(s = t : \alpha)}\quad \text{where } \Gamma' \text{ is a permutation of } \Gamma$$

$$\text{(congruence)}\quad \frac{\forall\Gamma(s_1 = t_1 : \alpha_1)\cdots\forall\Gamma(s_n = t_n : \alpha_n)}{\forall\Gamma(f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n) : \beta)}\quad f : \alpha_1, \ldots, \alpha_n \to \beta \in \Sigma$$

$$\text{(existential introduction)}\quad \frac{\forall\varnothing(s_1\theta = t_1\theta : \alpha_1)\cdots\forall\varnothing(s_n\theta = t_n\theta : \alpha_n)}{\exists\Delta(s_1 = t_1 : \alpha_1, \cdots, s_n = t_n : \alpha_n)}$$

for any substitution $\theta : \Delta \to \mathcal{T}_\Sigma$

In the existential introduction rule, we assume $\mathcal{V}(\Delta) = \mathcal{V}(\overrightarrow{s_i = t_i})$. We write

$$\mathcal{R} \vdash e$$

for a theorem e under a set $\mathcal{R}$ of axioms .

2.2 Many-sorted Term Rewriting and Narrowing

We define term rewriting and narrowing for axioms by regarding them as left-to-right oriented conditional equations, i.e. *conditional term rewriting systems* [Kap84, BK86]. In this section, we give some preliminary definitions on term rewriting systems and known properties which are mainly related with equational logic. We use standard notations and terminology of term rewriting [DJ90]. Firstly, operations on terms are defined.

Definition 2.2.1
Let $\mathbb{N}_+$ be the set of positive naturals, and $\mathbb{N}_+^*$ the set of all sequences on $\mathbb{N}_+$,

denoted by $n_1 \cdot \ldots \cdot n_k$, where $n_1, \ldots, n_k \in \mathbb{N}_+$. The set $\mathcal{P}os(t)$ of *positions* in a term t is inductively defined as follows:

$$\mathcal{P}os : \mathcal{T}_\Sigma(\Gamma) \to \mathcal{P}(\mathbb{N}_+^*)$$

$$\mathcal{P}os(t) \stackrel{def}{=} \begin{cases} \{\epsilon\} & \text{if } t \text{ is a variable or constant symbol,} \\ \{\epsilon\} \cup \{i \cdot p | 1 \leq i \leq n \text{ and } p \in \mathcal{P}os(t_i)\} & \text{if } t \equiv f(t_1, \ldots, t_n). \end{cases}$$

The *subterm* of t at a position p, denoted by $t_{|p}$ is inductively defined as follows:

$$t_{|_} : \mathcal{T}_\Sigma(\Gamma) \times \mathbb{N}_+^* \to \mathcal{T}_\Sigma(\Gamma)$$

$$t_{|p} \stackrel{def}{=} \begin{cases} t & \text{if } p = \epsilon, \\ (t_i)_{|q} & \text{if } t \equiv f(t_1, \ldots, t_n) \text{ and } p = i \cdot q. \end{cases}$$

If $p \in \mathcal{P}os(t)$ then $t[s]_p$ denotes the term that is obtained from t by replacing the subterm at the position p by the term s.

Definition 2.2.2
A *context* C of sort α under a typing context Γ is a well-typed term containing a special constant $\Box^\beta$ of sort β. When $\Gamma' \triangleright t{:}\beta$, the term $C[t]$ denotes the well-typed term obtained from C by replacing the $\Box^\beta$ with t, and $\Gamma, \Gamma' \triangleright C[t] : \alpha$.

Definition 2.2.3
A *conditional term rewriting system* (CTRS for short) is a set of axioms if for each axiom $\forall\Gamma(l = r \Leftarrow \overrightarrow{s_i = t_i})$, l is not a variable. A CTRS is called *3-CTRS* [MH94] if $\mathcal{V}(l) \cup \mathcal{V}(\overrightarrow{s_i = t_i}) \supseteq \mathcal{V}(r)$ for each axiom $l = r \Leftarrow \overrightarrow{s_i = t_i}$. We often write an axiom of CTRSs as a *rewrite rule* $l \to r \Leftarrow \overrightarrow{s_i = t_i}$.

Definition 2.2.4
A rewrite relation $\to_\mathcal{R}$ on terms for a CTRS $\mathcal{R}$ is inductively defined as follows:

$$\begin{aligned} \to_{\mathcal{R}_0} &\stackrel{def}{=} \varnothing, \\ \to_{\mathcal{R}_{n+1}} &\stackrel{def}{=} \{ (C[l\theta], C[r\theta]) \mid \forall\Gamma(l \to r : \alpha \Leftarrow s_1 = t_1, ..., s_k = t_k) \in \mathcal{R}, \\ &\qquad \theta : \Gamma \to \mathcal{T}_\Sigma(\Delta),\ C \text{ is a context of sort } \alpha, \\ &\qquad s_i\theta \to_{\mathcal{R}_n}^* t_i\theta \text{ for every } i = 1, ..., k\}, \\ \to_\mathcal{R} &\stackrel{def}{=} \bigcup_{n \in \mathbb{N}} \to_{\mathcal{R}_n} . \end{aligned}$$

Here $\to^*_{\mathcal{R}_n}$ denotes the reflexive and transitive closure of $\to_{\mathcal{R}_n}$. The relation $\to^*_{\mathcal{R}}$ denotes the transitive-reflexive closure of $\to_{\mathcal{R}}$, $\to^+_{\mathcal{R}}$ denotes the transitive closure of $\to_{\mathcal{R}}$, and $\leftrightarrow^*_{\mathcal{R}}$, called *conversion*, denotes the transitive-reflexive-symmetric closure of $\to_{\mathcal{R}}$. We write $s \leftarrow_{\mathcal{R}} t$ if $t \to_{\mathcal{R}} s$.

A term s is a *normal form* with respect to $\mathcal{R}$ if there is no term t such that $s \to_{\mathcal{R}} t$. A substitution $\theta : (x_1 : \alpha_1, \ldots, x_n : \alpha_n) \to \mathcal{T}_\Sigma(\Gamma)$ is *normalized* (resp. *normalizable*) if for each $i = 1, \ldots, n$, $\theta(x_i)$ is (resp. has) a normal form. For a normalizable substitution θ, $\theta{\downarrow_{\mathcal{R}}}$ denotes the normalized substitution defined by $\theta{\downarrow_{\mathcal{R}}}\,(x) \stackrel{def}{=} a$ where $\theta(x) \to^!_{\mathcal{R}} a$.

We write $s \to^!_{\mathcal{R}} t$ if $s \to^*_{\mathcal{R}} t$ and t is a normal form. We write $s \downarrow_{\mathcal{R}} t$ if there exists a term u such that $s \to^*_{\mathcal{R}} u \leftarrow^*_{\mathcal{R}} t$. $\mathcal{R}$ is *terminating* if there is no infinite rewrite sequence $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \ldots$ and $\mathcal{R}$ is *confluent* if $s \leftrightarrow^*_{\mathcal{R}} t$ implies $s \downarrow_{\mathcal{R}} t$. $\mathcal{R}$ is *orthogonal* if

- left-hand sides of rules in $\mathcal{R}$ are linear, and
- for any two renamed versions of axioms $\forall X_1(l_1 = r_1 \Leftarrow c_1)$ and $\forall X_2(l_2 = r_2 \Leftarrow c_2)$ from $\mathcal{R}$ such that they have no variables in common, $l_{1|p}$ and l_2 are not unifiable for any non-variable position p in l_1, except in the case where $p = \epsilon$ and the two rules differ by a variable renaming.

Remark 2.2.5
The definition of $\to_{\mathcal{R}}$ in Definition 2.2.4 actually generates a rewrite relation on well-typed terms, i.e. it never generates a relation on non-well-typed terms. This is because the following substitution lemma holds [GM87]:

$$\text{(substitution)} \quad \frac{\forall \Gamma, x : \beta(s_1 = s_2 : \alpha) \quad \forall \Gamma(t_1 = t_2 : \beta)}{\forall \Gamma(s_1[t_1]_x = s_2[t_2]_x : \alpha)}$$

In the definition of $\to_{\mathcal{R}_{n+1}}$ in Definition 2.2.4, the well-typedness of the applications of substitution ($t\theta$) and context ($C[t]$) is guaranteed by this substitution lemma.

Definition 2.2.6
A CTRS $\mathcal{R}$ is called *properly oriented* [SMI95] if every $l \to r \Leftarrow \overrightarrow{s_i = t_i} \in \mathcal{R}$ with $\mathcal{V}(l) \not\supseteq \mathcal{V}(r)$ satisfies

$$\mathcal{V}(s_i) \subseteq \mathcal{V}(l) \cup \bigcup_{j=1}^{i-1} \mathcal{V}(t_j)$$

for all $i \in \{1, \ldots, n\}$. A CTRS $\mathcal{R}$ is called *right-stable* if every $l \to r \Leftarrow \overrightarrow{s_i = t_i} \in \mathcal{R}$ satisfies the following conditions:

$$(\mathcal{V}(l) \cup \bigcup_{j=1}^{i-1} \mathcal{V}(s_j = t_j) \cup \mathcal{V}(s_i)) \cap \mathcal{V}(t_i) = \varnothing$$

and t_i is either a linear constructor term (cf. Definition 2.4.1) or a ground normal form with respect to the system obtained by removing the conditions of rewrite rules, for every $i = 1, \ldots, n$.

In order to use term rewriting as an operational semantics of functional-logic programming, confluence of CTRSs is an indispensable property. A sufficient condition to guarantee confluence of 3-CTRSs is obtained by Suzuki *et al.*

Theorem 2.2.7 ([**SMI95**])
An orthogonal properly oriented right-stable 3-CTRS is confluent.

The conversion $\leftrightarrow^*_{\mathcal{R}}$ by rewriting and provable equality coincide. Yamada *et al.* showed that the convertibility of a class of CTRSs that contains orthogonal properly oriented right-stable 3-CTRSs corresponds to the deductive equality in one-sorted conditional equational logic. This result is straightforwardly extended to many-sorted case. We state their result in our setting.

Theorem 2.2.8 (**Logicality of CTRSs** [**YALSM97**])
Let $\mathcal{R}$ be an orthogonal properly oriented right-stable 3-CTRS. Then,

$$\mathcal{R} \vdash \forall \Gamma(s = t) \quad \Leftrightarrow \quad s \leftrightarrow^*_{\mathcal{R}} t$$

The next result is Herbrand's theorem for equational logic. At the end of this section, using this result, we discuss logical meaning of narrowing.

Theorem 2.2.9 (**Herbrand's Theorem for equational logic**)
Let $\mathcal{R}$ be a set of axioms.

$$\begin{aligned} & \mathcal{R} \vdash \exists \Delta(s_1 = t_1, \ldots, s_n = t_n) \\ \Leftrightarrow \quad & \mathcal{R} \vdash \forall \varnothing(s_1\theta = t_1\theta), \ldots, \mathcal{R} \vdash \forall \varnothing(s_n\theta = t_n\theta) \quad \text{for some } \theta : \Delta \to \mathcal{T}_\Sigma \end{aligned}$$

Proof Obvious from the (existential introduction) deduction rule in Definition 2.1.12. ■

We say that θ is an *answer substitution* for the existentially quantified equation $\exists\Delta(s_1 = t_1, \ldots, s_n = t_n)$ in the above Herbrand's theorem.

Remark 2.2.10
This form of Herbrand's theorem for equational logic does not require the nonempty sort assumption as in Goguen and Meseguar's one (Corollary 24 in [GM87]) because if there is a variable in Δ whose sort is empty the substitution $\theta : \Delta \to \mathcal{T}_\Sigma$ is undefined. Hence $\mathcal{R} \vdash \forall\varnothing(s_i\theta = t_i\theta)$ for each i and $\mathcal{R} \vdash \exists\Delta(s_1 = t_1, \ldots, s_n = t_n)$ has never been derivable.

Conditional narrowing is a proof method for existentially quantified equations by obtaining answer substitutions as the solutions. Since we define it as an operation on terms, we regard equations as terms in the narrowing process, namely we use a term $\mathsf{eq}(s,t)$ for an equation $s = t$ [MH94].

Definition 2.2.11 (**eq-terms**)
Let $\mathcal{R}$ be a CTRS over an S-sorted signature Σ. The CTRS $\mathcal{R}_{\mathsf{eq}}$ under the sort set S_{eq} and the S_{eq}-sorted signature Σ_{eq} is defined as follows:

- $S_{\mathsf{eq}} = S \cup \{\mathsf{Bool}\}$.
- $\Sigma_{\mathsf{eq}} = \Sigma \cup \{\mathsf{eq}^\alpha : \alpha, \alpha \to \mathsf{Bool} \mid \alpha \in S\} \cup \{\mathsf{true} : \mathsf{Bool}\}$.
- $\mathcal{R}_{\mathsf{eq}} = \mathcal{R} \cup \{\forall x : \alpha(\mathsf{eq}^\alpha(x,x) = \mathsf{true}) \mid \alpha \in S\}$.

The introduction of eq^α (we often omit the superscript α if it is clear from the context) and true does not essentially change the equational theory generated by $\mathcal{R}$.

Proposition 2.2.12
Let $\mathcal{R}$ be a confluent CTRS. Then, $\mathcal{R}_{\mathsf{eq}}$ is also confluent. And for any proved terms $\Gamma \rhd s : \alpha$ and $\Gamma \rhd t : \alpha$,

$$\mathcal{R} \vdash \forall\Gamma(s = t : \alpha) \quad \Leftrightarrow \quad \mathsf{eq}^\alpha(s,t) \to^*_{\mathcal{R}_{\mathsf{eq}}} \mathsf{true}.$$

Proof The preservation of confluence of $\mathcal{R}_{\mathsf{eq}}$ is due to modularity consideration; see Middeldorp [Mid90]. By confluence of $\mathcal{R}$, if $s \leftrightarrow^*_{\mathcal{R}_{\mathsf{eq}}} t$ then $s \downarrow_{\mathcal{R}_{\mathsf{eq}}} t$. Hence $\mathsf{eq}^\alpha(s,t) \to^*_{\mathcal{R}_{\mathsf{eq}}} \mathsf{true}$. Applying the logicality of CTRSs (Theorem 2.2.8), we have the proposition. ■

Definition 2.2.13 (**Conditional narrowing [MH94]**)
Let $\mathcal{R}_{\mathsf{eq}}$ be a CTRS and

- S_1, S_2 sequences of well-typed terms under a typing context Δ,
- s well-typed term under Δ, and
- T sequences of well-typed terms under Δ'.

Narrowing relation $\rightsquigarrow$ on sequences of well-typed terms is defined as follows:

$$S_1, s, S_2 \rightsquigarrow_{\theta,\mathcal{R}} T$$

if there exists

- a non-variable position $u \in \mathcal{P}os(s)$,
- a substitution $\theta : \Gamma', \Delta \to \mathcal{T}_\Sigma(\Delta')$
- an axiom $\forall\Gamma(l = r : \alpha \Leftarrow s_1 = t_1 : \beta_1, \cdots, s_n = t_n : \beta_n)$ in $\mathcal{R}$ (here we rename this $\forall\Gamma'(l' = r' : \alpha \Leftarrow s'_1 = t'_1 : \beta_1, \cdots, s'_n = t'_n : \beta_n)$ with respect to the variable names, where Δ and Γ' are disjoint)

such that

- θ is a most general unifier of $s_{|u}$ and l',
- $T = S_1\theta, (s[r']_u, \mathsf{eq}(s'_1, t'_1), \cdots, \mathsf{eq}(s'_n, t'_n))\theta, S_2\theta$.

We write $S_1 \rightsquigarrow^*_{\theta,\mathcal{R}} S_n$ if there exists a narrowing derivation $S_1 \rightsquigarrow_{\theta_1,\mathcal{R}} S_2 \rightsquigarrow_{\theta_2,\mathcal{R}} \cdots \rightsquigarrow_{\theta_{n-1},\mathcal{R}} S_n$ such that $\theta = \theta_{n-1} \circ \ldots \circ \theta_2 \circ \theta_1$, where the domains of substitutions are suitably extended for the composition. We often omit $\mathcal{R}$ in $\rightsquigarrow_{\theta,\mathcal{R}}$ if it is clear from the context.

Conditional narrowing defined above is sound and complete for rewriting in the sense presented below. We use the completeness result of conditional narrowing with respect to properly oriented right-stable 3-CTRSs obtained by Suzuki. We state his result below. We write $\top$ for a sequence consisting only of a finite number (possibly zero) of true's.

Theorem 2.2.14 (**Soundness [MH94] and completeness [Suz98] of conditional narrowing for rewriting**)
Let $\mathcal{R}$ be an orthogonal properly oriented right-stable 3-CTRS and $S =$

$s_1, \ldots, s_n$ be a sequence of well-typed terms under the typing context Δ. Conditional narrowing is sound for rewriting by $\mathcal{R}_{\mathsf{eq}}$, i.e.

$$\begin{aligned} & S \rightsquigarrow^*_\theta \top \quad \text{where } \theta : \Delta \to \mathcal{T}_\Sigma(\Gamma) \\ \Rightarrow \; & s_i\theta \to^*_{\mathcal{R}_{\mathsf{eq}}} \mathsf{true} \quad \text{for each } i = 1, \ldots, n. \end{aligned}$$

Moreover conditional narrowing is complete for rewriting: Let $\Delta \rhd s{:}\alpha$. For every *normalized* substitution[2] $\theta : \Delta \to \mathcal{T}_\Sigma(\Gamma)$,

$$s\theta \to^*_{\mathcal{R}_{\mathsf{eq}}} \mathsf{true} \quad \Rightarrow \quad s \rightsquigarrow^*_{\theta'} \mathsf{true}$$

where $\theta' : \Delta \to \mathcal{T}_\Sigma(\Gamma')$ such that $\theta' \preccurlyeq \theta[\Delta]$.

We will try to justify that narrowing is a sound and complete proof method for existentially quantified equations. Combining Herbrand Theorem for Equational Logic (Theorem 2.2.9) and Proposition 2.2.12, the provability $\mathcal{R} \vdash \exists\Delta(s_1 = t_1, \ldots, s_n = t_n)$ is equivalent to

$$\mathsf{eq}(s_1\theta, t_1\theta) \to^!_{\mathcal{R}_{\mathsf{eq}}} \mathsf{true}, \ldots, \mathsf{eq}(s_n\theta, t_n\theta) \to^!_{\mathcal{R}_{\mathsf{eq}}} \mathsf{true}$$

for some ground substitution $\theta : \Delta \to \mathcal{T}_\Sigma$. But we cannot immediately apply the completeness theorem of narrowing (Theorem 2.2.14), i.e. for each $i = 1, \ldots, n$,

$$\mathsf{eq}(s_i\theta, t_i\theta) \to^!_{\mathcal{R}_{\mathsf{eq}}} \mathsf{true} \quad \Rightarrow \quad \mathsf{eq}(s_i, t_i) \rightsquigarrow^*_{\theta'} \mathsf{true}$$

where some $\theta' : \Delta \to \mathcal{T}_\Sigma(\Gamma)$ such that $\theta' \preccurlyeq \theta[\Delta]$. Because to apply the completeness theorem of narrowing, the answer substitution θ in $\mathsf{eq}(s_i\theta, t_i\theta) \to^!_{\mathcal{R}_{\mathsf{eq}}} \mathsf{true}$ must be *normalized*. If the CTRS $\mathcal{R}_{\mathsf{eq}}$ is non-terminating[3], we cannot directly conclude the existence of a normalized answer substitution even if $\mathcal{R} \vdash \exists\Delta(s_1 = t_1, \ldots, s_n = t_n)$. However, the existence of a normalized answer substitution for a particular class of provable existentially quantified equations can be shown without assuming the termination of $\mathcal{R}_{\mathsf{eq}}$. This problem is solved in Section 2.6 by considering algebraic models of FLPs presented in Section 2.3 and 2.5.

[2] Actually Suzuki state this completeness with respect to sufficiently normali*zable* substitutions [Suz98]. The completeness with respect to normali*zed* substitutions is an immediate consequence.

[3] Non-termination of a CTRS is positively used as "lazy evaluation" feature of functional-logic programming, so we do not assume the termination of FLP.

2.3 Algebraic Semantics

In this section, we recall algebraic semantics of many-sorted conditional equational logic. For more detail, see [GM85, MG85, Wec92]. We present a model of a set of axioms $\mathcal{R}$ constructed syntactically and review its property.

Definition 2.3.1 (**Quotient term algebra**)
Let $\mathcal{R}$ be a set of axioms and Γ a typing context. The equivalence relation $=_{\mathcal{R},\Gamma}$ on well-typed terms is defined by $s =_{\mathcal{R},\Gamma} t$ if $\mathcal{R} \vdash \forall\Gamma(s = t)$. The quotient term algebra $\mathcal{T}_\Sigma(\Gamma)/=_{\mathcal{R},\Gamma}$ is defined as follows:
carrier:

$$\mathcal{T}_\Sigma(\Gamma)^\alpha/=_{\mathcal{R},\Gamma} = \{t \mid \Gamma \rhd t : \alpha\}/=_{\mathcal{R},\Gamma} \quad \text{for each sort } \alpha \in S.$$

operation:

$$f_{\mathcal{T}_\Sigma(\Gamma)/=_{\mathcal{R},\Gamma}}([t_1], \ldots, [t_n]) = [f(t_1, \ldots, t_n)]$$

for each function symbol $f : \alpha_1, \ldots, \alpha_n \to \alpha$. Here $[t]$ denotes the $=_{\mathcal{R},\Gamma}$-equivalence class of a well-typed term t.

We use the following notations.

Definition 2.3.2
Let $\mathcal{R}$ be a set of axioms and $\mathcal{A}$ a Σ-algebra.

(i) We write

$$\mathcal{A} \models \forall\Gamma(s = t \Leftarrow \overrightarrow{s_i = t_i})$$

if for all assignments $\theta : \Gamma \to \mathcal{A}$, $\theta^\#(s) = \theta^\#(t)$ whenever $\theta^\#(s_i) = \theta^\#(t_i)$ for all $i = 1, \cdots, n$. Then we say that $\forall\Gamma(s = t \Leftarrow s_1 = t_1, \cdots, s_n = t_n)$ is *valid* in a Σ-algebra $\mathcal{A}$. Note that all assignments $\theta : \Gamma \to \mathcal{A}$ in this definition are undefined (c.f. Definition 2.1.10), this conditional equation is valid.

(ii) We write

$$\mathcal{A} \models \exists\Gamma(\overrightarrow{s_i = t_i})$$

if there exists an assignment $\theta : \Gamma \to \mathcal{A}$, called *witness*, such that $\theta^\#(s_1) = \theta^\#(t_1), \ldots, \theta^\#(s_n) = \theta^\#(t_n)$. Then we say that $\mathcal{A}$ is a *model* of the equation, or $\exists\Gamma(\overrightarrow{s_i = t_i})$ is *valid* in $\mathcal{A}$.

(iii) We write

$$\mathcal{A} \models \mathcal{R}$$

if for all conditional equations $\forall\Gamma(s = t \Leftarrow s_1 = t_1, \cdots, s_n = t_n)$ in $\mathcal{R}$, $\mathcal{A} \models \forall\Gamma(s = t \Leftarrow s_1 = t_1, \cdots, s_n = t_n)$. Then we say that $\mathcal{A}$ is a model of $\mathcal{R}$.

(iv) The set $\mathcal{M}od(\mathcal{R})$ of all models of $\mathcal{R}$ is defined as follows:

$$\mathcal{M}od(\mathcal{R}) \stackrel{def}{=} \{\mathcal{A} \mid \mathcal{A} \models \mathcal{R}\}.$$

(v) Let e be a universally or existentially quantified equation. We write

$$\mathcal{R} \models e$$

if for all Σ-algebras $\mathcal{A} \in \mathcal{M}od(\mathcal{R}), \mathcal{A} \models e$.

Soundness and completeness are stated below.

Theorem 2.3.3 (**Soundness and completeness of many-sorted conditional equational logic [GM87]**)
Let $\mathcal{R}$ be a set of axioms. Then $\mathcal{T}_\Sigma(\Gamma)/=_{\mathcal{R},\Gamma} \in \mathcal{M}od(\mathcal{R})$ and

$$\mathcal{R} \vdash \forall\Gamma(s = t) \;\Leftrightarrow\; \mathcal{T}_\Sigma(\Gamma)/=_{\mathcal{R},\Gamma} \models \forall\Gamma(s = t) \;\Leftrightarrow\; \mathcal{R} \models \forall\Gamma(s = t).$$

Especially, in case of $\Gamma = \varnothing$, we obtain the following. We define $\mathcal{Q}_\mathcal{R} \stackrel{def}{=} \mathcal{T}_\Sigma(\varnothing)/=_{\mathcal{R},\varnothing}$.

Corollary 2.3.4
Let $\mathcal{R}$ be a set of axioms. Then $\mathcal{Q}_\mathcal{R} \in \mathcal{M}od(\mathcal{R})$. We call $\mathcal{Q}_\mathcal{R}$ the *quotient term model*. And

$$\mathcal{R} \vdash \forall\varnothing(s = t) \;\Leftrightarrow\; \mathcal{Q}_\mathcal{R} \models \forall\varnothing(s = t) \;\Leftrightarrow\; \mathcal{R} \models \forall\varnothing(s = t).$$

Moreover, the model $\mathcal{Q}_\mathcal{R}$ has initiality in the all models $\mathcal{M}od(\mathcal{R})$ of $\mathcal{R}$.

Proposition 2.3.5 (**Initiality of the quotient term model**)
For any Σ-algebra $\mathcal{A} \in \mathcal{M}od(\mathcal{R})$, there exists a unique homomorphism $\phi : \mathcal{Q}_\mathcal{R} \to \mathcal{A}$.

Likewise a similar result holds for existentially quantified equations [GM87].

Theorem 2.3.6
Let $\mathcal{R}$ be axioms. Then

$$\mathcal{R} \vdash \exists\Delta(\overrightarrow{s_i = t_i}) \;\Leftrightarrow\; \mathcal{R} \models \exists\Delta(\overrightarrow{s_i = t_i}) \;\Leftrightarrow\; \mathcal{Q}_\mathcal{R} \models \exists\Delta(\overrightarrow{s_i = t_i}).$$

2.4 Many-sorted First-order FLP in Equational Logic Framework

In this section, we define the syntax of FLP and its solving method in the framework of conditional equational logic. Then the quotient term model $\mathcal{Q}_{\mathcal{R}}$ is immediately a model of an FLP $\mathcal{R}$.

Definition 2.4.1 (**Signature**)
A many-sorted first-order FLP signature (FLP signature, for short) Σ is an S-sorted signature which is divided into two disjoint sets CON and FUN, where CON is a set of *constructor symbols* and FUN is a set of *defined function symbols*. A defined function symbol is called a *function symbol* for short, hereafter. A term built from constructors and variables is called a *constructor term*. Moreover the FLP signature must satisfies the following.

- The sort set S contains the sort Bool.
- The signature Σ contains the following symbols:
  - $\mathsf{steq}^{\alpha} : \alpha, \alpha \to \mathsf{Bool} \in$ FUN,
  - $\&\ : \mathsf{Bool}, \mathsf{Bool} \to \mathsf{Bool} \in$ FUN,
  - $\mathsf{true} : \mathsf{Bool} \in$ CON.

Definition 2.4.2 (**Strict equality**)
Let Σ be a FLP signature. The set **STEQ** of axioms is the following.

$$\begin{aligned}
&\forall \varnothing(\mathsf{steq}^{\alpha}(c,c) = \mathsf{true}) \text{ for each } c : \alpha \in \text{CON},\\
&\forall \overrightarrow{x_i : \alpha_i}, \overrightarrow{y_i : \alpha_i}(\mathsf{steq}^{\alpha}(d(x_1, \cdots, x_n), d(y_1, \cdots, y_n))\\
&\qquad = \mathsf{steq}^{\alpha_1}(x_1, y_1)\ \&\ \cdots\ \&\ \mathsf{steq}^{\alpha_n}(x_n, y_n))\\
&\qquad \text{for each } d : \alpha_1, \ldots, \alpha_n \to \alpha \in \text{CON},\\
&\qquad\qquad \forall x : \mathsf{Bool}(\mathsf{true}\ \&\ x = x).
\end{aligned}$$

We call an equation of the form $\mathsf{steq}^{\alpha}(s,t) = \mathsf{true}$ a *strict equation* (we often omit superscript α), where terms s and t do not contain any occurrence of steq^{α}.

Definition 2.4.3 (**FLP**)
A set $\mathcal{R}$ of axioms is called a *many-sorted first-order FLP* or simply a *first-order FLP* if $\mathcal{R}$ satisfies the following:

(i) $\mathcal{R}$ is built from a many-sorted first-order FLP signature.

(ii) $\mathcal{R}$ is a properly-oriented orthogonal 3-CTRS.

(iii) $\mathcal{R}$ contains the set **STEQ** of axioms.

And each equation $\forall\Gamma(l = r \Leftarrow \overrightarrow{s_i = t_i})$ in $\mathcal{R}$ satisfies the following:

(i) l is the form $f(t_1, \cdots, t_n)$ where $f \in \text{FUN}$ and $t_1, \cdots, t_n$ are constructor terms.

(ii) All the equations $s_1 = t_1, \ldots, s_n = t_n$ are strict equations.

Note that any functional-logic program is confluent because a CTRS satisfying the above conditions is an orthogonal properly oriented right-stable 3-CTRS (Theorem 2.2.7)[SMI95].

Definition 2.4.4 (**Query**)
Let $\mathcal{R}$ be a first-order FLP. A *query* of FLP is an existentially quantified equation of the form

$$\exists\Delta(\overrightarrow{\mathsf{steq}(s_i, t_i) = \mathsf{true}}).$$

Execution of FLP means to prove the query under the first-order FLP $\mathcal{R}$ as axioms by obtaining answer substitutions, i.e.

$$\mathcal{R} \vdash \exists\Delta(\overrightarrow{\mathsf{steq}(s_i, t_i) = \mathsf{true}}).$$

Let $\mathcal{R}$ be a first-order FLP. Now clearly we have

$$\mathcal{R}_{\mathsf{eq}} \vdash \forall\Gamma(\mathsf{steq}(s, t) = \mathsf{true}) \quad \Leftrightarrow \quad \mathcal{R}_{\mathsf{eq}} \vdash \forall\Gamma(\mathsf{eq}(\mathsf{steq}(s, t), \mathsf{true}) = \mathsf{true}).$$

We can specialize the definition of narrowing (Definition 2.2.13), which is on eq-terms, to steq-terms for solving queries consisting of strict equations as follows:

Definition 2.4.5 (**Conditional narrowing for strict equations**)
Let $\mathcal{R}_{\mathsf{eq}}$ be a first-order FLP and S_1, S_2, T be sequences of strict equations or true's. The narrowing relation $\rightsquigarrow$ on sequences of well-typed terms is defined as follows: $S_1, s, S_2 \rightsquigarrow_\theta T$ if there exists a non-variable position $u \in \mathcal{P}os(s)$, a new variant (with respect to variable-renaming) $\forall\Gamma(l = r \Leftarrow \mathsf{steq}(s_1, t_1) = \mathsf{true}, \cdots, \mathsf{steq}(s_n, t_n) = \mathsf{true})$ of a conditional equation in $\mathcal{R}$, and a substitution θ such that

- θ is a most general unifier of $s_{|u}$ and l,
- $T = S_1, (s[r]_u, \mathsf{steq}(s_1, t_1), \cdots, \mathsf{steq}(s_n, t_n))\theta, S_2$.

We write $S_1 \leadsto^*_\theta S_n$ if there exists a narrowing derivation $S_1 \leadsto_{\theta_1} S_2 \leadsto_{\theta_2} \cdots \leadsto_{\theta_{n-1}} S_n$ such that $\theta = \theta_{n-1} \circ \ldots \circ \theta_2 \circ \theta_1$.

Clearly this change does not affect the soundness and completeness of narrowing for queries. We state this as a corollary to the combination of Theorem 2.2.9 and 2.2.14.

Corollary 2.4.6
Let $\mathcal{R}$ be a first-order FLP.

$$\begin{aligned} & \mathcal{R} \vdash \exists\Delta(\overrightarrow{\mathsf{steq}(s_i, t_i) = \mathsf{true}}) \\ & \quad \text{with a normalized answer substitution } \theta : \Delta \to \mathcal{T}_\Sigma \\ \Leftrightarrow\ & \mathsf{steq}(s_1, t_1), \ldots, \mathsf{steq}(s_n, t_n) \leadsto^*_{\theta'} \top \ \text{ where } \theta' \preccurlyeq \theta[\Delta]. \end{aligned}$$

Since a first-order FLP is a (particular) set of axioms, we can always construct the quotient term model $\mathcal{Q}_\mathcal{R}$ for any first-order FLP $\mathcal{R}$.

Corollary 2.4.7
Let $\mathcal{R}$ be a first-order FLP. Then $\mathcal{Q}_\mathcal{R} \in \mathcal{M}od(\mathcal{R})$.

We can also immediately get the following correspondence between $\mathcal{Q}_\mathcal{R}$ and narrowing from Theorem 2.3.6 and Corollary 2.4.6.

Corollary 2.4.8
Let $\mathcal{R}$ be a first-order FLP.

$$\begin{aligned} & \mathcal{Q}_\mathcal{R} \models \exists\Delta \,.\, \overrightarrow{\mathsf{steq}(s_i, t_i) = \mathsf{true}} \\ & \quad \text{with a normalized answer substitution } \theta : \Delta \to \mathcal{T}_\Sigma \\ \Leftrightarrow\ & \mathsf{steq}(s_1, t_1), \ldots, \mathsf{steq}(s_n, t_n) \leadsto^*_{\theta'} \top \ \text{ where } \theta' \preccurlyeq \theta[\Delta]. \end{aligned}$$

2.5 Least Complete Herbrand Model

Levi *et al.* [LPB+87, GLMP91] introduced a cpo model for their single-sorted logic plus functional language K-LEAF, called the least complete Herbrand model. In this section we give a many-sorted version of this model by following their construction.

We first give some preliminary definitions on algebraic cpos. Other basic notions, results on cpos and domain theoretic semantics of programming languages will also be used without comment; see, for example, [Gun92].

A partially ordered set, with order $\sqsubseteq$, is called *cpo* if it has a least element $\bot$ and any directed subset S of D has a least upper bound $\bigsqcup S$ in D. Let D be a cpo. An element z is called *compact* if for any directed subset S of D, $z \sqsubseteq \bigsqcup S$ implies that there exists an element $y \in S$ such that $z \sqsubseteq y$. An element x is called *total* if the only upper bound of x in D is x itself. D is *algebraic* if for all $x \in D$, the set $S_x = \{z \mid z \sqsubseteq x \text{ and } z \text{ is compact}\}$ is directed and $x = \bigsqcup S_x$.

We extend the signature CON to include bottom elements $\bot^\alpha$ for each sort α, i.e. $\text{CON}_\bot \stackrel{def}{=} \text{CON} \cup \{\bot^\alpha : \alpha \mid \alpha \in S\}$. For a partial function $f : D \to E$, $\mathsf{def}(f) = \{d \mid (d, e) \in f\}$ is the domain of defined points of f.

Definition 2.5.1 (**Tree**)
A $\text{CON}_\bot$-*tree* is a partial function $t : \mathbb{N}_+^* \to \text{CON}_\bot$ such that, for all $u \in \mathbb{N}_+^*$ and $i \in \mathbb{N}_+$,

(i) if $u \cdot i \in \mathsf{def}(t)$ then $u \in \mathsf{def}(t)$,

(ii) for any $u \in \mathsf{def}(t)$, $t(u)$ is a constructor symbol $c : \alpha_1, \ldots, \alpha_n \to \beta$ iff $u \cdot i \in \mathsf{def}(t)$ and the target sort of $t(u \cdot i)$ is α_i for each $i = 1, \ldots, n$,

(iii) if $u \cdot n \in \mathsf{def}(t)$ then $u \cdot 1, \ldots, u \cdot n \in \mathsf{def}(t)$.

Definition 2.5.2
Let α be a sort. The *complete Herbrand universe* H^α is defined as

$$\text{H}^\alpha \stackrel{def}{=} \{t \mid t \text{ is a } \text{CON}_\bot\text{-tree such that the target sort of } t(\epsilon) \text{ is } \alpha\}$$

where ϵ denotes the empty sequence of $\mathbb{N}_+^*$, so $t(\epsilon)$ denotes the root symbol of a tree t.

Definition 2.5.3
Let $\mathcal{A}$ be a complete Herbrand algebra. For each $\alpha \in S$ the order $\sqsubseteq_{\text{H}^\alpha}$ on H^α is defined as:

$$s \sqsubseteq_{\text{H}^\alpha} t \iff \tilde{s}(x) \sqsubseteq \tilde{t}(x) \quad \text{for all } x \in \mathbb{N}_+^*$$

where

$$\tilde{s}(x) \stackrel{def}{=} \begin{cases} \bot^{\alpha} & \text{if } s(x) \text{ is undefined} \\ s(x) : \alpha & \text{otherwise} \end{cases}$$

Proposition 2.5.4
For each $\alpha \in S$, $(\mathrm{H}^{\alpha}, \sqsubseteq_{\mathrm{H}^{\alpha}})$ is an algebraic cpo.
Proof Because the continuous function spaces on an algebraic cpo form an algebraic cpo [Gun92]. ■

Moreover, the cpo $(\mathrm{H}^{\alpha}, \sqsubseteq)$ has the following properties [MNRA92]:

(i) The compact elements of H^{α} are the $\textsc{Con}_{\bot}$-trees having a finite number of nodes.

(ii) The total elements of H^{α} are the $\textsc{Con}_{\bot}$-trees without any occurrence of $\bot^{\alpha}$.

The set of ground constructor terms of a sort α, denoted by D^{α}, is a proper subset of the complete Herbrand universe H^{α} by identifying them with the compact and total elements of H^{α}, i.e. finite trees without any occurrence of $\bot^{\alpha}$.

Definition 2.5.5 (**Complete Herbrand algebra**)
A Σ-algebra $\mathcal{A}$ is called *complete Herbrand algebra* if it satisfies the following:
carrier:

$$\mathrm{H}^{\alpha} \quad \text{for each sort } \alpha.$$

operations:

$$c_{\mathcal{A}} : \mathrm{H}^{\alpha_1} \times \cdots \times \mathrm{H}^{\alpha_n} \to \mathrm{H}^{\alpha},$$
$$c_{\mathcal{A}}(t_1, \ldots, t_n) = c(t_1, \ldots, t_n)$$

for each constructor symbol $c : \alpha_1, \ldots, \alpha_n \to \alpha \in \textsc{Con}$. Note that the right-hand side of the above definition actually denotes a $\textsc{Con}^{\alpha}_{\bot}$-tree. And for each defined function symbol $f : \alpha_1, \ldots, \alpha_n \to \alpha \in \textsc{Fun}$,

$$f_{\mathcal{A}} : \mathrm{H}^{\alpha_1} \times \cdots \times \mathrm{H}^{\alpha_n} \to \mathrm{H}^{\alpha},$$
$$f_{\mathcal{A}} \text{ is a continuous function.}$$

For both cases, if $n = 0$ then these are constant functions returning an element of H^{α}.

Definition 2.5.6
Define $\mathcal{H}erb\mathcal{A}lg$ as the class of all complete Herbrand algebras and the order $\sqsubseteq_{\mathcal{H}erb\mathcal{A}lg}$ on $\mathcal{H}erb\mathcal{A}lg$ is defined as:

$$\mathcal{A} \sqsubseteq_{\mathcal{H}erb\mathcal{A}lg} \mathcal{B} \overset{def}{\iff} f_{\mathcal{A}}(a_1, \ldots, a_n) \sqsubseteq_{\mathrm{H}^\alpha} f_{\mathcal{B}}(a_1, \ldots, a_n)$$

for each $f : \alpha_1, \ldots, \alpha_n \to \alpha \in \Sigma, a_1 \in \mathrm{H}^{\alpha_1}, \ldots, a_n \in \mathrm{H}^{\alpha_n}$.

Proposition 2.5.7
$(\mathcal{H}erb\mathcal{A}lg, \sqsubseteq_{\mathcal{H}erb\mathcal{A}lg})$ is an algebraic cpo.

Definition 2.5.8
An operator $\mathrm{T}_{\mathcal{R}} : \mathcal{H}erb\mathcal{A}lg \to \mathcal{H}erb\mathcal{A}lg$ is defined as follows:

$$\mathrm{T}_{\mathcal{R}}(\mathcal{A}) \overset{def}{=} (\{\mathrm{H}^\alpha \mid \alpha \in S\}, \{c_{\mathcal{A}} \mid c \in \textsc{Con}\} \cup \{\mathrm{S}_{\mathcal{R}}(\mathcal{A}, f) \mid f \in \textsc{Fun}\}),$$

where for each $f : \alpha_1, \ldots, \alpha_n \to \alpha \in \textsc{Fun}$ and $\mathcal{A} \in \mathcal{H}erb\mathcal{A}lg$, $\mathrm{S}_{\mathcal{R}}(\mathcal{A}, f) : \mathrm{H}^n \to \mathrm{H}$ is defined as follows:

$$\mathrm{S}_{\mathcal{R}}(\mathcal{A}, f)(h_1, \cdots, h_n) \overset{def}{=} \begin{cases} \theta^{\#}(r) & \text{if there exist } \forall\Gamma(f(l_1, \cdots, l_n) = r \\ & \qquad \Leftarrow s_1 = t_1, \cdots, s_m = t_m) \in \mathcal{R} \\ & \quad \text{and } \theta : X \to \mathcal{A}, \\ & \quad h_i = \theta^{\#}(l_i) \text{ for every } i \in \{1, \cdots, n\}, \\ & \quad \theta^{\#}(s_i) = \theta^{\#}(t_i) \text{ for every } i \in \{1, \cdots, m\}, \\ \bot^\alpha & \text{otherwise.} \end{cases}$$

The operator $\mathrm{T}_{\mathcal{R}}$ is a continuous function on $\mathcal{H}erb\mathcal{A}lg$. This can be proved by showing that each construct of $\mathrm{T}_{\mathcal{R}}$ is continuous (this is straightforward checking, but there are many tiresome things [GLMP91, Gun92]). By the fixpoint theorem on cpos, $\mathrm{T}_{\mathcal{R}}$ has the least fixpoint expressed as follows:

$$\mathcal{H}_{\mathcal{R}} \overset{def}{=} \bigsqcup_{i \in \mathbb{N}} {\mathrm{T}_{\mathcal{R}}}^i(\bot_{\mathcal{H}erb\mathcal{A}lg})$$

where $\bot_{\mathcal{H}erb\mathcal{A}lg}$ is the least element of $\mathcal{H}erb\mathcal{A}lg$.

$\mathcal{H}_{\mathcal{R}}$ is of course a complete Herbrand algebra and its operations are defined as a set of appropriate continuous functions corresponding to the program $\mathcal{R}$. Namely, we have the following.

Theorem 2.5.9
$\mathcal{H}_{\mathcal{R}}$ is the least (with respect to $\sqsubseteq_{\mathcal{H}erb\mathcal{A}lg}$) Herbrand model of a program $\mathcal{R}$.

In the model $\mathcal{H}_\mathcal{R}$, the program for the strict equality has the desired declarative meaning. More precisely, the meaning of the strict equality in $\mathcal{H}_\mathcal{R}$ is a function $\mathsf{steq}^\alpha_{\mathcal{H}_\mathcal{R}} : \mathrm{H}^\alpha \times \mathrm{H}^\alpha \to \mathrm{H}^{\mathsf{Bool}}$ that is characterized by the following propositions:

Proposition 2.5.10 ([**LPB$^+$87, Ham95**])
Let $u, v \in \mathrm{H}^\alpha$. Then,

(i) $\mathsf{steq}^\alpha_{\mathcal{H}_\mathcal{R}}(u, v) = \mathsf{true} \;\Leftrightarrow\; u =_{\mathrm{H}^\alpha} v$ and $u, v \in \mathrm{D}^\alpha$,

(ii) $\mathsf{steq}^\alpha_{\mathcal{H}_\mathcal{R}}(u, v) = \bot_\mathrm{H} \;\Leftrightarrow\; u \neq_{\mathrm{H}^\alpha} v$ or $u \notin \mathrm{D}^\alpha$ or $v \notin \mathrm{D}^\alpha$.

Clearly $\mathsf{steq}^\alpha_{\mathcal{H}_\mathcal{R}}$ is a strict function on both arguments, hence it is called "strict equality". Note that the binary relation $\mathsf{steq}^\alpha_{\mathcal{H}_\mathcal{R}}(-,-) = \mathsf{true}$ is not an equivalence relation on H^α because it is not reflexive, i.e. when $u \in \mathrm{H}^\alpha$ and $u \notin \mathrm{D}^\alpha$, $\mathsf{steq}^\alpha_{\mathcal{H}_\mathcal{R}}(u, u) = \bot$. This also affects syntactic level, namely a proved term for $\Gamma \rhd t : \alpha$,

$$\mathcal{R} \vdash \forall\Gamma(\mathsf{steq}(t, t) = \mathsf{true})$$

does not hold in general. This only holds when $t \to^!_\mathcal{R} d$ and d is a constructor term.

2.6 Equivalence of the Validity in Models

In Section 2.3 and 2.5, we saw that a first-order FLP has at least three models: the quotient term model $\mathcal{Q}_\mathcal{R}$, the operational model of narrowing and the least complete Herbrand model $\mathcal{H}_\mathcal{R}$. The correspondence between $\mathcal{Q}_\mathcal{R}$ and narrowing has been stated in Corollary 2.4.8. In this section, we show the correspondence between $\mathcal{Q}_\mathcal{R}$ and $\mathcal{H}_\mathcal{R}$, more precisely, the equivalence of the validity of a query in $\mathcal{Q}_\mathcal{R}$ and $\mathcal{H}_\mathcal{R}$.

Before proving the theorem, we will discuss why the form of query is restricted to strict equations. The following example shows that if we use a usual equation as query, then there are cases that the validity of a query is different in $\mathcal{Q}_\mathcal{R}$ and in $\mathcal{H}_\mathcal{R}$.

Example 2.6.1

Assume the following signature:

$$\mathsf{Nat}, \mathsf{NatList} \in S,$$
$$\mathsf{f} : \mathsf{Nat} \to \mathsf{Nat}, \mathsf{ones} : \mathsf{NatList},$$
$$\mathsf{ns} : \mathsf{Nat} \to \mathsf{NatList} \in \textsc{Fun}$$
$$1, 2, 3 : \mathsf{Nat}, \quad \text{“::”} \ : \mathsf{Nat}, \mathsf{NatList} \to \mathsf{NatList} \in \textsc{Con}$$

and a program

$$\mathcal{R} = \left\{ \begin{array}{lcl} \forall \varnothing(\mathsf{f}(1) & = & 1 : \mathsf{Nat}), \\ \forall \varnothing(\mathsf{ones} & = & 1 :: \mathsf{ones} : \mathsf{NatList}), \\ \forall n(\mathsf{ns}(n) & = & n :: \mathsf{ns}(n) : \mathsf{NatList}) \end{array} \right\}.$$

We can construct the models $\mathcal{Q}_\mathcal{R}$ and $\mathcal{H}_\mathcal{R}$ for $\mathcal{R}$. Consider the query $\exists\varnothing(\mathsf{f}(2) = \mathsf{f}(3))$. In $\mathcal{H}_\mathcal{R}$, by $\mathsf{f}_{\mathcal{H}_\mathcal{R}}(2) = \mathsf{f}_{\mathcal{H}_\mathcal{R}}(3) = \bot_{\text{H}}$, the query is valid:

$$\mathcal{H}_\mathcal{R} \models \exists\varnothing(\mathsf{f}(2) = \mathsf{f}(3) : \mathsf{Nat}).$$

But in $\mathcal{Q}_\mathcal{R}$, by $f_{\mathcal{Q}_\mathcal{R}}([2]) = [\mathsf{f}(2)] = \{\mathsf{f}(2)\}$ and $f_{\mathcal{Q}_\mathcal{R}}([3]) = [\mathsf{f}(3)] = \{\mathsf{f}(3)\}$, we have

$$\mathcal{Q}_\mathcal{R} \not\models \exists\varnothing(\mathsf{f}(2) = \mathsf{f}(3) : \mathsf{Nat}).$$

Next, we consider the query $\exists\varnothing(\mathsf{ones} = \mathsf{ns}(1))$. By $\mathsf{ones}_{\mathcal{H}_\mathcal{R}} = \mathsf{ns}_{\mathcal{H}_\mathcal{R}}(1) = 1 :: 1 :: 1 :: \cdots$ (infinite list of 1s), the following holds:

$$\mathcal{H}_\mathcal{R} \models \exists\varnothing(\mathsf{ones} = \mathsf{ns}(1) : \mathsf{NatList}).$$

But by $\mathsf{ones}_{\mathcal{Q}_\mathcal{R}} = [\mathsf{ones}] = \{\mathsf{ones}, 1 :: \mathsf{ones}, 1 :: 1 :: \mathsf{ones}, \cdots\}$ and $\mathsf{ns}_{\mathcal{Q}_\mathcal{R}}(1) = [\mathsf{ns}(1)] = \{\mathsf{ns}(1), 1 :: \mathsf{ns}(1), 1 :: 1 :: \mathsf{ns}(1), \cdots\}$, we have

$$\mathcal{Q}_\mathcal{R} \not\models \exists\varnothing(\mathsf{ones} = \mathsf{ns}(1) : \mathsf{NatList}).$$

Hence the validity of a query is different in $\mathcal{Q}_\mathcal{R}$ and in $\mathcal{H}_\mathcal{R}$ in these examples.

Since $\mathcal{Q}_\mathcal{R}$ is the initial model of $\mathcal{R}$ by Theorem 2.3.4 , the following holds:

$$\mathcal{Q}_\mathcal{R} \models \exists\Delta(s = t : \alpha) \quad \Rightarrow \quad \mathcal{H}_\mathcal{R} \models \exists\Delta(s = t : \alpha)$$

However, the above example shows that when the query is a usual equation, the reverse implication does not hold in general. The difference between the

validity of $\mathcal{Q}_\mathcal{R}$ and $\mathcal{H}_\mathcal{R}$ is caused by the fact that $\mathcal{Q}_\mathcal{R}$ does not express the notions of partial function and infinite data structure. From this fact, it may seem that $\mathcal{Q}_\mathcal{R}$ is inadequate as a model of a functional-logic program. But there is an important advantage of $\mathcal{Q}_\mathcal{R}$ over $\mathcal{H}_\mathcal{R}$, namely, having the soundness and completeness results between $\mathcal{Q}_\mathcal{R}$ and the operational model of narrowing (Theorem 2.2.14). Using narrowing, the validity of a query can be checked operationally. However, it is difficult to find a complete and computable operational model for $\mathcal{H}_\mathcal{R}$ like narrowing for $\mathcal{Q}_\mathcal{R}$. Such a complete operational model, which must find *all* solutions, may not exist, because it would constitute a procedure that is able to compute a witness that includes an infinite data structure. Therefore, if we use a general equation as a query, its validity cannot always be shown operationally.

Using strict equations solves this problem, because strict equality in $\mathcal{H}_\mathcal{R}$ is an equality function on constructor terms. Roughly speaking, since constructor terms are defined and finite data structure, the equality on undefined values and infinite data structures need not be considered by using the strict equality and the following equivalence can be shown:

$$\mathcal{Q}_\mathcal{R} \models \exists X(\mathsf{steq}(s,t) = \mathsf{true}) \quad \Leftrightarrow \quad \mathcal{H}_\mathcal{R} \models \exists X(\mathsf{steq}(s,t) = \mathsf{true}).$$

The above statement will be proved in Theorem 2.6.7. To prove the theorem, a few lemmas and propositions are needed.

Lemma 2.6.2
Let $E_1, \ldots, E_n, F$ be cpos, $f : E_1 \times \ldots \times E_n \to F$ be a continuous function and $D, D_1, \ldots, D_n$ be the sets of all compact and total elements of $F, E_1, \ldots, E_n$ respectively. Suppose e_i is a compact element of the cpo E_i for each $i = 1, \ldots, n$. If $f(e_1, \ldots, e_n) \in D$, then for all elements $d_i \in D_i$ with $d_i \sqsupseteq e_i$ for each $i = 1, \ldots n$, $f(d_1, \ldots, d_n) = f(e_1, \ldots, e_n)$ holds.
Proof Suppose e_i is a compact element of the cpo E_i for each $i = 1, \ldots, n$ such that $f(e_1, \ldots, e_n) \in D$. Take elements $d_i \in D_i$ such that $d_i \sqsupseteq e_i$ for $i = 1, \ldots n$. By the monotonicity of f,

$$f(d_1, \ldots, d_n) \sqsupseteq f(e_1, \ldots, e_n).$$

By $f(e_1, \ldots, e_n) \in D$, $f(e_1, \ldots, e_n)$ is a total element of the cpo F. So $f(d_1, \ldots, d_n) = f(e_1, \ldots, e_n)$.

In order to prove Proposition 2.6.6, we require the notion of derived operator [GTEJ77].

Definition 2.6.3 (**Derived operator**)
Let $\mathcal{A} = (A, \Sigma_{\mathcal{A}})$ be an Σ-algebra. Let $\Delta = (x_1 : \alpha_1, \cdots, x_n : \alpha_n)$ be a typing context and $\Delta \ \triangleright \ t : \alpha$ a proved term. The derived operator $t_{\mathcal{A}} : A^{\alpha_1} \times \ldots \times A^{\alpha_1} \to A^{\alpha}$ is defined as follows:

$$t_{\mathcal{A}}(a_1, \cdots, a_n) \stackrel{def}{=} \theta^{\#}(t)$$
where $\theta : \Delta \to \mathcal{A}$ is the assignment defined as:
$$\theta(x_i) \stackrel{def}{=} a_i \text{ for } i = 1, \cdots, n.$$

The following lemma is due to Goguen *et al.* [GTEJ77].

Lemma 2.6.4
Let $\mathcal{A} = (A, \Sigma_{\mathcal{A}})$ be an Σ-algebra such that A is a cpo, and for all $f \in \Sigma_{\mathcal{A}}$, f is a continuous function. Let $x_1 : \alpha_1, \cdots, x_n : \alpha_n \ \triangleright \ t : \alpha$. Then the derived operator $t_{\mathcal{A}} : A^{\alpha_1} \times \ldots \times A^{\alpha_n} \to A^{\alpha}$ is a continuous function.

The following lemma relating the denotation in $\mathcal{H}_{\mathcal{R}}$ and rewriting is proved by J. C. González-Moreno *et al.* (Lemma 5.2 of [GMHGRA92]). Actually, they showed this lemma in the applicative complete Herbrand algebra. The following lemma is a non-applicative (i.e. $\mathcal{H}_{\mathcal{R}}$) and many-sorted version of their lemma, which can be proved in the same way.

Lemma 2.6.5
Let $\mathcal{R}$ be a program, $\triangleright e : \alpha, t : \alpha$ ground well-typed terms such that $t \in \mathrm{D}^{\alpha}$. Then,

$$\mathcal{H}_{\mathcal{R}}[\![e]\!] = t \ \Rightarrow \ e \to^{!}_{\mathcal{R}} t.$$

(the notation $\mathcal{H}_{\mathcal{R}}[\![e]\!]$ denotes a interpretation of a term e in $\mathcal{H}_{\mathcal{R}}$, see Definition 2.1.9).

The next proposition is the key result for proving the equivalence of the validity of query in the two models.

Proposition 2.6.6
Let $\mathcal{R}$ be a program and $\exists\Delta(\mathsf{steq}(s,t) = \mathsf{true})$ a query. Then there exists a constructor term substitution $\theta : \Delta \to \mathcal{T}_{\Sigma}$ (i.e. the image of θ is a set of constructor terms) such that

$$\begin{aligned} & \mathcal{H}_{\mathcal{R}} \models \exists\Delta(\mathsf{steq}(s,t) = \mathsf{true}) \\ \Rightarrow \quad & \mathcal{Q}_{\mathcal{R}} \models \forall\varnothing(\mathsf{steq}(s,t)\theta = \mathsf{true}). \end{aligned}$$

Proof Suppose $\eta = \{x_1 \mapsto h_1, \cdots, x_n \mapsto h_n\}$ is a witness of $\exists\Delta(\mathsf{steq}(s,t) = \mathsf{true})$, where $h_1 \in \mathrm{H}^{\alpha_1}, \cdots, h_n \in \mathrm{H}^{\alpha_n}$ and $\Delta = (x_1 : \alpha_1, \cdots, x_n : \alpha_n)$, i.e. $\eta : \Delta \to \mathcal{H}_\mathcal{R}$ satisfies $\eta^\#(\mathsf{steq}(s,t)) = \mathsf{true}$. Then

$$\begin{array}{rl} & \eta^\#(\mathsf{steq}(s,t)) \\ = & \mathsf{steq}(s,t)_{\mathcal{H}_\mathcal{R}}(h_1, \cdots, h_n). \end{array}$$

Note that $\mathsf{steq}(s,t)_{\mathcal{H}_\mathcal{R}}$ is a derived operator on $\mathcal{H}_\mathcal{R}$. Since each H^{α_i} is an algebraic CPO, $h_i = \bigsqcup\{z_i \mid \text{ compact } z_i \sqsubseteq h_i\}$ for $i = 1, \cdots, n$. Furthermore,

$$\begin{array}{rl} & \mathsf{steq}(s,t)_{\mathcal{H}_\mathcal{R}}(h_1, \cdots, h_n) \\ = & \mathsf{steq}(s,t)_{\mathcal{H}_\mathcal{R}}(\bigsqcup\{z_1 \mid \text{ compact } z_1 \sqsubseteq h_1\}, \cdots, \bigsqcup\{z_n \mid \text{ compact } z_n \sqsubseteq h_n\}). \\ = & \bigsqcup\{\mathsf{steq}(s,t)_{\mathcal{H}_\mathcal{R}}(z_1, \cdots, z_n) \mid i = 1, \cdots, n, \text{ compact } z_i \sqsubseteq h_i\} \\ & (\mathsf{steq}(s,t)_{\mathcal{H}_\mathcal{R}} \text{ is continuous (Lemma 2.6.4)}) \\ = & \mathsf{true} \quad \text{(by assumption).} \end{array}$$

Therefore we see that there exist compact elements $\hat{z_1} \in \mathrm{H}^{\alpha_1}, \cdots, \hat{z_n} \in \mathrm{H}^{\alpha_n}$ such that $\hat{z_1} \sqsubseteq h_1, \cdots, \hat{z_n} \sqsubseteq h_n$ and

$$\mathsf{steq}(s,t)_{\mathcal{H}_\mathcal{R}}(\hat{z_1}, \cdots, \hat{z_n}) = \mathsf{true}.$$

So $\mathsf{steq}_{\mathcal{H}_\mathcal{R}}(s,t)(\hat{z_1}, \cdots, \hat{z_n}) \in \mathrm{D}^{\mathsf{Bool}}$. By Lemma 2.6.2, there exist $d_1 \in \mathrm{D}^{\alpha_1}, \cdots, d_n \in \mathrm{D}^{\alpha_n}$ such that $d_1 \sqsupseteq \hat{z_1}, \cdots, d_n \sqsupseteq \hat{z_n}$, and

$$\mathsf{steq}(s,t)_{\mathcal{H}_\mathcal{R}}(d_1, \cdots, d_n) = \mathsf{true}. \tag{2.1}$$

We fix the constructor terms $d_1, \cdots, d_n$ determined above. Let $\rho : \Delta \to \mathcal{H}_\mathcal{R}$ and $\theta : \Delta \to \mathcal{T}_\Sigma$ be assignments such that $\rho : \{x_1 \mapsto d_1, \cdots, x_n \mapsto d_n\}$ and $\theta : \{x_1 \mapsto d_1, \cdots, x_n \mapsto d_n\}$. Notice that θ is a substitution to constructor terms. By definition of the derived operator, Eq. (2.1) is rewritten as $\rho^\#(\mathsf{steq}(s,t)) = \mathsf{true}$. Let $\phi : \mathcal{T}_\Sigma \to \mathcal{H}_\mathcal{R}$ be the unique homomorphism. So $\phi^\#(\mathsf{steq}(s,t)\theta) = \mathsf{true}$ from $\mathcal{T}_\Sigma$ to $\mathcal{H}_\mathcal{R}$. By Lemma 2.6.5, we obtain $\mathsf{steq}(s,t)\theta \to^!_\mathcal{R} \mathsf{true}$. Hence we conclude $\mathcal{Q}_\mathcal{R} \models \forall\varnothing(\mathsf{steq}(s,t)\theta = \mathsf{true})$. ■

By the above proposition, we can conclude the existence of a normalized answer substitution for any valid query without a termination assumption of programs because a constructor term substitution is always normalized. So we can use narrowing as a sound and complete operational model of FLP in view of conditional equational logic. This is an answer to the problem presented in the last proof of Section 2.2. We now give the main theorem in this section.

Theorem 2.6.7
Let $\mathcal{R}$ be a program and $\exists\Delta(\mathsf{steq}(s,t) = \mathsf{true})$ a query. Then

$$\begin{aligned} & \mathcal{Q}_\mathcal{R} \models \exists\Delta(\mathsf{steq}(s,t) = \mathsf{true}) \\ \Leftrightarrow \quad & \mathcal{H}_\mathcal{R} \models \exists\Delta(\mathsf{steq}(s,t) = \mathsf{true}). \end{aligned}$$

Proof [⇒]:By $\mathcal{H}_\mathcal{R} \in \mathcal{M}od(\mathcal{R})$ and Theorem 2.2.9.
[⇐]:Suppose $\mathcal{H}_\mathcal{R} \models \exists\Delta(\mathsf{steq}(s,t) = \mathsf{true})$. By Proposition 2.6.6 there exists a normalized substitution θ such that

$$\mathcal{Q}_\mathcal{R} \models \forall\varnothing(\mathsf{steq}(s,t)\theta = \mathsf{true}).$$

By Herbrand's theorem for equational logic (Theorem 2.2.9), we have

$$\mathcal{Q}_\mathcal{R} \models \exists\Delta(\mathsf{steq}(s,t) = \mathsf{true}).$$

■

The implication (⇐) of the above theorem states that if a query has a witness in $\mathcal{H}_\mathcal{R}$, it also has a witness in $\mathcal{Q}_\mathcal{R}$. If the image of the witness[4] in $\mathcal{H}_\mathcal{R}$ is a set of constructor terms that are compact and total elements of H, then this statement is straightforward. But if the image of the witness contains an element that is not a constructor term, i.e., neither a compact element (meaning an infinite tree, called an infinite witness) nor a total element (meaning a tree that contains the element $\bot_\mathrm{H}$, called a partial witness), then this theorem presents quite a remarkable fact. The implication (⇐) of the theorem means that if an infinite or partial witness is found in $\mathcal{H}_\mathcal{R}$, then the existence of the witness in $\mathcal{Q}_\mathcal{R}$ whose image is a set of finite trees can be deduced. In other words, there does not exist a case where the query has only infinite or partial witnesses in $\mathcal{H}_\mathcal{R}$. If the query has an infinite or partial witness in $\mathcal{H}_\mathcal{R}$, then it certainly has a witness whose image is a set of constructor terms in $\mathcal{H}_\mathcal{R}$.

By this theorem, the execution of programs is regarded as solving equations in the complete Herbrand model $\mathcal{H}_\mathcal{R}$. This means that the function symbols may denote partial functions, and the computation may deal with infinite data. Moreover, narrowing can be used as a method for solving valid queries, because Proposition 2.6.6 ensures the existence of a normalized substitution in the quotient term model $\mathcal{Q}_\mathcal{R}$. The equivalence between the two semantics offers us this natural understanding and a solving method as well.

[4]If θ is a witness, the set $\{\theta(x) \mid x \in \mathcal{D}om(\theta)\}$ is called the image of the witness.

2.7 Categorical Semantics

In this section, we present categorical semantics of many-sorted conditional equational logic for a semantics of FLP. The categorical semantics can be considered as a generalization of the algebraic semantics, namely the algebraic semantics is a special case of the categorical semantics where the base category is $\mathcal{S}et$ of sets and functions. This categorical semantics is also extended to the semantics of an interactive functional-logic language in the next chapter.

A standard way to give categorical semantics to algebraic theories is known, for example [Cro93, Pit95]. In [Cro93] and [Pit95], the categorical semantics of many-sorted *un*conditional equational logic which does not have *existentially* quantified equations has been given. In this section we extend it to give the meaning of conditional and existentially quantified equations and show the soundness and completeness of this semantic treatment. The extension is straightforward, as we see below. Basic category theory will be assumed without comment; see [Mac71, Cro93].

Definition 2.7.1
Let $\mathcal{C}$ be a category with finite products and Σ a signature. A *structure* $\mathbb{M}$ in $\mathcal{C}$ for Σ is a pair $(\llbracket - \rrbracket_S : S \to \mathrm{obj}\ \mathcal{C}, \llbracket - \rrbracket_\Sigma : \Sigma \to \mathrm{arr}\ \mathcal{C})$ of functions such that

$$\llbracket f \rrbracket_\Sigma : \llbracket \alpha_1 \rrbracket_S \times \ldots \times \llbracket \alpha_n \rrbracket_S \to \llbracket \alpha \rrbracket_S$$

for each function symbol $f : \alpha_1 \times \ldots \times \alpha_n \to \alpha$. Given a context $\Gamma = (x_1 : \alpha_1, \ldots, x_n : \alpha_n)$, we set $\llbracket \Gamma \rrbracket \stackrel{def}{=} \llbracket \alpha_1 \rrbracket \times \ldots \times \llbracket \alpha_n \rrbracket$.

Definition 2.7.2
The meaning of proved terms in a structure $\mathbb{M}$ in a category $\mathcal{C}$ with finite products is given by the following function $\llbracket - \rrbracket_\mathbb{M}$ from proved terms to arrows of $\mathcal{C}$.

$$\begin{aligned}
\llbracket \Gamma, x : \alpha \ \triangleright\ x : \alpha \rrbracket_\mathbb{M} &= \pi : \llbracket \Gamma \rrbracket \times \llbracket \alpha \rrbracket_S \to \llbracket \alpha \rrbracket_S \\
\llbracket \Gamma \ \triangleright\ k : \alpha \rrbracket_\mathbb{M} &= \llbracket k \rrbracket_\Sigma : \llbracket \Gamma \rrbracket \to \llbracket \alpha \rrbracket_S \\
\llbracket \Gamma \ \triangleright\ f(t_1, \ldots, t_n) : \alpha \rrbracket_\mathbb{M} &= \llbracket f \rrbracket_\Sigma \circ \langle a_1, \ldots, a_n \rangle : \llbracket \Gamma \rrbracket \to \llbracket \alpha \rrbracket_S \\
\text{where } \llbracket \Gamma \ \triangleright\ t_i : \alpha_i \rrbracket_\mathbb{M} &= a_i : \llbracket \Gamma \rrbracket \to \llbracket \alpha_i \rrbracket_S \ \text{ for } i = 1, \ldots, n.
\end{aligned}$$

Validity is defined in a similar way to the algebraic semantics.

Definition 2.7.3
We use the following notations: Let $\mathbb{M}$ be a structure in a category $\mathcal{C}$.

(i) We write

$$\mathbb{M} \models_{\text{cat}} \forall\Gamma(s = t \Leftarrow s_1 = t_1, \cdots, s_n = t_n)$$

if for all arrows $w : A \to [\![\Gamma]\!]$,

$$[\![\Gamma \ \triangleright \ s]\!]_{\mathbb{M}} \circ w = [\![\Gamma \ \triangleright \ t]\!]_{\mathbb{M}} \circ w$$

whenever

$$[\![\Gamma \ \triangleright \ s_i]\!]_{\mathbb{M}} \circ w = [\![\Gamma \ \triangleright \ t_i]\!]_{\mathbb{M}} \circ w$$

for all $i = 1, \cdots, n$. Then we say that $\forall\Gamma(s = t \Leftarrow s_1 = t_1, \cdots, s_n = t_n)$ is *valid* in the structure $\mathbb{M}$.

(ii) We write

$$\mathbb{M} \models_{\text{cat}} \exists\Delta(\overrightarrow{s_i = t_i})$$

if there exists an arrow $w : 1 \to [\![\Delta]\!]$ such that $[\![\Delta \ \triangleright \ s_i]\!] \circ w = [\![\Delta \ \triangleright \ t_i]\!] \circ w$ for each $i = 1, \ldots, n$. Then we say that $\exists\Delta(\overrightarrow{s_i = t_i})$ is valid in the structure $\mathbb{M}$.

(iii) Let $\mathcal{R}$ be a set of conditional equations.

$$\mathbb{M} \models_{\text{cat}} \mathcal{R}$$

if for all conditional equations $\forall\Gamma(s = t \Leftarrow \overrightarrow{s_i = t_i})$ in $\mathcal{R}$, $\mathbb{M} \models_{\text{cat}} \forall\Gamma(s = t \Leftarrow \overrightarrow{s_i = t_i})$. Then we say that the structure $\mathbb{M}$ is a *model* of $\mathcal{R}$.

(iv) The category of all models of $\mathcal{R}$ in $\mathcal{C}$, denoted by $\mathcal{M}od(\mathcal{R}, \mathcal{C})$, has

- as objects the models of $\mathcal{R}$ in $\mathcal{C}$ and
- as arrows the homomorphism of models [Cro93] of $\mathcal{R}$ in $\mathcal{C}$.

(v) Let e be a universally or existentially quantified equation.

$$\mathcal{R} \models_{\text{cat}} e$$

if for all category $\mathcal{C}$ with finite products and models $\mathbb{M} \in \mathcal{M}od(\mathcal{R}, \mathcal{C})$, $\mathbb{M} \models_{\text{cat}} e$.

Soundness and completeness of this categorical semantics for equational logic are proved in the same way as in [Cro93]. Below we show these properties by following the standard method.

Lemma 2.7.4 (**Semantics of substitution**)
Let $\mathbb{M}$ be a structure in a category $\mathcal{C}$ with finite products, $\Gamma' \rhd s : \beta$ a proved term where $\Gamma' = (x_1{:}\alpha_1, \ldots, x_n{:}\alpha_n)$ and, $\Gamma \rhd t_i{:}\alpha_i$ a proved term for each $i = 1, \ldots, n$. Define $\theta : \Gamma' \to \mathcal{T}_\Sigma(\Gamma)$ as $x_i \mapsto t_i$. Then

$$[\![\Gamma \rhd s\theta : \beta]\!]_{\mathbb{M}} = [\![\Gamma' \rhd s{:}\beta]\!]_{\mathbb{M}} \circ \langle [\![\Gamma \rhd t_1{:}\alpha_1]\!]_{\mathbb{M}}, \ldots, [\![\Gamma \rhd t_n{:}\alpha_n]\!]_{\mathbb{M}} \rangle$$

Proof By induction on the derivation of $\Gamma' \rhd s : \beta$. ■

To prove completeness, the classifying category $\mathcal{C}l(\mathcal{R})$ and the generic model $\mathbb{G}$ of $\mathcal{R}$ in $\mathcal{C}l(\mathcal{R})$, which intuitively corresponds to the quotient term model in algebraic semantics, are used.

Definition 2.7.5 (**Classifying category**)
Let $\mathcal{R}$ be a set of axioms. The classifying category $\mathcal{C}l(\mathcal{R})$ is defined as follow: let $\Gamma = (x_1{:}\alpha_1, \ldots, x_n{:}\alpha_n)$, $\Gamma' = (x'_1{:}\alpha_1, \ldots, x'_n{:}\alpha_n)$.
object: A finite sequence $(\alpha_1, \ldots, \alpha_n)$ of sorts.
arrow: A finite sequence

$$([\Gamma \rhd t_1{:}\beta_1], \ldots, [\Gamma \rhd t_m{:}\beta_m]) : (\alpha_1, \ldots, \alpha_n) \to (\beta_1, \ldots, \beta_m)$$

where $[\Gamma \rhd t{:}\beta]$ denotes an equivalence class of a proved term modulo the equivalence relation $\sim$ defined as:

$$\Gamma \rhd t{:}\alpha \sim \Gamma' \rhd t'{:}\alpha \overset{def}{\iff} \mathcal{R} \vdash \forall\Gamma \,.\, t = t'\theta : \alpha$$

where $\theta : \Gamma \to \Gamma'$ is a variable-renaming. This arrow will be written as $(\Gamma \mid \overrightarrow{t}) : \overrightarrow{\alpha} \to \overrightarrow{\beta}$.
identity: $\mathrm{id}_{\overrightarrow{\alpha}} \overset{def}{=} (\Gamma \mid \overrightarrow{x_i})$
composition: Let $\Gamma' = (x'_1{:}\alpha_1, \ldots, x'_m{:}\alpha_m)$. For arrows $(\Gamma \mid \overrightarrow{t}) = ([\Gamma \rhd t_1 : \beta_1], \ldots, [\Gamma \rhd t_m : \beta_m]) : \overrightarrow{\alpha} \to \overrightarrow{\beta}$ and $(\Gamma' \mid \overrightarrow{t'}) = ([\Gamma' \rhd t'_1 : \gamma_1], \ldots, [\Gamma' \rhd t'_l : \gamma_l]) : \overrightarrow{\beta} \to \overrightarrow{\gamma}$,

$$(\Gamma' \mid \overrightarrow{t'}) \circ (\Gamma \mid \overrightarrow{t}) : \overrightarrow{\alpha} \to \overrightarrow{\gamma} \overset{def}{=} ([\Gamma \rhd t'_1\{\vec{x'} \mapsto \vec{t'}\}], \ldots, [\Gamma \rhd t'_l\{\vec{x'} \mapsto \vec{t'}\}]).$$

terminal object: The empty sequence ()
binary product: $(\Gamma' \mid \overrightarrow{t'}) \times (\Gamma \mid \overrightarrow{t}) \overset{def}{=} (\Gamma', \Gamma \mid \overrightarrow{t'}, \overrightarrow{t})$.

Definition 2.7.6 (**Generic model**)
Let Σ be an S-sorted signature and $\mathcal{R}$ a set of axioms. The generic model $\mathbb{G} = (\llbracket _ \rrbracket_S, \llbracket _ \rrbracket_\Sigma)$ of $\mathcal{R}$ in $\mathcal{C}l(\mathcal{R})$ is defined as follows:

$$\llbracket \alpha \rrbracket_S = (\alpha) \quad \text{for } \alpha \in S$$
$$\llbracket f : \alpha_1, \ldots, \alpha_n \to \beta \rrbracket_\Sigma = (x_1{:}\alpha_1, \ldots, x_n{:}\alpha_n \mid f(x_1, \ldots, x_n) : \beta) : \overrightarrow{\alpha_i} \to \beta$$

Theorem 2.7.7
Let $\mathcal{R}$ be a set of axioms and e a universally or existentially quantified equation. Then

$$\mathcal{R} \vdash e \quad \Leftrightarrow \quad \mathcal{R} \models_{\text{cat}} e.$$

Proof

soundness($\Rightarrow$): Let $\mathbb{M}$ be a model of $\mathcal{R}$ in $\mathcal{C}$. It is sufficient to show that each deduction rule of conditional equation logic is sound. We only check (axiom) and (existentially introduction) rules, which are additional part from unconditional many-sorted equational logic.
(axiom): Let $\Gamma = (y_1 : \beta_1, \ldots, y_m : \beta_m)$ and Γ' be contexts. Take an arbitrary axiom $\forall\Gamma(s = t{:}\alpha \Leftarrow s_1 = t_1{:}\alpha_1, \cdots, s_n = t_n{:}\alpha_n) \in \mathcal{R}$ and a substitution $\theta = \{\overrightarrow{x_j \mapsto u_j : \beta_j}\} : \Gamma \to \mathcal{T}_\Sigma(\Gamma')$. Suppose

$$\llbracket \Gamma' \rhd s_i\theta : \alpha_i \rrbracket_\mathbb{M} = \llbracket \Gamma' \rhd t_i\theta : \alpha_i \rrbracket_\mathbb{M} \quad \text{for each } i = 1, \ldots, n.$$

Define $v \stackrel{def}{=} \langle \llbracket \Gamma' \rhd u_1{:}\beta_1 \rrbracket_\mathbb{M}, \ldots, \llbracket \Gamma' \rhd u_m{:}\beta_m \rrbracket_\mathbb{M} \rangle : \llbracket \Gamma' \rrbracket \to \llbracket \beta_1 \rrbracket \times \cdots \times \llbracket \beta_m \rrbracket$. By Lemma 2.7.4,

$$\llbracket \Gamma \rhd s_i : \alpha_i \rrbracket_\mathbb{M} \circ v = \llbracket \Gamma \rhd t_i : \alpha_i \rrbracket_\mathbb{M} \circ v \quad \text{for each } i = 1, \ldots, n.$$

Since $\mathbb{M}$ is a model of $\mathcal{R}$, we have

$$\llbracket \Gamma \rhd s : \alpha \rrbracket_\mathbb{M} \circ v = \llbracket \Gamma \rhd t : \alpha \rrbracket_\mathbb{M} \circ v. \tag{2.2}$$

Therefore

$$\llbracket \Gamma' \rhd s\theta : \alpha \rrbracket_\mathbb{M} = \llbracket \Gamma \rhd s : \alpha \rrbracket_\mathbb{M} \circ v = \llbracket \Gamma \rhd t : \alpha \rrbracket_\mathbb{M} \circ v = \llbracket \Gamma' \rhd t\theta : \alpha \rrbracket_\mathbb{M}.$$

(existential introduction): Let $\Delta = (y_1{:}\beta_1, \ldots, y_m{:}\beta_m)$ and a ground substitution $\theta = \{x_i \mapsto u_i\} : \Delta \to \mathcal{T}_\Sigma$ for each $i = 1, \ldots n$. Suppose

$$\llbracket \rhd s_i\theta : \alpha_i \rrbracket_\mathbb{M} = \llbracket \rhd t_i\theta : \alpha_i \rrbracket_\mathbb{M}.$$

Define $w \stackrel{def}{=} \langle [\![\triangleright u_1{:}\beta_1]\!]_{\mathbb{M}}, \ldots, [\![\triangleright u_m{:}\beta_m]\!]_{\mathbb{M}} \rangle : 1 \to [\![\Delta]\!]$. By Lemma 2.7.4,

$$[\![\Delta \triangleright s_i : \alpha_i]\!]_{\mathbb{M}} \circ w = [\![\Delta \triangleright t_i : \alpha_i]\!]_{\mathbb{M}} \circ w. \quad \text{for each } i = 1, \ldots, n.$$

Hence

$$\mathbb{M} \models_{\text{cat}} \exists \Delta \,.\, s_1 = t_1{:}\alpha_1, \ldots, s_n = t_n{:}\alpha_n.$$

completeness($\Leftarrow$): For a universally quantified equation,

$$\begin{array}{ll} & \mathcal{R} \models_{\text{cat}} \forall \Gamma \,.\, s = t : \alpha \\ \Rightarrow & \mathbb{G} \models_{\text{cat}} \forall \Gamma \,.\, s = t : \alpha \\ \Rightarrow & (\Gamma \mid s) = [\![\Gamma \triangleright s{:}\alpha]\!]_{\mathbb{G}} = [\![\Gamma \triangleright t{:}\alpha]\!]_{\mathbb{G}} = (\Gamma \mid t) \\ \Rightarrow & \mathcal{R} \vdash \forall \Gamma \,.\, s = t : \alpha. \end{array}$$

For an existentially quantified equation,

$$\begin{array}{ll} & \mathcal{R} \models_{\text{cat}} \exists \Delta \,.\, s = t : \alpha \\ \Rightarrow & \mathbb{G} \models_{\text{cat}} \exists \Delta \,.\, s = t : \alpha \\ \Rightarrow & (\Delta \mid s) \circ w = [\![\Delta \triangleright s{:}\alpha]\!]_{\mathbb{G}} \circ w = [\![\Delta \triangleright t{:}\alpha]\!]_{\mathbb{G}} \circ w = (\Delta \mid t) \circ w \end{array}$$

where $w : 1 \to [\![\Delta]\!]$ is some arrow in $\mathcal{C}l(\mathcal{R})$, which is expressed as

$$w = (\mid t_1{:}\beta_1, \ldots, t_n{:}\beta_n) : 1 \to (\beta_1, \ldots, \beta_n)$$

where $t_i \in \mathcal{T}_\Sigma$ for each $i = 1, \ldots, n$. Define a substitution $\theta = \{\overrightarrow{x_i \mapsto t_i}\} : \Delta \to \mathcal{T}_\Sigma$. Then

$$\begin{array}{ll} & (\mid s\theta) = (\mid t\theta) : 1 \to \alpha \\ \Rightarrow & \mathcal{R} \vdash \forall \varnothing \,.\, s\theta = t\theta : \alpha \\ \Rightarrow & \mathcal{R} \vdash \exists \Delta \,.\, s = t : \alpha. \end{array}$$

■

2.8 Deriving Algebraic Semantics from Categorical Semantics

We derive algebraic semantics of equational logic defined in Section 2.3 from the categorical semantics. Suppose that $\mathcal{A} = (\{A^\alpha \mid \alpha \in S\}, \Sigma_{\mathcal{A}})$ is an Σ-algebra and it is a model of a set $\mathcal{R}$ of axioms. We take the category $\mathcal{S}et$ of

all sets and functions as the category $\mathcal{C}$ in the categorical semantics. Clearly $\mathcal{S}et$ has finite products as cartesian products and there are projection (π_i) and pairing $(\langle -,- \rangle)$ functions and the terminal object 1 (one element set $\{*\}$). Define the structure $\mathbb{M}_{\mathcal{A}} = (\llbracket - \rrbracket_S, \llbracket - \rrbracket_\Sigma)$ for $\mathcal{A}$ as follows.

$$\llbracket \alpha \rrbracket_S = A^\alpha \quad \text{for } \alpha \in S$$
$$\llbracket f \rrbracket_\Sigma = f_{\mathcal{A}} : A^{\alpha_1} \times \ldots \times A^{\alpha_n} \to A^\alpha \quad \text{for } f : \alpha_1 \times \ldots \times \alpha_n \to \alpha \in \Sigma.$$

Then we know that the categorical meaning of a proved term t in the structure $\mathbb{M}$ on $\mathcal{S}et$ is the derived operator $t_{\mathcal{A}}$ defined in Definition 2.6.3.

Lemma 2.8.1
Let $\mathbb{M}_{\mathcal{A}}$ be a structure in $\mathcal{S}et$ generated from a Σ-algebra $\mathcal{A}$. Then

$$\llbracket x_1 : \alpha_1, \ldots, x_n : \alpha_n \ \triangleright \ t : \alpha \rrbracket_{\mathbb{M}_{\mathcal{A}}} = t_{\mathcal{A}} : A^{\alpha_1} \times \ldots \times A^{\alpha_n} \to A^\alpha$$

Proof Induction on the structure of a well-typed term t. Let Γ be $x_1 : \alpha_1, \ldots, x_n : \alpha_n$.
Base case:
$\llbracket \Gamma \ \triangleright \ x_k : \alpha_k \rrbracket_{\mathbb{M}_{\mathcal{A}}} = \pi_k : A^{\alpha_1} \times \ldots \times A^{\alpha_n} \to A^{\alpha_k}$, where $\pi_k(a_1, \ldots, a_n) = a_k$ for each $a_i \in A^{\alpha_i}, i = 1, \ldots, n$. By the definition of derived operator, $(x_k)_{\mathcal{A}} : A^{\alpha_1} \times \ldots \times A^{\alpha_n} \to A^\alpha$ is defined as:

$$\begin{aligned} &(x_k)_{\mathcal{A}}(a_1, \ldots, a_n) = \theta(x_k) \\ &\text{where } \theta : \Gamma \to \mathcal{A} \\ &\qquad \theta(x_i) = a_i \quad \text{for each } i = 1, \ldots, n. \end{aligned}$$

Namely

$$(x_k)_{\mathcal{A}}(a_1, \ldots, a_n) = \theta(x_k) = a_k = \pi_k(a_1, \ldots, a_n).$$

Hence $(x_k)_{\mathcal{A}} = \pi_k$.
Induction step:
$\llbracket \Gamma \ \triangleright \ f(t_1, \ldots, t_m) \rrbracket_{\mathbb{M}_{\mathcal{A}}} = f_{\mathcal{A}} \circ \langle b_1, \ldots, b_m \rangle : A^{\alpha_1} \times \ldots \times A^{\alpha_m} \to A^\alpha$ where $\llbracket \Gamma \ \triangleright \ t_i : \alpha_i \rrbracket_{\mathbb{M}_{\mathcal{A}}} = b_i$ for each $i = 1, \ldots, m$. Let $a_i \in A^{\alpha_i}$ for each $i = 1, \ldots, n$. By the induction hypothesis, $b_i = t_{i\mathcal{A}}$ for each $i = 1, \ldots, m$. Therefore

$$\begin{aligned} f_{\mathcal{A}} \circ \langle b_1, \ldots, b_m \rangle(a_1, \ldots, a_n) &= f_{\mathcal{A}}(t_{1\mathcal{A}}(a_1, \ldots, a_n), \ldots, t_{m\mathcal{A}}(a_1, \ldots, a_n)) \\ &= f_{\mathcal{A}}(\theta^\#(t_1)(a_1, \ldots, a_n), \ldots, \theta^\#(t_m)(a_1, \ldots, a_n)) \\ &= f(t_1, \ldots, t_m)_{\mathcal{A}}(a_1, \ldots, a_n) \end{aligned}$$

where $\theta : \Gamma \to \mathcal{A}$ such that $\theta(x_i) = a_i$ for $i = 1, \ldots, n$. Hence $f_{\mathcal{A}} \circ \langle b_1, \ldots, b_m \rangle = f(t_1, \ldots, t_m)_{\mathcal{A}}$. ■

Conversely, for a model $\mathbb{M} = (\llbracket - \rrbracket_S, \llbracket - \rrbracket_\Sigma) \in \mathcal{M}od(\mathcal{R}, \mathcal{S}et)$, we can define the following Σ-algebra $\mathcal{A}_\mathbb{M} = (\{A^\alpha \mid \alpha \in S\}, \Sigma_{\mathcal{A}_\mathbb{M}})$:

carrier:

$$A^\alpha \stackrel{def}{=} \llbracket \alpha \rrbracket_S \quad \text{for } \alpha \in S$$

operation:

$$f_{\mathcal{A}_\mathbb{M}} \stackrel{def}{=} \llbracket f \rrbracket_\Sigma : A^{\alpha_1} \times \cdots \times A^{\alpha_n} \to A^\alpha \text{ for } f : \alpha_1, \ldots, \alpha_n \to \alpha_1.$$

Proposition 2.8.2

Let e be a universally quantified conditional equation or an existentially quantified equation and $\mathbb{M}$ a structure in $\mathcal{S}et$. Then

(i) $\mathcal{A} \models e \;\Rightarrow\; \mathbb{M}_\mathcal{A} \models e$,

(ii) $\mathbb{M} \models e \;\Rightarrow\; \mathcal{A}_\mathbb{M} \models e$.

Proof

(i) Case: e is a universally quantified equation

Suppose $\Gamma = (x_1 : \alpha_1, \ldots, x_n : \alpha_m)$ and $\mathcal{A} \models \forall \Gamma(s = t : \alpha \Leftarrow s_1 = t_1 : \alpha_1, \ldots, s_n = t_n : \alpha_n)$. Let B be a set (an object of $\mathcal{S}et$) and $w : B \to \llbracket \Gamma \rrbracket$ a function (an arrow of $\mathcal{S}et$) such that $\llbracket \Gamma \triangleright s_i : \alpha_i \rrbracket_{\mathbb{M}_\mathcal{A}} \circ w = \llbracket \Gamma \triangleright t_i : \alpha_i \rrbracket_{\mathbb{M}_\mathcal{A}} \circ w$ for each $i = 1, \ldots, n$ where

$$w(b) \stackrel{def}{=} (b_1, \ldots, b_m) \quad \text{for each } b \in B$$

where $b_i \in A^{\alpha_i}$ (note: the superscript b indicates just a syntactical superscript) for each $i = 1, \ldots, m$. Define the assignment $\theta : \Gamma \to \mathcal{A}$ as $\theta : x_j \mapsto b_j$ for each $j = 1, \ldots, m$. We obtain for $b \in B$ and each $i = 1, \ldots, n$,

$$\begin{aligned}
\theta^\#(s_i) &= s_{i\mathcal{A}}(b_1, \ldots, b_m) && \text{(by definition of derived operator)} \\
&= \llbracket \Gamma \triangleright s_i \rrbracket_{\mathbb{M}_\mathcal{A}}(b_1, \ldots, b_m) && \text{(by definition of } \llbracket _ \rrbracket) \\
&= \llbracket \Gamma \triangleright s_i \rrbracket_{\mathbb{M}_\mathcal{A}} \circ w(b) \\
&= \llbracket \Gamma \triangleright t_i \rrbracket_{\mathbb{M}_\mathcal{A}} \circ w(b) && \text{(assumption)} \\
&= \llbracket \Gamma \triangleright t_i \rrbracket_{\mathbb{M}_\mathcal{A}}(b_1, \ldots, b_m) \\
&= t_{i\mathcal{A}}(b_1, \ldots, b_m) \\
&= \theta^\#(t_i)
\end{aligned}$$

By $\mathcal{A} \models \forall\Gamma(s = t : \alpha \Leftarrow s_1 = t_1 : \alpha_1, \ldots, s_n = t_n : \alpha_n)$, we have $\theta^{b\#}(s) = \theta^{b\#}(t)$. Then

$$\begin{aligned}[\![\Gamma \triangleright s]\!]_{\mathbb{M}_\mathcal{A}} \circ w(b) &= [\![\Gamma \triangleright s]\!]_{\mathbb{M}_\mathcal{A}}(b_1, \ldots, b_m) = s_\mathcal{A}(b_1, \ldots, b_m) \\ &= \theta^{b\#}(s) = \theta^{b\#}(t) \\ &= [\![\Gamma \triangleright t]\!]_{\mathbb{M}_\mathcal{A}} \circ w(b).\end{aligned}$$

Hence $\mathbb{M}_\mathcal{A} \models \forall\Gamma(s = t : \alpha \Leftarrow s_1 = t_1 : \alpha_1, \ldots, s_n = t_n : \alpha_n)$.

Case: e is an existentially quantified equation
Suppose $\Delta = (x_1 : \alpha_1, \ldots, x_n : \alpha_m)$ and $\mathcal{A} \models \exists\Delta(s = t : \alpha)$. Now there exists an assignment $\theta : \Delta \to \mathcal{A}$ such that $x_i \mapsto a_i \in A^{\alpha_i}$ for each $i = 1, \ldots, m$ and $\theta^\#(s) = \theta^\#(t)$. Then

$$\begin{aligned}[\![\triangleright s : \alpha]\!]_{\mathbb{M}_\mathcal{A}}(a_1, \ldots, a_n) &= s_\mathcal{A}(a_1, \ldots, a_n) \\ &= \theta^\#(s) = \theta^\#(t) = [\![\triangleright t : \alpha]\!]_{\mathbb{M}_\mathcal{A}}(a_1, \ldots, a_n)\end{aligned}$$

Take a function $w : 1 \to [\![\Delta]\!]$ as $w : * \mapsto (a_1, \ldots, a_m)$. Then $[\![\Delta \triangleright s : \alpha]\!]_{\mathbb{M}_\mathcal{A}} \circ w = [\![\Delta \triangleright t : \alpha]\!]_{\mathbb{M}_\mathcal{A}} \circ w$. Hence $\mathbb{M}_\mathcal{A} \models \exists\Delta(s = t : \alpha)$.

(ii) Similar to (i).

■

Chapter 3

Interactive First-order FLP

In this chapter, we add interaction to a first-order FLP. Here the word "interaction" is meant in the sense of Barendregt's consideration of "interactive functional programming" [Bar96, Bar97]. More precisely, "interaction" means computation with side-effects outside of the programming environment, like file systems, user inputs, terminal outputs, etc. This kind of programming feature is necessary in realistic programs.

Our approach of defining syntax and semantics of an interactive functional-logic language is given by the following way.

- Syntax is given by an extension of Moggi's computational metalanguage [Mog91].

- Axiomatic semantics is given by the deduction system of the computational metalanguage.

- Categorical semantics is given by categories with finite products and the strong monad for side-effects [Mog88, Mog91].

- Operational semantics is given by translating the computational metalanguage to a conditional term rewriting system.

This chapter falls into two parts. In the first four sections we describe the computational metalanguage and its semantics following Moggi. In the next three sections we define the translation from interactive FLPs using the computational metalanguage to CTRSs and discuss its correctness.

3.1 Computational Metalanguage

Moggi's computational metalanguage is a formal system designed for describing and reasoning several kinds of computations, which includes partial computation, nondeterministic computations, computations with side-effects, exceptions or continuations. The computational metalanguage is an extension of many-sorted equational logic where the extensions are

- the introduction of computational types, and
- the introduction of let-terms having the computational types.

The most important feature of the computational metalanguage is that it distinguishes between values and computations by types and nests of let-terms express sequential computation in imperative programming languages.

These features and formalization of the computational metalanguage matches with our approach to an interactive first-order functional-logic language in view of equational logic because we can regards the interaction as an instance of computation that can be expressed by the computational metalanguage, and interactive functional-logic programs as restricted form of axioms of the computational metalanguage. Therefore we use the computational metalanguage as the basis of the first-order interactive functional-logic language. We extend Moggi's computational metalanguage to include conditional and existentially quantified equations for syntax.

The formal system of the computational metalanguage is defined by giving sorts, signature, terms, equations and deduction rules in the same way as in the case of many-sorted conditional equational logic.

Definition 3.1.1 ([**Mog91**])
A set S of sorts for the computational metalanguage is a set of *types* given by the following grammar:

$$\begin{array}{rcll} \alpha & ::= & b & \text{base type} \\ & | & T\alpha & \text{computational type} \end{array}$$

In this chapter, we use the naming convention that $\alpha, \alpha_1, \ldots$ denote types and $b, b_1, \ldots$ denote base types. Moggi's idea is to distinguish *values* and *computations* by types, which is done by introducing a unary type constructor T. We think of $T\alpha$ as a type of *computations* of elements of type α and a base type as a type of *values*.

A *computational signature* Σ is an S-sorted signature such that for every function symbol $f : \alpha_1, \ldots, \alpha_n \to \alpha$, every source sort $\alpha_1, \ldots, \alpha_n$ is a base type. The raw terms are defined as follows:

$$\begin{array}{rcll} t & ::= & x & \text{variable} \\ & | & k & \text{constant} \\ & | & f(t_1, \ldots, t_n) & \text{function term} \\ & | & [t]_T & \text{unit term} \\ & | & \mathsf{let}\ x := t_1\ \mathsf{in}\ t_2 & \mathsf{let}\text{-term} \end{array}$$

where k is a constant symbol of zero arity and f a function symbol of non-zero arity n in the signature Σ.

The typing rules consist of the rules given in Definition 2.1.3 and the following:

$$([_]) \quad \frac{\Gamma \ \triangleright\ t : \alpha}{\Gamma \ \triangleright\ [t]_T : T\alpha}$$

$$(\text{let}) \quad \frac{\Gamma \ \triangleright\ s : T\alpha \quad \Gamma, x : \alpha \ \triangleright\ t : T\beta}{\Gamma \ \triangleright\ \mathsf{let}\ x := s\ \mathsf{in}\ t : T\beta}$$

Universally quantified (conditional) and existentially quantified equations are also defined for these proved terms as specified by Definition 2.1.11. The notion of substitution is obviously extended to unit and let-terms: A substitution for the computational metalanguage is an assignment $\theta : \Delta \to \mathcal{T}_\Sigma(\Gamma)$ such that

$$\begin{aligned} \theta^\#(x) &= \theta(x), \\ \theta^\#(k) &= k, \\ \theta^\#(f(t_1, \ldots, t_n)) &= f_{\mathcal{A}}(\theta^\#(t_1), \ldots, \theta^\#(t_n)), \\ \theta^\#([t]_T) &= [\theta^\#(t)]_T, \\ \theta^\#(\mathsf{let}\ y := t_1\ \mathsf{in}\ t_2) &= \begin{cases} \mathsf{let}\ y := \theta^\#(t_1)\ \mathsf{in}\ t_2 & y \in \Delta \\ \mathsf{let}\ y := \theta^\#(t_1)\ \mathsf{in}\ \theta^\#(t_2) & y \notin \Delta\ \&\ y \notin \Gamma \\ \mathsf{let}\ y := \theta^\#(t_1)\ \mathsf{in}\ \theta^\#(t_2[w]_y) & y \notin \Delta\ \&\ y \in \Gamma \end{cases} \end{aligned}$$

where w is a fresh variable.

Definition 3.1.2
The deduction rules for the computational metalanguage consists of the rules

of Definition 2.1.12 and the following adding rules:

$$([_].\xi)\quad \frac{\forall\Gamma(t_1 = t_2 : \alpha)}{\forall\Gamma([t_1]_T = [t_2]_T) : T\alpha}$$

$$(\mathrm{let}.\xi)\quad \frac{\forall\Gamma(t_1 = t_2 : T\alpha_1)\quad \forall\Gamma, x : \alpha_1(t_1' = t_2' : T\alpha_2)}{\forall\Gamma((\mathsf{let}\ x := t_1\ \mathsf{in}\ t_1') = (\mathsf{let}\ x := t_2\ \mathsf{in}\ t_2') : T\alpha_2)}$$

Axiom schema **LET** is defined for any proved terms $\Gamma \triangleright t{:}\alpha$, $\Gamma \triangleright t_1{:}T\alpha_1$, $\Gamma, x_1{:}\alpha_1 \triangleright t_2 : T\alpha_2$, $\Gamma, x_2{:}\alpha_2 \triangleright t_3 : T\alpha_3$:

$$\begin{aligned}
(\mathrm{ass})\quad & \forall\Gamma((\mathsf{let}\ x_2 := (\mathsf{let}\ x_1 := t_1\ \mathsf{in}\ t_2)\ \mathsf{in}\ t_3) \\
& \qquad = (\mathsf{let}\ x_1 := t_1\ \mathsf{in}\ (\mathsf{let}\ x_2 := t_2\ \mathsf{in}\ t_3)) : T\alpha_3) \\
(\mathrm{let}.\beta)\quad & \forall\Gamma((\mathsf{let}\ x_1 := [t]_T\ \mathsf{in}\ t_2) = t_2[t]_{x_1} : T\alpha_2) \\
(\mathrm{unit})\quad & \forall\Gamma((\mathsf{let}\ x_1 := t_1\ \mathsf{in}\ [t_1]_T) = t_1 : T\alpha_1).
\end{aligned}$$

We write

$$\mathcal{R} \vdash_{\mathrm{cml}} e$$

where a theorem e is deduced from a set $\mathcal{R}$ of axioms and **LET** by using these deduction rules.

Again, a Herbrand theorem hold for the computational metalanguage.

Corollary 3.1.3 (**Herbrand's Theorem for the computational metalanguage**)
Let $\mathcal{R}$ be a set of axioms. Then, we have

$$\begin{aligned}
& \mathcal{R} \vdash \exists\Delta(s_1 = t_1, \ldots, s_n = t_n) \\
\Leftrightarrow\quad & \mathcal{R} \vdash \forall\varnothing(s_1\theta = t_1\theta), \ldots, \mathcal{R} \vdash \forall\varnothing(s_n\theta = t_n\theta) \quad \text{for some } \theta : \Delta \to \mathcal{T}_\Sigma
\end{aligned}$$

Proof It is clear from the (existential introduction) deduction rule in Definition 2.1.12. ■

3.2 Categorical Semantics Based on Strong Monads

The semantics of the computational metalanguage is given by categorical structure called strong monads. This categorical semantics is an extension of

the categorical semantics of many-sorted conditional equational logic given in Section 2.7, where terms of base types (i.e. values) are interpreted in exact same way as in the categorical semantics of many-sorted conditional equational logic, and terms of computational types (i.e. computations) are interpreted by using the monad structure.

In this section, we review the semantics of the computational metalanguage by following Moggi [Mog91]. First we give three preliminary definitions of monads from category theory.

Definition 3.2.1 ([**Mac71**])
A *monad* over a category $\mathcal{C}$ is a triple (T, η, μ), where $T : \mathcal{C} \to \mathcal{C}$ is a functor and $\eta : \mathrm{Id}_{\mathcal{C}} \stackrel{\cdot}{\to} T$, $\mu : T^2 \stackrel{\cdot}{\to} T$ are natural transformations, satisfying the following:

$$\begin{aligned} \mu_{TA}; \mu_A &= T(\mu_A); \mu_A, \\ \eta_{TA}; \mu_A &= \mathrm{id}_{TA} = T(\eta_A); \mu_A. \end{aligned}$$

Definition 3.2.2 (**Strong monad [Mog88]**)
A *strong monad* over a category $\mathcal{C}$ with finite products is a monad (T, η, μ) together with a natural transformation $\mathbf{t}_{A,B} : A \times TB \stackrel{\cdot}{\to} T(A \times B)$, called a tensor strength, such that

$$\begin{aligned} \mathbf{t}_{1,A}; T(r_A) &= r_{TA} \\ \mathbf{t}_{A\times B,C}; T(\alpha_{A,B,C}) &= \alpha_{A,B,TC}; (\mathrm{id}_A \times \mathbf{t}_{B,C}); \mathbf{t}_{A,B\times C} \\ (\mathrm{id}_A \times \eta_B); \mathbf{t}_{A,B} &= \eta_{A,B} \\ (\mathrm{id}_A \times \mu_B); \mathbf{t}_{A,B} &= \mathbf{t}_{A,TB}; T(\mathbf{t}_{A,B}); \mu_{A,B} \end{aligned}$$

where r and α are the natural isomorphisms

$$\begin{aligned} r_A &: (1 \times A) \to A, \\ \alpha_{A,B,C} &: (A \times B) \times C \to A \times (B \times C). \end{aligned}$$

Definition 3.2.3 (**Kleisli triple [Mac71]**)
A *Kleisli triple* over a category $\mathcal{C}$ is a triple $(T, \eta, _^*)$ that consists of

- a function $T : \mathrm{obj}\,\mathcal{C} \to \mathrm{obj}\,\mathcal{C}$,
- a natural transformation $\eta : \mathrm{Id}_{\mathcal{C}} \stackrel{\cdot}{\to} T$, and
- a function $_^*$ that yields an arrow $f^* : TA \to TB$ for each $f : A \to TB$

and furthermore, the following equations hold:

- $\eta_A^* = \mathrm{id}_{TA}$,
- $\eta_A; f^* = f$ for $f : A \to TB$,
- $f^*; g^* = (f; g^*)^*$ for $f : A \to TB$ and $g : B \to TC$.

It is well-known that there is a one-one correspondence between Kleisli triples and monads [Man76]. So we refer to a Kleisli triple with a tensor strength as a strong monad.

Next, the meaning of terms in a category equipped with finite products and a strong monad is defined. The meaning of existentially and universally quantified (conditional) equations is defined in the same way as in the first-order FLP. Then, we show that this semantics is sound and complete with respect to the deduction system defined in Definition 3.1.2.

Definition 3.2.4 ([**Mog91**])
Let $\mathcal{C}$ be a category with finite products. The structure $\mathbb{M}$ for the computational metalanguage in $\mathcal{C}$ is $(\llbracket_\rrbracket_S, \llbracket_\rrbracket_\Sigma, (T, \eta, _^*, \mathbf{t}))$ where $\llbracket_\rrbracket_S : S \to \mathrm{obj}\,\mathcal{C}$ and $\llbracket - \rrbracket_\Sigma : \Sigma \to \mathrm{arr}\,\mathcal{C}$ are functions such that

$$\llbracket k \rrbracket_\Sigma : 1 \to \llbracket \alpha \rrbracket_S$$

for each constant symbol $k : \alpha \in \Sigma$,

$$\llbracket f \rrbracket_\Sigma : \llbracket \alpha \rrbracket_S \times \cdots \times \llbracket \alpha_n \rrbracket_S \to \llbracket \alpha \rrbracket_S$$

for each function symbol $f : \alpha_1 \times \cdots \times \alpha_n \to \alpha \in \Sigma$, and $(T, \eta, _^*, \mathbf{t})$ is a strong monad such that

$$\llbracket T\alpha \rrbracket_S = T \llbracket \alpha \rrbracket_S.$$

The meaning of a proved term in a structure $\mathbb{M}$ in $\mathcal{C}$ is defined by Definition 2.7.2 together with the following:

$$\begin{aligned} \llbracket \Gamma \ \triangleright\ [t]_T : T\alpha \rrbracket_{\mathbb{M}} &= \eta_{\llbracket \alpha \rrbracket_S} \circ \llbracket \Gamma \ \triangleright\ t : \alpha \rrbracket_{\mathbb{M}} \\ \llbracket \Gamma \ \triangleright\ \mathsf{let}\ x := s\ \mathsf{in}\ t : T\beta \rrbracket_{\mathbb{M}} &= g_2^* \circ \mathbf{t}_{\llbracket \Gamma \rrbracket, \llbracket \alpha \rrbracket_S} \circ \langle \mathrm{id}_{\llbracket \Gamma \rrbracket}, g_1 \rangle \\ \text{where } g_1 &= \llbracket \Gamma \ \triangleright\ s : T\alpha \rrbracket_{\mathbb{M}} \\ g_2 &= \llbracket \Gamma, x : \alpha \ \triangleright\ t : T\beta \rrbracket_{\mathbb{M}}. \end{aligned}$$

The notion of *validity* and *model* for this structure is defined in the same way as in Definition 2.7.3. We use the symbol $\models_{\mathrm{cml}}$ for validity in this structure of the computational metalanguage.

The deduction system of the computational metalanguage is sound and complete with respect to this categorical semantics. Formally, this is stated as follows.

Theorem 3.2.5
Let Σ be a computational signature, $\mathcal{R}$ a set of axioms and e an unconditional universally or existential quantified equation. Then,

$$\mathcal{R} \vdash_{\mathrm{cml}} e \quad \Leftrightarrow \quad \mathcal{R} \models_{\mathrm{cml}} e.$$

Proof The proof is to extend the soundness and completeness result of Moggi's computational metalanguage [Mog91] by treating conditional and existential equations in the same way as in the proof of Theorem 2.7.7.

3.3 Interactive First-order FLP

In this section, we define interactive functional-logic programs as particular form of axioms of the computational metalanguage and queries as existentially quantified equations.

Definition 3.3.1
An *interactive FLP signature* Σ is an S-sorted signature satisfying the requirements of

- the first-order FLP signature (Definition 2.4.1), and
- the computational signature (Definition 3.1.1).

Definition 3.3.2 (**Interactive first-order FLP**)
A set $\mathcal{R}$ of axioms is called an *interactive FLP* if $\mathcal{R}$ satisfies the following:

(i) $\mathcal{R}$ is built from an interactive first-order FLP signature.

(ii) $\mathcal{R}$ is a properly-oriented orthogonal 3-CTRS[1].

[1] An interactive FLP $\mathcal{R}$ is not a usual CTRS because it contains let-terms. However the definition of properly orientedness, orthogonality and type-3-ness are applicable to the interactive FLP of the following reasons:

- orthogonality only depends on the left-hand sides of $\mathcal{R}$ and they are non-let-terms,
- properly orientedness and type-3-ness are conditions on variable occurrences.

(iii) $\mathcal{R}$ contains the set **STEQ** of axioms.

Furthermore, for each axiom $\forall\Gamma(l = r \Leftarrow \overrightarrow{s_i = t_i})$ in $\mathcal{R}$, the following conditions are satisfied:

(i) l is of the form $f(t_1, \ldots, t_n)$ where $f : b_1, \ldots, b_n \to \alpha \in$ FUN and $t_1, \ldots, t_n$ are constructor terms.

(ii) If the type of r is a computational type, the conditional part must be empty.

(iii) All the equations $s_1 = t_1, \ldots, s_n = t_n$ are strict equations on base types.

Definition 3.3.3 (**Query of interactive first-order FLP**)
Let $\mathcal{R}$ be a interactive FLP. A query of interactive FLP is an existentially quantified equation of the form

$$\exists\Delta \,.\, \overrightarrow{s_i = t_i}.$$

Execution of interactive FLP is to prove the query under the interactive FLP $\mathcal{R}$ as a set of axioms by obtaining answer substitutions using the deduction system of the computational metalanguage, i.e.

$$\mathcal{R} \vdash_{\text{cml}} \exists\Delta \,.\, \overrightarrow{s_i = t_i}$$

Example 3.3.4
Hereafter we omit universal quantifications in examples.

Sort set S:
Int, Unit, IntList, IO Int, IO Unit, ...

Signature Σ
read : IO Int,
write : Int → IO Unit,
readseq : Int → IO IntList,
main : IO Unit ∈ FUN
() : Unit,
:: : Int,IntList → IntList (cons),
[] : IntList (nill) ∈ CON

nth : IntList,Int → Int, (and other built-in functions ...) ∈ FUN

Interactive FLP $\mathcal{R}$
readseq(0) = [[]]$_{IO}$
readseq(S(k)) = let n:=read in
let u := readseq(k) in
[n::u]$_{IO}$

main = let u:= readseq(5) in
let m:= [nth(u,3)]$_{IO}$ in
write(m)

3.4 The Strong Monad for Side-effects

The computational metalanguage is a formal system for reasoning about several kinds of programming languages by changing the interpretation of T in a structure of semantics. Here we fix the interpretation of T for our interactive functional-logic language, by imposing the interpretation of *side-effects* on T. This interpretation is given by the strong monad for side-effects [Mog91]. In this section we review the definition of it.

Definition 3.4.1
To express interaction, The following strong monad $(T, \eta, _^*, \mathbf{t})$ for side-effects over a cartesian closed category $\mathcal{C}$ (having curry and eval) [Mog88] is defined as follows. Let St be an object of $\mathcal{C}$, whose intention is a set of *states*.

- $TA \stackrel{def}{=} \mathrm{St} \Rightarrow A \times \mathrm{St}$ (exponential).
- $\eta_A = \mathrm{curry}_{\mathrm{St},(A\times\mathrm{St}),A}(\eta_{A\times\mathrm{St}})$.
- $\mu_A = \mathrm{curry}_{\mathrm{St},(A\times\mathrm{St}),(T^2A)}(\mathrm{eval}_{\mathrm{St},T(A\times\mathrm{St})}; T(\mathrm{eval}_{\mathrm{St},(A\times\mathrm{St})}; \mu_{A\times\mathrm{St}})$.
- $\mathbf{t}_{A,B} = \mathrm{curry}_{\mathrm{St},(A\times\mathrm{St}),(A\times TB)}(\alpha_{A,TB,\mathrm{St}}; (\mathrm{id}_A \times \mathrm{eval}_{\mathrm{St},(B\times\mathrm{St})}; \mathbf{t}_{A,B\times\mathrm{St}}; T(\alpha^{-1}_{A,B,\mathrm{St}}))$.

The above is a general definition of the strong monad for side-effects in an arbitrary ccc. For a more concrete description, in the category $\mathcal{S}et$ of sets and functions, the strong monad is expressed as the following Kleisli triple [Mog91], where the object St is considered as the set of *states*. We use a meta-notation $\lambda x.t$ for denoting a function.

- $TA \stackrel{def}{=} \mathrm{St} \Rightarrow A \times \mathrm{St}$ (function space).
- $\eta_A(a) = \lambda s.(a, s)$.
- The function $_^*$ is defined as follows: If $f : A \to TB$ and $g \in TA$, then $f^*(g) = \lambda s.f(a)(s')$ where $(a, s') = g(s)$.
- $\mathbf{t}_{A,B}(a, g) = \lambda s.((a, b), s')$ where $(b, s') = g(s)$.

Example 3.4.2
If we interpret the program $\mathcal{R}$ in Example 3.3.4 in $\mathcal{S}et$, we can give a structure $(\llbracket _ \rrbracket_S, \llbracket _ \rrbracket_\Sigma, (T, \eta, _^*, \mathbf{t}))$ for a model as follows:

$$\llbracket \mathsf{Int} \rrbracket_S \stackrel{def}{=} \mathbb{Z} \quad \text{(the set of integers)}$$
$$\llbracket \mathsf{IntList} \rrbracket_S \stackrel{def}{=} \mathbb{Z}^* \quad \text{(the set of all finite sequences of } \mathbb{Z}. \text{)}$$

Then, we set

$$\llbracket \mathsf{St} \rrbracket_S \stackrel{def}{=} \mathbb{Z}^*$$
$$T \stackrel{def}{=} \mathsf{IO} : \mathrm{obj}\ \mathcal{S}et \to \mathrm{obj}\ \mathcal{S}et$$
$$\mathsf{IO}\ A \stackrel{def}{=} \mathbb{Z}^* \Rightarrow A \times \mathbb{Z}^* \quad \text{for every set } A.$$

3.5 A Translation from Interactive FLPs to CTRSs

Next, we give the operational semantics for interactive FLPs. Moggi did not give a reduction based operational semantics. If we give such a operational semantics for the computational metalanguage directly, it may not be sufficient one because it seems to be no way to reduce let-terms to non-let-terms without specifying the interpretation of T.

So our strategy to give the operational semantics is to use term rewriting for evaluating, and narrowing for solving interactive FLPs. Since an interactive FLP contains let-terms, rewriting needs to handle let-terms. Instead of directly defining rewriting on let-terms, we first translate let-terms to usual conditional rewrite rules, and then use usual conditional term rewriting and narrowing. The reason for taking this approach is that if we use usual conditional term rewriting for interactive FLPs, soundness and completeness

result of conditional narrowing can be obtained by using existing results. Since soundness and completeness of a solving method for queries are most important properties of FLP, we think that the translation approach we take here for interactive FLPs is convenient to ensure these properties without a complicated proof of completeness.

The ideas of the translation from interactive FLPs to CTRSs are the following:

- to be explicit "state passing variables" in function terms of a computational type $T\alpha$, and
- to use the conditional part of a conditional rewrite rule for expressing sequential computation, instead of using a let-term in an interactive FLP.

The second idea comes from Suzuki *et al.*'s consideration of "conditions as where-clauses" and its justification in rewriting theory [SMI95].

First we give a notion of normal form of let-terms, which is used in the translation.

Definition 3.5.1
let-normal forms are well-typed terms given by the following grammar:

$$m ::= x \mid k \mid f(t_1, \ldots, t_n) \mid \mathsf{let}\ x := t\ \mathsf{in}\ m$$

where $t_1, \ldots, t_n, t$ are well-typed non-let-terms.

α-conversion of the let-binding variables is derivable from the set **LET** of axioms.

Lemma 3.5.2
If $\Gamma \triangleright s{:}\alpha$, $\Gamma, x{:}T\alpha \triangleright t{:}\beta$ and $y(\neq x)$ do not occur in Γ,

$$\mathbf{LET} \vdash_{\mathrm{cml}} (\alpha)\ \ \forall\Gamma((\mathsf{let}\ x := t_1\ \mathsf{in}\ t_2) = (\mathsf{let}\ y := t_1\ \mathsf{in}\ t_2[y]_x)).$$

Proof

$$\begin{array}{lll} & \mathsf{let}\ x := s\ \mathsf{in}\ t & \\ = & \mathsf{let}\ x := (\mathsf{let}\ y := s\ \mathsf{in}\ [y]_T\ \mathsf{in}\ t) & (\mathrm{unit}) \\ = & \mathsf{let}\ y := s\ \mathsf{in}\ (\mathsf{let}\ x := [y]_T\ \mathsf{in}\ t) & (\mathrm{ass}) \\ = & \mathsf{let}\ y := s\ \mathsf{in}\ t[y]_x & (\mathrm{let}.\beta). \end{array}$$

Theorem 3.5.3
Let $\Gamma \triangleright s : \alpha$. There is a let-normal form m such that $\mathbf{LET} \vdash \forall\Gamma(s = m)$. Such a normal form is unique up to the congruence generated by (α).
Proof Similar to the normal form theorem in [Has97].

We can choose exactly one let-normal form that includes exactly the same let-binding variables m from the (α)-equivalence class of a let-normal form of s. We define the function normalize which maps any term to such chosen let-normal form.

Definition 3.5.4
Let $\mathcal{R}$ be an interactive FLP over an S-sorted FLP signature Σ. Define the new sort set S° generated by the following grammar:

$$\begin{array}{rcl} S^\circ \ni \alpha & ::= & b \\ & | & \mathsf{St} \\ & | & b \times \mathsf{St} \end{array}$$

where $b \in S$ and St (the type of *states*) is new type not occurring in S. The translation $_^\circ$ from an interactive FLP $\mathcal{R}$ to a CTRS $\mathcal{R}^\circ$ over the S°-sorted signature Σ° is defined as follows. We overload the translation function $_^\circ$ over several syntactic objects, i.e., signatures, proved terms, rules and equations, and furthermore the auxiliary function $_^\bullet$ over rules and equations. For signatures, the translation $_^\circ$ from Σ to a S°-sorted signature is defined as follows:

$$\begin{aligned} (k : b)^\circ &\stackrel{def}{=} k : b \\ (k : T\beta)^\circ &\stackrel{def}{=} k : \mathsf{St} \to \beta \times \mathsf{St} \\ (f : \alpha_1, \ldots, \alpha_n \to b)^\circ &\stackrel{def}{=} f : \alpha_1, \ldots, \alpha_n \to b \\ (f : \alpha_1, \ldots, \alpha_n \to T\beta)^\circ &\stackrel{def}{=} f : \alpha_1, \ldots, \alpha_n, \mathsf{St} \to \beta \times \mathsf{St} \\ \Sigma^\circ &\stackrel{def}{=} \{f^\circ \mid f \in \Sigma\} \cup \{\langle _, _ \rangle^b : b \times \mathsf{St} \mid \text{base type } b \in S\}. \end{aligned}$$

The superscript of the pairing constructor symbol $\langle _, _ \rangle^b$ is omitted hereafter. For proved terms, the translation $_^\circ$ takes a pair of a proved term and a variable s_0, which may be attached to a translated term, and returns a proved term over the signature Σ°. Below, we assume that Γ is a typing context, y

an arbitrary variable not occurring in Γ and, s and t are non-let-terms.

$$
\begin{aligned}
(\Gamma \ \triangleright\ t : b,\ s_0)^\circ &\stackrel{def}{=} \Gamma \ \triangleright\ t : b \\
(\Gamma \ \triangleright\ x : T\beta,\ s_0) &\stackrel{def}{=} \Gamma \ \triangleright\ x : \beta \times \mathsf{St} \\
(\Gamma \ \triangleright\ [t]_T : T\beta,\ s_0)^\circ &\stackrel{def}{=} \Gamma, s_0 : \mathsf{St} \ \triangleright\ (t, s_0) : \beta \times \mathsf{St} \\
(\Gamma \ \triangleright\ f(t_1, \ldots, t_n) : T\beta,\ s_0)^\circ &\stackrel{def}{=} \Gamma, s_0 : \mathsf{St} \ \triangleright\ f(t_1, \ldots, t_n, s_0) : \beta \times \mathsf{St}.
\end{aligned}
$$

A typing context Γ is often omitted in $(_)^\circ$ for proved terms.

$$
\begin{aligned}
&(\Gamma \ \triangleright\ \mathsf{let}\ x_1 := a_1\ \mathsf{in}\ \ldots \mathsf{let}\ x_n := a_n\ \mathsf{in}\ t, s_0)^\circ \\
&\stackrel{def}{=} (\Gamma, s_0 : \mathsf{St} \ \triangleright\ g(\overrightarrow{z_i}, s_0), \\
&\qquad \{g(\overrightarrow{z_i}, s_0) = (t, s_n)^\circ \Leftarrow (a_1, s_0)^\circ = \langle x_1, s_1\rangle, \ldots, (a_n, s_{n-1})^\circ = \langle x_n, s_n\rangle\})
\end{aligned}
$$

Here g is a fresh function symbol and

$$
\{\overrightarrow{z_i : \alpha_i}\} \stackrel{def}{=} \mathcal{V}(\mathsf{let}\ x_1 := a_1\ \mathsf{in}\ \ldots \mathsf{let}\ x_n := a_n\ \mathsf{in}\ t)\,.
$$

It is obvious that the translated proved terms are actually proved terms, i.e., they have a typing proof. For rules, the translation $_^\circ$ takes a rule and returns a pair of translated rules.

$$
\begin{aligned}
(\forall\Gamma\ .\ l \to r : \alpha)^\circ &\stackrel{def}{=} (\forall\Gamma\ .\ l \to \mathsf{normalize}(r) : \alpha)^\bullet \\
(\forall\Gamma\ .\ l \to r : b)^\bullet &\stackrel{def}{=} \forall\Gamma\ .\ l \to r : b \\
(\forall\Gamma\ .\ l \to t : T\beta)^\bullet &\stackrel{def}{=} \forall\Gamma, s_0\ .\ l' = t' : \beta \times \mathsf{St}
\end{aligned}
$$

$$
\begin{aligned}
&(\forall\Gamma \ \triangleright\ l \to \mathsf{let}\ x_1 := a_1\ \mathsf{in}\ \ldots \mathsf{let}\ x_n := a_n\ \mathsf{in}\ t : T\beta)^\bullet \\
&\qquad \stackrel{def}{=} \forall\Gamma, \overrightarrow{x_i : \alpha_i}, \overrightarrow{s_i : \mathsf{St}}\ .\ (l, s_0)^\circ \to (t, s_n)^\circ : \beta \times \mathsf{St} \\
&\qquad\qquad \Leftarrow (a_1, s_0)^\circ = \langle x_1, s_1\rangle, \ldots, (a_n, s_{n-1})^\circ = \langle x_n, s_n\rangle
\end{aligned}
$$

where $(l', \varnothing) \stackrel{def}{=} (l, s_0)^\circ, (t', \varnothing) \stackrel{def}{=} (t, s_0)^\circ$, t is a non-let-term and $s_0, \ldots, s_n$ are variables. $\mathcal{R}^\circ$ is defined by translating each rule in $\mathcal{R}$:

$$
\mathcal{R}^\circ \stackrel{def}{=} \{(\forall\Gamma\ .\ l \to r)^\circ \mid \forall\Gamma\ .\ l \to r \in \mathcal{R}\}.
$$

For universally quantified equations, the translation $_^\circ$ using the auxiliary function $_^\bullet$ takes a universally quantified equation is defined as follows:

$$(\forall\Gamma\ .\ s = t : \alpha)^\circ \stackrel{def}{=} (\forall\Gamma\ .\ \mathsf{normalize}(s) = \mathsf{normalize}(t) : \alpha)^\bullet$$
$$(\forall\Gamma\ .\ s = t : b)^\bullet \stackrel{def}{=} \forall\Gamma\ .\ s = t : b$$
$$(\forall\Gamma\ .\ s = t : T\beta)^\bullet \stackrel{def}{=} (\forall\Gamma, s_0 : \mathsf{St}\ .\ s' = t' : \beta \times \mathsf{St}, \delta_1 \cup \delta_2)$$

where $(s', \delta_1) \stackrel{def}{=} (s, s_0)^\circ$ and $(t', \delta_2) \stackrel{def}{=} (t, s_0)^\circ$. For existentially quantified equations, the translation $_^\circ$ using the auxiliary function $_^\bullet$ takes an existentially quantified equation and yields a pair consisting of a set of translated equation and a set of equations to be added to $\mathcal{R}^\circ$.

$$(\exists\Delta\ .\ \overrightarrow{s_i = t_i : \alpha_i})^\circ \stackrel{def}{=} (\overrightarrow{e_i}, \bigcup \delta_i)$$

where $(e_i, \delta_i) \stackrel{def}{=} (\mathsf{normalize}(s_i) = \mathsf{normalize}(t_i))^\bullet$ for each i, and

$$(s = t : b)^\bullet \stackrel{def}{=} (s = t : b, \varnothing)$$
$$(s = t : T\beta)^\bullet \stackrel{def}{=} (s' = t' : \beta \times \mathsf{St}, \delta_1 \cup \delta_2)$$

where $(s', \delta_1) \stackrel{def}{=} (s, s_0)^\circ$ and $(t', \delta_2) \stackrel{def}{=} (t, s_0)^\circ$.

Example 3.5.5
The translated CTRS $\mathcal{R}^\circ$ from the interactive FLP $\mathcal{R}$ presented in Example 3.3.4 is as follows.
readseq(0,s0) → []
readseq(S(k),s0) → ⟨n::u, s2⟩ ⇐read(s0)=⟨n,s1⟩, readseq(k,s1)=⟨u,s2⟩
main(s0) → write(m,s2) ⇐readseq(s0)=⟨u,s1⟩, nth(u,3,s1)=⟨m,s2⟩

We can easily see the following proposition.

Proposition 3.5.6
If $\mathcal{R}$ is an interactive first-order FLP over an interactive FLP signature Σ, then $\mathcal{R}^\circ$ a first-order FLP over a first-order FLP signature Σ°.

An interactive first-order FLP $\mathcal{R}$ may have function symbols of a computational type $T\alpha$, for example, read : IO Int and write : IO Int in Example 3.5.5. But such function symbols may not have axioms to determine their

return values. For instance, in the program $\mathcal{R}$ of Example 3.5.5 there is no axiom for read and write. This also means that we can freely interpret the meaning of read and write for the program $\mathcal{R}$ in any model, or equivalently, we can freely add axioms for read and write preserving the model-property of any structure. In expression, this is roughly expressed as follows. Let $f : \alpha_1, \ldots, \alpha_n \to T\alpha$ be a function symbol not occurring as an axiom of the form $f(t_1, \ldots, t_n) = r : T\alpha$ in $\mathcal{R}$. Then for any model $\mathbb{M}$ of $\mathcal{R}$, we have the following.

(i) $\mathbb{M} \models \mathcal{R}$ where $[\![f]\!]_\Sigma$ is an arbitrary interpretation of f.

(ii) $\mathbb{M} \models \mathcal{R} \cup \{f(t_1, \ldots, t_n) = r : T\alpha\}$ where $[\![f]\!]_\Sigma$ satisfies the axiom $f(t_1, \ldots, t_n) = r : T\alpha$.

A function symbol like f is called *built-in interactive function symbol*, which will be defined in Definition 3.6.1. Axioms for built-in function symbols f, are not specified in (i). From the view point of realistic implementation of interactive FLPs, this is is preferable. Because (i) means that we can freely implement the behaver of these interactive function symbols, and it does not impose that such implementation (axiomatization) is given by rewrite rules. In other words, even if we implement interactive functions in, for example, the C language, property (i) holds, where we mean by "implement" an arbitrary interpretation ($[\![f]\!]_\Sigma$).

However, the approach we take here is like (ii). We implement the behavior of interactive functions by rewrite rules. The reason is, as discussed in the introduction of this section, that we want to use term rewriting and narrowing as the operational semantics of interactive FLPs and to ensure its soundness and completeness by using existing results of narrowing.

3.6 Soundness of the Translation _°

In this section, we show that the translation _° is a sound translation with respect to the deduction system of the computational metalanguage.

We start by defining built-in interactive axioms.

Definition 3.6.1 (**Built-in interactive axioms**)
Let $\mathcal{R}$ be an interactive FLP over the S-sorted signature Σ. A set $\mathcal{R}_\mathrm{b}$ of axioms is called *built-in interactive axioms* if the following conditions are satisfied:

- $\mathcal{R}_{\mathrm{b}}$ is a first-order FLP over S°-sorted signature Σ°.
- For each axiom $l = r \in \mathcal{R}_{\mathrm{b}}$, l has the form $f(t_1, \ldots, t_n, s)$ where $f : b_1, \ldots, b_n, \mathrm{St} \to \alpha \times \mathrm{St} \in \mathrm{FUN}^{\circ} \subseteq \Sigma^{\circ}$, is called *interactive function symbol*.
- For any interactive function symbol $f : b_1, \ldots, b_n, \mathrm{St} \to \alpha \times \mathrm{St}$ and ground terms $t_1, \ldots, t_n$, there exist ground terms $a : \alpha, s : \mathrm{St}$ such that

$$f(t_1, \ldots, t_n) \to^*_{\mathcal{R}^{\circ} \cup \mathcal{R}_{\mathrm{b}}} \langle a, s \rangle.$$

Example 3.6.2
read and write in Example 3.3.4 are built-in interactive function symbols. An example of built-in interactive axioms for them are the following:
read : IntList → Int × IntList
read(n::s)=⟨n,s⟩
write : Int × IntList → Unit × IntList
write(n,s)=⟨(),n::s⟩

We need the following lemma for proving the soundness of the translation.

Lemma 3.6.3
Let $\Gamma \rhd t : \alpha$ and s_0 a variable. If t is neither a let-term nor a variable, for $\theta : \Gamma' \to \mathcal{T}_{\Sigma}(\Gamma''), \Gamma' \subseteq \Gamma$,

$$(t, s_0)^{\circ}\theta = (t\theta, s_0)^{\circ}.$$

The translation $_^{\circ}$ is sound for universally quantified theorems in the following sense.

Theorem 3.6.4
Let $\mathcal{R}$ be an interactive first-order FLP over an S-sorted signature Σ, and $\mathcal{R}_{\mathrm{b}}$ a set of built-in interactive axioms over an S°-sorted signature Σ°. Define $(\forall \Gamma' \,.\, s' = t' : \alpha, \delta) \stackrel{def}{=} (\forall \Gamma \,.\, s = t : \alpha)^{\circ}$. Then,

$$\mathcal{R} \vdash_{\mathrm{cml}} \forall \Gamma \,.\, s = t : \alpha \quad \Rightarrow \quad \mathcal{R}^{\circ} \cup \delta \cup \mathcal{R}_{\mathrm{b}} \vdash \forall \varnothing \,.\, s'\sigma = t'\sigma : \alpha$$

for every ground substitution $\sigma : \Gamma' \to \mathcal{T}_{\Sigma}$.
Proof Case $\alpha = b$. Clear.
Case $\alpha = T\beta$: By induction on the height of the proof. We proceed by a case analysis of the last used rule.

(i) Case (axiom)
Note that since now t is a computational type, the used axiom does not have a conditional part (cf. Definition 3.3.2). Suppose the following proof using the computational metalanguage (hereafter, "cml" is indicated at the left of a proof by the computational metalanguage).

$$\mathrm{cml}\frac{}{\forall\Gamma' \,.\, l\theta = r\theta : T\beta}(\text{ax.}),\ (\forall\Gamma \,.\, l \to r) \in \mathcal{R}, \theta : \Gamma \to \mathcal{T}_\Sigma(\Gamma').$$

(i-a) Case r is a non-let-term. Then $(\forall\Gamma, s : \mathsf{St} \,.\, (l,s)^\circ \to (r,s)^\circ) \in \mathcal{R}^\circ$. Since by Lemma 3.6.3, $(l\theta, s)^\circ = (l,s)^\circ\theta$ we can construct the following proof using many-sorted equational logic:

$$\frac{}{\forall\Gamma', s : \mathsf{St} \,.\, (l,s)^\circ\theta = (r,s)^\circ\theta}(\text{ax.}).$$

(i-b) Case r is a let-term. Without loss of generality, we can assume that r is a let-normal form ($\mathsf{let}\ x_1 := a_1\ \mathsf{in}\ \ldots \mathsf{let}\ x_n := a_n\ \mathsf{in}\ t$),

$$\mathrm{cml}\frac{}{\forall\Gamma' \,.\, l\theta = r\theta}(\text{ax.}),\ \forall\Gamma \,.\, (l \to r) \in \mathcal{R},\ \theta : \Gamma \to \mathcal{T}_\Sigma(\Gamma').$$

Then, the used axiom is translated to

$$\delta_1 : (l, s_0)^\circ \to (t, s_n)^\circ \Leftarrow (a_1, s_0)^\circ = \langle x_1, s_1\rangle, \ldots, (a_n, s_{n-1})^\circ = \langle x_n, s_n\rangle \in \mathcal{R}^\circ.$$

We will construct a proof for

$$\mathcal{R}^\circ \cup \{\delta_1, \delta_2\} \cup \mathcal{R}_\mathrm{b} \vdash (l\theta, s_0)^\circ\sigma = g(\overrightarrow{z_i}, s_0)\sigma$$

where $\sigma : \Gamma' \to \mathcal{T}_\Sigma$ is some ground substitution and

$$\delta_2 : g(\overrightarrow{z_i}, s_0) \to (t\theta, s_n)^\circ \Leftarrow (a_1, s_0)^\circ = \langle x_1, s_1\rangle, \ldots, (a_n, s_{n-1})^\circ = \langle x_n, s_n\rangle \in \mathcal{R}^\circ.$$

In the same way as the construction of a substitution ρ in (vi), we can define a substitution $\rho \stackrel{def}{=} \{\overrightarrow{z_i \mapsto z_i}, \overrightarrow{s_i \mapsto s_i}, \overrightarrow{x_i \mapsto x_i}\}$ where each z_i is an arbitrary ground term. Then,

$$\dfrac{\dfrac{\dfrac{}{(l,s_0)^\circ\theta\rho = (t,s_n)^\circ\theta\rho}\,(\text{ax.}),\delta_1}{(l\theta,s_0)^\circ\rho = (t\theta,s_n)\rho}\,(\text{Lemma 3.6.3}) \quad \dfrac{\overrightarrow{\dfrac{}{(a_i\theta,s_{i-1})^\circ\rho = \langle x_i,s_i\rangle\rho}}}{g(\overrightarrow{z_i},s_0)\rho = (t\theta,s_n)^\circ\rho}\,(\text{ax.}),\delta_2}{(l\theta,s_0)^\circ\rho = g(\overrightarrow{z_i},s_0)\rho}\,(\text{tr.})$$

(ii) Case (reflexivity), (transitivity), (symmetry), (permutation). Straightforward.

(iii) Case (congruence).

$$\mathrm{cml}\frac{\forall\Gamma\ .\ \overrightarrow{s_i = t_i : b_i}}{f(\overrightarrow{s_i}) = f(\overrightarrow{t_i}) : T\beta}(\text{congr.})$$

The function symbol $f : \overrightarrow{b_i} \to T\beta \in \Sigma$ is translated to $f : \overrightarrow{b_i}, \mathsf{St} \to \beta, \mathsf{St} \in \Sigma^\circ$. Hence

$$\dfrac{\dfrac{}{\forall\Gamma, s_0 : \mathrm{St}\ .\ \overrightarrow{(s_i, s_0)^\circ = (t_i, s_0)^\circ}}\ (\text{I.H.}) \qquad \dfrac{}{\forall\Gamma, s_0 : \mathrm{St}\ .\ s_0 = s_0}\ (\text{ref.})}{f(\overrightarrow{s_i}, s_0) = f(\overrightarrow{t_i}, s_0)}\ (\text{congr.})$$

(iv) Axioms in **LET**.

(iv-a) $\mathcal{R} \vdash_{\mathrm{cml}} (\text{ass})$. By normalization of let-terms in the translation $_^\circ$.

(iv-b) $\mathcal{R} \vdash_{\mathrm{cml}} (\text{let}.\beta)\ \forall\Gamma\ .\ (\mathsf{let}\ x_1 := [t]_T\ \mathsf{in}\ t_2) = t_2[t]_{x_1} : T\beta$. We will construct a proof for the translated $(\text{let}.\beta)$ by $_^\circ$:

$$\mathcal{R}^\circ \cup \delta \cup \mathcal{R}_{\mathrm{b}} \vdash g(\overrightarrow{z_i}, s_0)\sigma = (t_2[t]_{x_1}, s_0)^\circ\sigma$$

for any ground substitution $\sigma : \Gamma, s_0 : \mathrm{St} \to \mathcal{T}_\Sigma$ and

$$\delta : \forall\Gamma, s_0 : \mathsf{St}\ .\ g(\overrightarrow{z_i}, s_0) = (t_2, s_0)^\circ \Leftarrow \langle t, s_0\rangle = \langle x, s_1\rangle.$$

The proof is done in the following way. Let $\theta : \{x \mapsto t, s_1 \mapsto s_0\}$ be a substitution and $\sigma : \Gamma,\ s_0{:}\mathsf{St} \to \mathcal{T}_\Sigma$ be any ground substitution.

$$\dfrac{\dfrac{\dfrac{}{(t, s_0)^\circ\theta\sigma = \langle x, s_1\rangle\theta\sigma}\ (\text{ref.}), \delta}{g(\overrightarrow{z_i}, s_0)\theta\sigma = (t_2, s_0)^\circ\theta\sigma}\ (\text{ax.})}{g(\overrightarrow{z_i}, s_0)\sigma = (t_2, s_0)^\circ\sigma}\ (\theta \text{ does not affect})$$

(iv-c) $\mathcal{R} \vdash_{\mathrm{cml}} (\text{unit})$. Straightforward.

(v) Case $([_].\xi)$. Similar to the case (congruence).

(vi) Case $(\text{let}.\xi)$. Without loss of generality, we only consider let-normalized equations. Suppose

$$\mathrm{cml}\frac{\forall\Gamma\ .\ a_1 = a'_1 : T\alpha_1 \qquad \forall\Gamma, x : \alpha_1\ .\ t = t' : T\alpha_2}{\forall\Gamma\ .\ \begin{array}{l}\mathsf{let}\ x_1 := a_1\ \mathsf{in}\\ \quad \mathsf{let}\ x_2 := a_2\ \mathsf{in}\\ \quad \ldots\\ \quad \mathsf{let}\ x_n := a_n\ \mathsf{in}\\ \quad t\end{array} = \begin{array}{l}\mathsf{let}\ x_1 := a'_1\ \mathsf{in}\\ \quad \mathsf{let}\ x_2 := a'_2\ \mathsf{in}\\ \quad \ldots\\ \quad \mathsf{let}\ x_{n'} := a'_{n'}\ \mathsf{in}\\ \quad t'\end{array}}(\text{let}.\xi).$$

We will show

$$\mathcal{R}' \vdash g(\overrightarrow{z_i}, s_0)\rho = g'(\overrightarrow{z'_i}, s_0)\rho$$

where ρ is some ground substitution and

$$\begin{aligned}
&\mathcal{R}' \stackrel{def}{=} \mathcal{R}^\circ \cup \mathcal{R}_\mathrm{b} \cup \\
&\{\forall \overrightarrow{z_i}, s_0 \,.\, g(\overrightarrow{z_i}, s_0) = (t, s_n)^\circ \Leftarrow (a_1, s_0)^\circ = \langle x_1, s_1\rangle, \ldots, (a_n, s_{n-1})^\circ = \langle x_n, s_n\rangle \\
&\forall \overrightarrow{z'_i}, s_0 \,.\, g'(\overrightarrow{z'_i}, s_0) = (t', s'_{n'})^\circ \Leftarrow (a'_1, s'_0)^\circ = \langle x_1, s'_1\rangle, \ldots, (a'_{n'}, s'_{n'-1})^\circ = \langle x'_{n'}, s'_{n'}\rangle\}
\end{aligned}$$

By induction hypothesis,

$$\mathcal{R}^\circ \vdash \forall \Gamma, s_0 : \mathrm{St} \,.\, (a_1, s_0)^\circ = (a'_1, s_0)^\circ : \alpha \times \mathsf{St}, \tag{3.1}$$

$$\mathcal{R}^\circ \vdash \forall \Gamma, s_0 : \mathrm{St} \,.\, (t, s_0)^\circ = (t', s_0)^\circ : \alpha \times \mathsf{St}. \tag{3.2}$$

Let $\sigma : \Gamma, s_0 : \mathsf{St} \to \mathcal{T}_\Sigma$ be an arbitrary ground substitution. Then by the definition of built-in interactive axioms, there exist ground terms $\hat{x_1}, \hat{s_1}$ with respect to $\mathcal{R}'$ such that

$$\mathcal{R}' \vdash (a_1, s_0)^\circ\sigma = \langle \hat{x_1}, \hat{s_1}\rangle.$$

Next define a ground substitution $\sigma_1 : \{x_1 \mapsto \hat{x_1}, s_1 \mapsto \hat{s_1}\}$. Then there exist ground normal forms $\hat{x_2}, \hat{s_2}$ such that

$$\mathcal{R}' \vdash (a_2, s_1)^\circ\sigma\sigma_1 = \langle \hat{x_2}, \hat{s_2}\rangle.$$

Repeatedly, we have for some ground terms $\hat{x_n}, \hat{s_n}$,

$$\mathcal{R}' \vdash (a_n, s_{n-1})^\circ\sigma \cdots \sigma_{n-1} = \langle \hat{x_n}, \hat{s_n}\rangle.$$

Hence, by defining $\rho \stackrel{def}{=} \sigma \cdots \sigma_{n-1}$, we have

$$\mathcal{R}' \vdash (a_1, s_0)^\circ\rho = \langle x_1, s_1\rangle\rho, \ldots, \mathcal{R}' \vdash (a_n, s_{n-1})^\circ\rho = \langle x_n, s_n\rangle\rho. \tag{3.3}$$

Then, we can construct the following (slightly informal) proof.

$$\dfrac{\dfrac{\dfrac{(3.3)}{g(\overrightarrow{z_i}, s_0)\rho = (t_n, s_n)^\circ\rho}\ (\mathrm{ax.}) \qquad \dfrac{\dfrac{\dfrac{(3.3)\quad(3.2)}{(a'_1, s_0)^\circ\rho = \langle x_1, s_1\rangle\rho, \ldots, (a'_n, s_{n-1})^\circ\rho = \langle x_n, s_n\rangle\rho}\ (\mathrm{subst.})}{g'(\overrightarrow{z'_i}, s_0)\rho = (t'_n, s_n)^\circ\rho}\ (\mathrm{ax.}) \qquad (3.2)}{g'(\overrightarrow{z_i}, s_0) = (t_n, s_n)^\circ\rho}\ (\mathrm{tr.})}{g(\overrightarrow{z_i}, s_0)\rho = g'(\overrightarrow{z'_i}, s_0)\rho}\ (\mathrm{tr.})}{g(\overrightarrow{z_i}, s_0)\sigma = g'(\overrightarrow{z'_i}, s_0)\sigma}$$

■

The translation $_^\circ$ is also sound for existentially quantified theorem in the following sense.

Theorem 3.6.5
Let $\mathcal{R}$ be an interactive first-order FLP over an S-sorted signature Σ and $\mathcal{R}_\mathrm{b}$ a set of built-in interactive axioms over an S°-sorted signature Σ°. If

$$\mathcal{R} \vdash_\mathrm{cml} \exists\Delta \,.\, \overrightarrow{u_i = t_i : \alpha_i}$$

with an answer substitution $\theta : \Delta \to \mathcal{T}_\Sigma$, then

$$\mathcal{R}^\circ \cup \delta \cup \bigcup \zeta_j \cup \mathcal{R}_\mathrm{b} \vdash \exists\Delta \,.\, \overrightarrow{e_i}\sigma$$

where

- $\sigma : (s_0) \to \mathcal{T}_\Sigma$
- $(\overrightarrow{e_i}, \delta) \stackrel{def}{=} (\exists\Delta \,.\, \overrightarrow{u_i = t_i : \alpha_i})^\circ$
- an answer substitution is $\theta^\circ : \Delta \to \mathcal{T}_{\Sigma^\circ}$ such that $\theta^\circ(x_j) \stackrel{def}{=} v$, where $\Delta = (x_1, \ldots, x_n)$ and $(v, \zeta_j) \stackrel{def}{=} (\theta(x_j), s_0)^\circ$ for each $j = 1, \ldots, n$.

Proof Suppose $\mathcal{R} \vdash_\mathrm{cml} \exists\Gamma \,.\, u = t : T\beta$. By Herbrand's theorem for the computational metalanguage, there exists a ground substitution $\theta : \Gamma \to \mathcal{T}_\Sigma$ such that

$$\mathcal{R} \vdash_\mathrm{cml} u_i\theta = t_i\theta : \alpha_i \quad \text{for every } i.$$

For each equation $u_i = t_i$, translate both sides of the equation as follows:

$$(u_i', \delta_i) \stackrel{def}{=} (u_i, s_0)^\circ$$
$$(t_i', \xi_i) \stackrel{def}{=} (t_i, s_0)^\circ$$

and $\delta \stackrel{def}{=} \bigcup \delta_i \cup \bigcup \xi_i$. Note that this δ is the same as in $(\overrightarrow{u_i' = t_i'}, \delta) \stackrel{def}{=} (\exists\Delta \,.\, \overrightarrow{u_i = t_i : \alpha_i})^\circ$. Then we easily see that

$$(u_i'\,\theta^\circ, \delta_i\,\theta^\circ) = (u_i\theta, s_0)^\circ$$
$$(t_i'\,\theta^\circ, \xi_i\,\theta^\circ) = (t_i\theta, s_0)^\circ$$

where we obviously extend the notion of substitution to sets (δ_i and ξ_i). By the soundness of the translation $_^\circ$ (Theorem 3.6.4), for an arbitrary ground substitution $\sigma : (s_0) \to \mathcal{T}_\Sigma$,

$$\mathcal{R}^\circ \cup \bigcup \zeta_j \cup \delta \cup \mathcal{R}_\mathrm{b} \vdash \forall \varnothing \,.\, u_i' \theta^\circ \sigma = t_i' \theta^\circ \sigma \quad \text{for each } i.$$

Hence

$$\mathcal{R}^\circ \cup \bigcup \zeta_j \cup \delta \cup \mathcal{R}_\mathrm{b} \vdash \exists \Delta \,.\, \overrightarrow{u_i' \sigma = t_i' \sigma}.$$

■

3.7 Interaction and Solving Equations

In this section, we discuss how interaction and solving in interactive FLP are related. In Section 3.6, we showed that the translation $_^\circ$ was sound translation. In view of solving of query, by this result we derive the following completeness of narrowing for interactive functional-logic programs.

Theorem 3.7.1 (**Completeness of narrowing for interactive FLPs**) Let $\mathcal{R}$ be an interactive first-order FLP over an S-sorted signature Σ, and $\mathcal{R}_\mathrm{b}$ a set of built-in interactive axioms over an S°-sorted signature Σ°. Suppose

$$\mathcal{R} \vdash_\mathrm{cml} \exists \Delta \,.\, \overrightarrow{u_i = t_i : \alpha_i}$$

with an answer substitution $\theta : \Delta \to \mathcal{T}_\Sigma$. We apply Theorem 3.6.5. Let $\mathcal{R}' = \mathcal{R}^\circ \cup \delta \cup \bigcup \zeta_j \cup \mathcal{R}_\mathrm{b}$. If the substitution θ° determined by Theorem 3.6.5 is a normalizable answer substitution with respect to $\mathcal{R}'$, then for every $(\exists \Delta \,.\, \overrightarrow{e_i} \sigma) \in E$,

$$\overrightarrow{e_i} \sigma \rightsquigarrow^*_{\rho, \mathcal{R}'} \top$$

where $\rho \preccurlyeq \theta^\circ \downarrow_{\mathcal{R}'} [\Delta]$.

Proof

$$\begin{aligned}
& \mathcal{R} \vdash_\mathrm{cml} \exists \Delta \,.\, \overrightarrow{u_i = t_i : \alpha_i} \\
\Rightarrow\ & \mathcal{R}' \vdash \exists \Delta \,.\, \overrightarrow{e_i} \sigma \\
\Leftrightarrow\ & \mathcal{R}' \vdash \forall \varnothing \,.\, \overrightarrow{e_i} \sigma \theta^\circ \\
\Leftrightarrow\ & e_i \sigma \theta^\circ \to^*_{\mathcal{R}'} \mathsf{true} \\
\Rightarrow\ & e_i \sigma \rightsquigarrow^*_{\rho, \mathcal{R}'} \top \quad \text{where } \rho \preccurlyeq \theta^\circ \downarrow_{\mathcal{R}'} [\Delta].
\end{aligned}$$

■

The above result says that we can obtain the answer substitution of a given query with respect to an interactive functional-logic program by using conditional narrowing. However, there is a case that narrowing is *not sound* for solving under interactive functional-logic programs. This is caused from that the reverse implication of the soundness of the translation $_^\circ$ (Theorem 3.6.4) does not hold in general as we see below.

Example 3.7.2
We use the signature of Example 3.3.4 and skip:IO Unit $\in$ FUN.

Interactive program $\mathcal{R}$
main=let dummy:=skip in write(1)

Built-in interactive axioms $\mathcal{R}_{\mathrm{b}}$
skip(s) = ((),s)

Translated CTRS $\mathcal{R}^\circ$
main(s0) = write(1,s1) $\Leftarrow$skip(s0) = $\langle$dummy,s1$\rangle$
Then, we easily show that

$$\mathcal{R}^\circ \cup \mathcal{R}_{\mathrm{b}} \vdash \forall \varnothing \,.\, \mathsf{main}(s) = \mathsf{write}(1,s).$$

where s is an arbitrary constant of the type IntList. But

$$\mathcal{R} \not\vdash_{\mathrm{cml}} \forall \varnothing \,.\, \mathsf{main} = \mathsf{write}(1).$$

We will compare two logics, i.e. many-sorted conditional equational logic and the computational metalanguage, with respect to the translation $_^\circ$. From the above example, we see that many-sorted conditional equational logic has more theorems than the computational metalanguage. The reason is relatively clear because deduction in many-sorted computational equational logic needs to add the built-in interactive axioms (see Theorem 3.6.4), i.e. adding more axioms yields more theorems. Adding built-in interactive axioms is necessary for executing interactive functional-logic programs. If we do not add these axioms, for example, read(s), where s is an arbitrary constant of the type IntList, can not rewrite to a constructor term. By Theorem 3.6.4, we can use the computational metalanguage as a formal system of reasoning about interactive functional-logic program and such a correct reasoning is always simulated by conditional equational logic, or equivalently, rewriting or narrowing.

Example 3.7.3
Consider the program $\mathcal{R}$ in Example 3.3.4 and the following validation in the computational metalanguage.

$$\mathcal{R} \vdash_{\mathrm{cml}} \exists \mathsf{n}{:}\mathsf{Int}, \mathsf{c}{:}\mathsf{IO\,Int}\ .\ \mathsf{let\ x{:=}read\ in\ (let\ y{:=}write(n{+}x)\ in\ read)} = \mathsf{let\ z{:=}c\ in\ read} : \mathsf{IO\ Int} \quad (3.4)$$

This can be considered as proving a property of the interactive program $\mathcal{R}$. We demonstrate the translation $_^\circ$ and the application of Theorem 3.6.5.

An example of answer substitution for Eq. (3.4) is

$$\theta = \{\mathsf{n} \mapsto 2 : \mathsf{Int},\ \mathsf{c} \mapsto (\mathsf{let\ x'} = \mathsf{read\ in\ write}(2{+}\mathsf{x}')){:}\mathsf{IO\ Unit}\}$$

Equation 3.4 generates the following CTRS by the translation $_^\circ$,

$$\delta = \left\{ \begin{array}{l} \mathsf{g(n,s0)} \to \mathsf{read(s2)}{:}\ \mathsf{Unit} \times \mathsf{St} \Leftarrow \mathsf{read(s0)}{=}\langle \mathsf{x,s1}\rangle,\ \mathsf{write(u{+}x,s1)}{=}\langle \mathsf{y,s2}\rangle \\ \mathsf{h(c,s0)} \to \mathsf{read(s1)}{:}\ \mathsf{Unit} \times \mathsf{St} \Leftarrow \mathsf{c}{=}\langle \mathsf{z,s1}\rangle \end{array} \right\}$$

and the translated Eq. (3.4) in conditional equational logic is

$$\exists \mathsf{n}{:}\mathsf{Int} \times \mathsf{IntList},\ \mathsf{c}{:}\mathsf{Unit} \times \mathsf{St}\ .\ \mathsf{g}(\mathsf{n},s) = \mathsf{h}(\mathsf{c},s) :\ \mathsf{Unit} \times \mathsf{St}$$

where s is an arbitrary constant of the type IntList, which can be considered as some initial sequence of an input tape. Furthermore, the answer substitution θ is translated into

$$\theta^\circ = \{\mathsf{n} \mapsto 2{:}\mathsf{Int}, \mathsf{c} \mapsto \mathsf{k}(2,s){:}\mathsf{Int}{\times}\mathsf{St}\}$$

and the following additional rule is generated

$$\zeta = \{\mathsf{k(n,s0)} \to \mathsf{write}(\mathsf{n{+}x'}, \mathsf{s1}) : \mathsf{Unit} \times \mathsf{St} \Leftarrow \mathsf{read(s0)} = \langle \mathsf{x'}, \mathsf{s1}\rangle\}.$$

Then the translated Eq. (3.4) in conditional equational logic is

$$\mathcal{R}^\circ \cup \delta \cup \zeta \vdash \exists \mathsf{n}{:}\mathsf{Int} \times \mathsf{St}, \mathsf{c} :\ \mathsf{Unit} \times \mathsf{St}\ .\ \mathsf{g}(\mathsf{n},s) = \mathsf{h}(\mathsf{c},s){:}\mathsf{Unit} \times \mathsf{St}. \quad (3.5)$$

We see that θ° is actually an answer substitution for Eq. (3.5). For more concrete setting, e.g., we set s =[5,9]. This means that the sequence "5,9" is written in the input tape. Then we can use narrowing for Eq. (3.5) as follows

$$\mathsf{eq}(\mathsf{g}(\mathsf{n},[5,9]), \mathsf{h}(\mathsf{c},[5,9])) \rightsquigarrow_\rho^* \top$$

where

$$\rho = \{\mathsf{n} \mapsto 2, \mathsf{c} \mapsto \langle (), [9] \rangle, \ldots\}.$$

This answer substitution ρ means that in Eq. (3.4), the value n is 2, and the computation c consists of the null value () and the resulting tape where "9" is written. At this time, $\theta^{\circ}{\downarrow_{\mathcal{R}'}} = \rho[\Delta]$.

3.8 Meaning Preservation of the Translation $_^{\circ}$

It is necessary to ensure that the translation $_^{\circ}$ preserves the intended meaning of interaction. Intuitively it is clear from the ideas of this translation, but the formal statement of the preservation requires some consideration because

- theories are changed by the translation,
- the required categories for models are changed by the translation, namely
  - $\mathcal{R}$ is interpreted in an arbitrary cartesian closed category (ccc) equipped with the strong monad for side-effects.
  - $\mathcal{R}^{\circ}$ is interpreted in an arbitrary category with finite products.

The problem is how to compare the original interactive FLP and the translated CTRS and how to state the equivalence of them. One idea to realize such a comparison is to use the *equivalence of categories* [Mac71] between the classifying categories (i.e. term models) $\mathcal{C}l(\mathcal{R})$ and $\mathcal{C}l(\mathcal{R}^{\circ})$ (cf. Definition 2.7.5).

Recall that two categories $\mathcal{C}$ and $\mathcal{D}$ are *equivalent* if there is a functor $F : \mathcal{C} \to \mathcal{D}$ such that

- F is full and faithful (i.e. the arrow part of F is a one-to-one function),
- $\forall d \in \text{obj } \mathcal{D}, \exists c \in \text{obj } \mathcal{C}$ such that $d \cong F(c)$.

Now we see that $\mathcal{C}l(\mathcal{R})$ and $\mathcal{C}l(\mathcal{R}^{\circ})$ are not equivalent. To establish this equivalence, we must define a one-to-one correspondence between types and function symbols of $\mathcal{C}l(\mathcal{R})$ and $\mathcal{C}l(\mathcal{R}^{\circ})$. But this fails. From the definition of

the translation _°, it is a natural idea to define a functor $F : \mathcal{C}l(\mathcal{R}) \to \mathcal{C}l(\mathcal{R}^\circ)$ for equivalence by

$$F_{\mathrm{arr}}([\![f]\!] : [\![\alpha_1]\!] \times \ldots \times [\![\alpha_n]\!] \to [\![T\beta]\!]) \stackrel{def}{=} [\![f]\!] : [\![\alpha_1]\!] \times \ldots \times [\![\alpha_n]\!] \times \mathrm{St} \to [\![\beta]\!] \times \mathrm{St}.$$

To make F a functor, the corresponding object part is defined by

$$F_{\mathrm{obj}}([\![\alpha_1]\!] \times \ldots \times [\![\alpha_n]\!]) \stackrel{def}{=} [\![\alpha_1]\!] \times \ldots \times [\![\alpha_n]\!] \times \mathrm{St} \tag{3.6}$$

$$F_{\mathrm{obj}}([\![T\beta]\!]) \stackrel{def}{=} [\![\beta]\!] \times \mathrm{St}. \tag{3.7}$$

Then, F_{ob} is not clearly one-to-one because a type which has "$\times$St" at the end of the product type in $\mathcal{C}l(\mathcal{R}^\circ)$ does not always correspond to a computational type $T\alpha$ in $\mathcal{C}l(\mathcal{R})$ as we see in (3.6).

This disagreement is caused by the fact that the strong monad for side-effects is defined by $T\alpha \stackrel{def}{=} \mathrm{St} \Rightarrow \alpha \times \mathrm{St}$. In other words, although the type $T\alpha$ in $\mathcal{C}l(\mathcal{R})$ should correspond to the function type $\mathrm{St} \Rightarrow \alpha \times \mathrm{St}$, there is no such a function type in $\mathcal{C}l(\mathcal{R}^\circ)$ because $\mathcal{R}^\circ$ is first-order (i.e. all types are base types). In the translation _°, although $T\alpha$ is translated to $\alpha \times \mathrm{St}$, this is not one-to-one as discussed above.

To sum up, $\mathcal{C}l(\mathcal{R})$ is not equivalent to $\mathcal{C}l(\mathcal{R}^\circ)$. But there is a correspondence of arrows between $\mathcal{C}l(\mathcal{R})$ and $\mathcal{C}l(\mathcal{R}^\circ)$. This is easily obtained by the fact that the category $\mathcal{C}l(\mathcal{R})$ is a cartesian closed, namely,

$$\begin{aligned}\mathcal{C}l(\mathcal{R})(\alpha, T\beta) &= \mathcal{C}l(\mathcal{R})(\alpha, \mathrm{St} \Rightarrow \beta \times \mathrm{St}) \\ &\cong \mathcal{C}l(\mathcal{R})(\alpha \times \mathrm{St}, \beta \times \mathrm{St}) \\ &\cong \mathcal{C}l(\mathcal{R}^\circ)(\alpha \times \mathrm{St}, \beta \times \mathrm{St}).\end{aligned}$$

The last isomorphism is clear from the definition of _°.

Chapter 4

Simply-typed Applicative FLP

In this chapter, we give semantics of a class of higher-order functional-logic languages, called simply-typed applicative functional-logic languages. With respect to syntax, we introduce function types and applicative terms to treat function type data. We can view that this class of FLPs is syntactically a first-order one, so we can directly reuse semantics of first-order FLP of *syntactic* models, namely axiomatic, rewriting and the quotient term model. But the cpo model of first-order FLP, the least Herbrand model, is not directly applicable to applicative case because of the presence of function type terms. Since it is natural to interpret function type terms as (mathematical) functions, we extend the least Herbrand model to include function spaces as the semantic domain. This model is called *minimal applicative Herbrand model*. We explain that the "no junk" property of semantic domain is important for correspondence between the minimal applicative Herbrand model and narrowing. More precisely, we give correspondence between a syntactic solution of a query by narrowing and a semantic solution in the model.

4.1 Simply-typed Applicative Conditional Equational Logic

In this section, we introduce syntax and a deduction system for simply-typed applicative conditional equational logic in the same way as the first-order case defined in Chapter 2.

Definition 4.1.1 (**Signature**)
We mainly follow the terminology of Meinke's higher type universal algebra

[Mei92]. A set S of sorts and B of *type basis* are any nonempty sets. A set S of sort is a set of *types* S over a type basis B if $S = B \cup \{\alpha \Rightarrow \beta \mid \alpha, \beta \in S\}$. Each element $\alpha \in B$ and each element of the form $\alpha \Rightarrow \beta \in S$ are termed a *base type* and *function type* respectively. An *S-typed signature* Σ is a S-sorted signature such that for each function type $(\alpha \Rightarrow \beta) \in S$ we have application symbol $\mathsf{ap}^{\alpha\Rightarrow\beta} : (\alpha \Rightarrow \beta), \alpha \ \rightarrow \ \beta$.

Definition 4.1.2
Let Σ be an S-applicative signature. An S-typed Σ-algebra is an S-sorted Σ-algebra such that for each function type $\alpha \Rightarrow \beta \in S$, the following conditions are satisfied:

- $\mathcal{A}^{\alpha\Rightarrow\beta} \subseteq (\mathcal{A}^{\alpha} \Rightarrow \mathcal{A}^{\beta})$,
- $\mathsf{ap}_{\mathcal{A}}^{\alpha\Rightarrow\beta} : \mathcal{A}^{\alpha\Rightarrow\beta} \times \mathcal{A}^{\alpha} \Rightarrow \mathcal{A}^{\beta}$, $\mathsf{ap}_{\mathcal{A}}^{\alpha\Rightarrow\beta}(f, t) = f(t)$.

The superscript of ap is often omitted.

In this chapter, we only consider typed signatures. Since a typed signature is a particular many-sorted signature, all the definitions of syntactic objects and results on many-sorted conditional equational logic presented in Chapter 2 are immediately applicable. Instead of repeating definitions syntax related notations for typed signatures, we use the following terminology:

Sorted signature case	Typed signature case
Raw term	Applicative raw term
Well-typed term	Applicative term
Term algebra $\mathcal{T}_{\Sigma}(\Gamma)$	Applicative term algebra $\mathcal{AT}_{\Sigma}(\Gamma)$
Equation	Applicative equation
Axiom	Applicative axiom
CTRS	Applicative CTRS
Many-sorted conditional equational logic	Simply-typed applicative conditional equational logic
Quotient term algebra $\mathcal{Q}_{\mathcal{R}}$	Quotient applicative term algebra $\mathcal{AQ}_{\mathcal{R}}$

Note that we use that eq-term defined in Definition 2.2.11 is just the same form $\mathsf{eq}^{\alpha}(s, t)$ for the function symbol $\mathsf{eq}^{\alpha} : \alpha, \alpha \rightarrow \mathsf{Bool}$, not curried version

$\mathsf{eq}^{\alpha} : \alpha \Rightarrow \alpha \Rightarrow \mathsf{Bool}$, where $\Rightarrow$ is right associative. We use the usual abbreviations on applicative terms, e.g., an applicative term $\mathsf{ap}(\mathsf{ap}(f, \mathsf{ap}(s, 0)), 0)$ is written in abbreviated form as $f\ (s\ 0)\ 0$.

Then the following propositions are immediately obtained as corollaries of Theorem 2.3.4 and 2.3.6.

Corollary 4.1.3
Let $\mathcal{R}$ be a set of axioms. Then, $\mathcal{AQ}_{\mathcal{R}} \in \mathcal{M}od(\mathcal{R})$ and

$$\mathcal{R} \vdash \forall\varnothing(s = t) \quad \Leftrightarrow \quad \mathcal{AQ}_{\mathcal{R}} \models \forall\varnothing(s = t) \quad \Leftrightarrow \quad \mathcal{R} \models \forall\varnothing(s = t).$$

Moreover for any Σ-algebra $\mathcal{A} \in \mathcal{M}od(\mathcal{R})$, there exists a unique homomorphism $\phi : \mathcal{AQ}_{\mathcal{R}} \to \mathcal{A}$.

Corollary 4.1.4
Let $\mathcal{R}$ be a set of axioms. Then,

$$\mathcal{R} \vdash \exists\Delta(\overrightarrow{s_i = t_i}) \quad \Leftrightarrow \quad \mathcal{AQ}_{\mathcal{R}} \models \exists\Delta(\overrightarrow{s_i = t_i}) \quad \Leftrightarrow \quad \mathcal{R} \models \exists\Delta(\overrightarrow{s_i = t_i}).$$

4.2 Simply-typed Applicative FLP in Equational Logic Framework

We define applicative FLP and its solving method in the framework of applicative conditional equational logic.

Definition 4.2.1 (**Signature**)
An S-sorted signature Σ for simply-typed applicative FLP consists of the following three disjoint sets:

- $\{\mathsf{ap}^{\alpha\Rightarrow\beta} \mid \alpha, \beta \in S\}$ for application symbols,
- $\textsc{Con} = \{c : \alpha_1 \Rightarrow \cdots \Rightarrow \alpha_n \Rightarrow \beta \mid \alpha_1, \ldots, \alpha_n, \beta \text{ are base types}\}$ for constructor symbols,
- $\textsc{Fun}$ for defined function symbols,

and satisfies the following requirements:

- The sort set S contain the sort Bool.
- The signature Σ contains the following symbols:

- $\mathsf{steq}^{\alpha} : \alpha \Rightarrow \alpha \Rightarrow \mathsf{Bool} \in \textsc{Fun}$,
- $\&\ : \mathsf{Bool} \Rightarrow \mathsf{Bool} \Rightarrow \mathsf{Bool} \in \textsc{Fun}$,
- $\mathsf{true} : \mathsf{Bool} \in \textsc{Con}$.

Note that steq and $\&$ are curried function symbols and they are constant symbols (i.e. arity 0) in first-orders sense. A term that does not contain any occurrence of a defined function symbol is called *applicative constructor term*.

Definition 4.2.2 (**Applicative Strict Equality**)
Let Σ be a signature for applicative FLP. The set of axioms **ASTEQ** is the following.

$$\begin{aligned}
&\forall\varnothing(\mathsf{steq}^{\alpha}\ c\ \ c = \mathsf{true}) \quad \text{for each } c : \alpha \in \textsc{Con},\\
&\forall\overrightarrow{x_i : \alpha_i}, \overrightarrow{y_i : \alpha_i}\ .\ \mathsf{steq}^{\alpha}\ (d\ x_1\ \cdots\ x_n)\ \ (d\ y_1\ \cdots\ y_n))\\
&\quad = (\mathsf{steq}^{\alpha_i}\ x_1\ y_1)\ \ \&\ \cdots\ \&\ (\mathsf{steq}^{\alpha_i}\ x_n\ y_n))\\
&\qquad \text{for each } d : \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n \Rightarrow \alpha \in \textsc{Con},\\
&\qquad \forall x : \mathsf{Bool}(\mathsf{true}\ \&\ x = x).
\end{aligned}$$

We call an equation of the form $(\mathsf{steq}^{\alpha}\ s\ t) = \mathsf{true}$, where the terms s and t do not contain any occurrence of steq, an *applicative strict equation*.

Definition 4.2.3 (**Applicative FLP**)
A set $\mathcal{R}$ of applicative axioms is called an *applicative FLP* or simply *program* if $\mathcal{R}$ satisfies the following:

(i) $\mathcal{R}$ is built from a signature for many-sorted first-order FLP.

(ii) $\mathcal{R}$ is an orthogonal 3-CTRSs.

(iii) $\mathcal{R}$ contain the axioms **ASTEQ**.

Moreover, for each equation $\forall\Gamma(l = r \Leftarrow \overrightarrow{s_i = t_i})$ in $\mathcal{R}$, the following condition is satisfied:

(i) l is the form $f\ t_1\ \cdots\ t_n)$ where $f \in \textsc{Fun}$ and $t_1, \cdots, t_n$ are applicative constructor terms.

(ii) All the equations $s_1 = t_1, \ldots, s_n = t_n$ are strict equations.

Definition 4.2.4 (Query)
Let $\mathcal{R}$ be a program. A *query* of applicative FLP is an existentially quantified equation of the form

$$\exists\Delta(\overrightarrow{\mathsf{steq}\ s_i\ t_i = \mathsf{true}}).$$

Execution of FLP is to prove the query under the program $\mathcal{R}$ as axioms by obtaining witnesses, i.e.

$$\mathcal{R} \vdash \exists\Delta(\overrightarrow{\mathsf{steq}\ s_i\ t_i = \mathsf{true}}).$$

We can easily see that any applicative FLP is an orthogonal properly oriented right-stable 3-CTRS. So we can use conditional narrowing as a sound and complete solving method with respect to queries that have normalized solutions.

Example 4.2.5
Assume the following signature:

$$\begin{gathered}\mathsf{plus} : \mathsf{Nat} \Rightarrow \mathsf{Nat} \Rightarrow \mathsf{Nat},\\ \mathsf{map} : (\mathsf{Nat} \Rightarrow \mathsf{Nat}) \Rightarrow \mathsf{NatList} \Rightarrow \mathsf{Nat},\\ \mathsf{Nil} : \mathsf{NatList},\ \mathsf{S} : \mathsf{Nat} \Rightarrow \mathsf{Nat},\ \mathsf{Cons} : \mathsf{Nat} \Rightarrow \mathsf{NatList} \Rightarrow \mathsf{Nat}\end{gathered}$$

and a program consisting of an addition function on encoded natural numbers and "map" function over list.

$$\mathcal{R} = \left\{ \begin{array}{rl} \forall \mathsf{y} : \mathsf{Nat}. & \mathsf{plus\ 0\ y} = \mathsf{y} \\ \forall \mathsf{x,y} : \mathsf{Nat}. & \mathsf{plus\ (S\ x)\ y} = \mathsf{S\ (plus\ x\ y)} \\ \forall \mathsf{f} : \mathsf{Nat} \Rightarrow \mathsf{Nat}. & \mathsf{map\ f\ Nil} = \mathsf{Nil} \\ \forall \mathsf{f} : \mathsf{Nat} \Rightarrow \mathsf{Nat}, \mathsf{x} : \mathsf{Nat}, \mathsf{xs} : \mathsf{NatList}. & \mathsf{map\ f\ (Cons\ x\ xs)} \\ & \qquad = \mathsf{Cons\ (f\ x)\ (map\ f\ xs).}) \end{array} \right\}$$

Under the program $\mathcal{R}$, examples of queries are:

$$\mathcal{R} \models \exists \mathsf{z} : \mathsf{Nat}\ .\ \mathsf{steq\ (plus\ (S\ 0)\ z)\ (S\ z)} = \mathsf{true},$$

or

$$\mathcal{R} \models \exists \mathsf{f} : \mathsf{Nat} \Rightarrow \mathsf{Nat}\ .\ \mathsf{steq\ (map\ f\ [0, (S\ 0)])\ [S\ 0, S\ (S\ 0)]} = \mathsf{true}.$$

Operational and algebraic semantics of the simply-typed applicative functional-logic language will be concerned with this kind of query where solutions are

not only of base types but also of function types. Operational semantics of narrowing for the applicative functional-logic program will return the answers for the above queries as

$$\mathsf{z} \mapsto \mathsf{0} \quad \text{or} \quad \mathsf{z} \mapsto \mathsf{S\ 0} \quad \text{or} \quad \mathsf{z} \mapsto \mathsf{S\ 0} \quad \text{or} \quad \ldots$$

and

$$\mathsf{f} \mapsto \mathsf{S} \quad \text{or} \quad \mathsf{f} \mapsto \mathsf{plus\ (S\ 0)}.$$

Corollary 4.2.6
Let $\mathcal{R}$ be an applicative FLP.

$$\begin{aligned} & \mathcal{R} \vdash \forall\Gamma(\mathsf{steq}(s\theta, t\theta) = \mathsf{true}) \\ & \quad \text{for some normalized substitution } \theta : \Delta \to \mathcal{T}_\Sigma \\ \Leftrightarrow\ & \mathsf{steq}(s,t) \rightsquigarrow^*_{\theta'} \mathsf{true} \ \text{ where } \theta' \preccurlyeq \theta. \end{aligned}$$

Since an applicative FLP is also a (particular) set of axioms, the following is obtained from Theorem 2.3.4.

Corollary 4.2.7
Let $\mathcal{R}$ be an applicative FLP. Then $\mathcal{AQ}_\mathcal{R} \in \mathcal{M}od(\mathcal{R})$.

4.3 Minimal Applicative Herbrand Model

In this section, we give an applicative version of the least complete Herbrand model defined in Section 2.5 for first-order many-sorted FLP. This model called *minimal applicative Herbrand model* provides the following natural meaning of an applicative FLP:

- *Data* in applicative FLP are (infinite) constructor terms and *functions* (not function symbols) defined by a program of an applicative term rewriting system.
- *Functions* in a higher-order functional-logic programming are partial functions on the domain of *data*.
- A *query* of a higher-order functional-logic programming is an existentially quantified strict equation which can contain function type variables as quantified variables.

The soundness and completeness of narrowing with respect to the minimal applicative Herbrand model show that such intuitive explanations are completely correct. The construction of the minimal applicative model is a relatively natural extension of the least complete Herbrand model presented in Section 2.5. We add function spaces to the semantic domains for interpreting function types.

First we construct the applicative Herbrand universe.

Definition 4.3.1
Let Σ be a signature for applicative FLP. Define $\text{ACON} \stackrel{def}{=} \text{CON}^\alpha \cup \{\bot^\alpha : \alpha \mid \alpha \in S\} \cup \{\mathsf{ap}^{\alpha\Rightarrow\beta} \mid \alpha, \beta \in S\}$. The applicative Herbrand universe of sort α, AH^α, is the set of all ACON-trees (c.f. Definition 2.5.1).

The set of ground applicative data terms of the sort α, AD^α, is a proper subset of the applicative Herbrand universe AH^α by identifying them with the compact and total elements of AH^α, i.e. finite trees without any occurrence of $\bot^\alpha$.

Definition 4.3.2 (**Applicative Herbrand Algebra**)
A Σ-algebra $\mathcal{A}$ is called *applicative Herbrand algebra* if it satisfies the following:
carrier:

$$\text{AH}^\alpha \quad \text{for each sort } \alpha.$$

operations:

$$c_\mathcal{A} : \text{AH}^{\alpha_1\Rightarrow\ldots\Rightarrow\alpha_n\Rightarrow\alpha},$$
$$c_\mathcal{A}\ t_1\ \ldots\ t_n = c\ t_1\ \ldots\ t_n$$

for each constructor symbols $c : \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n \Rightarrow \alpha$. And

$$f_\mathcal{A} \text{ is a continuous function in } \text{AH}^\alpha$$

for each defined function symbol $f : \alpha \in \text{FUN}$.

Definition 4.3.3
Let $\mathcal{A}$ be an applicative Herbrand algebra. For each $\alpha \in S$, the order $\sqsubseteq_{\text{AH}^\alpha}$ on AH^α is defined as:

$$s \sqsubseteq_{\text{AH}^\alpha} t \iff^{def} s^\alpha(x) \sqsubseteq t^\alpha(x) \text{ for all } x \in \mathbb{N}_+^*.$$

Define $\mathcal{AH}erb\mathcal{A}lg$ as the class of all applicative Herbrand algebras, and the order $\sqsubseteq_{\mathcal{AH}erb\mathcal{A}lg}$ on $\mathcal{AH}erb\mathcal{A}lg$ as:

$$\mathcal{A} \sqsubseteq_{\mathcal{AH}erb\mathcal{A}lg} \mathcal{B} \stackrel{def}{\Longleftrightarrow} f_{\mathcal{A}} \sqsubseteq_{\mathrm{AH}^\alpha} f_{\mathcal{B}} \text{ for each } f : \alpha \in \mathrm{FUN}.$$

Proposition 4.3.4
For each $\alpha \in S$, $(\mathrm{AH}^\alpha, \sqsubseteq_{\mathrm{AH}^\alpha})$ and $(\mathcal{AH}erb\mathcal{A}lg, \sqsubseteq_{\mathcal{AH}erb\mathcal{A}lg})$ are algebraic cpos.

Definition 4.3.5
Let Σ be an S-typed signature and $\mathcal{R}$ an applicative FLP. Define an operator $\mathrm{T}_{\mathcal{R}} : \mathcal{AH}erb\mathcal{A}lg \to \mathcal{AH}erb\mathcal{A}lg$ as:

$$\begin{aligned}\mathrm{T}_{\mathcal{R}}(\mathcal{A}) \stackrel{def}{=} (&\{\mathrm{AH}^\alpha \mid \alpha \in S\},\\ &\{\{\mathsf{ap}_{\mathcal{A}}^{\alpha\Rightarrow\beta}\} \mid \alpha,\beta \in S\} \cup \mathrm{CON}_{\mathcal{A}} \cup \{\Phi_{\mathcal{A}}(f) \in \mathrm{AH}^\alpha \mid f : \alpha \in \mathrm{FUN}\}).\end{aligned}$$

For each $f \in \mathrm{FUN}^{\alpha_1\Rightarrow\ldots\Rightarrow\alpha_n\Rightarrow\alpha}$, a mapping $\Phi_{\mathcal{A}}(f) : \mathrm{AH}^{\alpha_1} \Rightarrow \ldots \Rightarrow \mathrm{AH}^{\alpha_n} \Rightarrow \mathrm{AH}^\alpha$ is defined as:

$$\Phi_{\mathcal{A}}(f)(h_1)\cdots(h_n) \stackrel{def}{=} \begin{cases} \delta^{\#}(r) & \text{if} \quad \text{there exists } (\forall X(f\ l_1\ \cdots\ l_n = r \\ & \qquad \Leftarrow s_1 = t_1, \cdots, s_m = t_m) \in \mathcal{R}, \\ & \quad \text{and } \delta : X \to \mathcal{A}, \\ & \quad \text{for every } i \in \{1,\cdots,n\}, h_i = \delta^{\#}(l_i), \\ & \quad \text{for every } i \in \{1,\cdots,m\}, \delta^{\#}(s_i) = \delta^{\#}(t_i), \\ \bot^\alpha & \text{otherwise.} \end{cases}$$

The operator $\mathrm{T}_{\mathcal{R}}$ is a continuous function on $\mathcal{AH}erb\mathcal{A}lg$. This can be proved by showing that each construct of $\mathrm{T}_{\mathcal{R}}$ is continuous similar to first-order case. By the fixpoint theorem on cpos, $\mathrm{T}_{\mathcal{R}}$ has the least fixpoint expressed as follows:

$$\mathcal{AH}_{\mathcal{R}}{}' \stackrel{def}{=} \bigsqcup_{i\in N} \mathrm{T}_{\mathcal{R}}{}^i(\bot^{\mathcal{AH}erb\mathcal{A}lg}).$$

Theorem 4.3.6
$\mathcal{AH}_{\mathcal{R}}{}'$ is the least (with respect to $\sqsubseteq_{\mathcal{AH}erb\mathcal{A}lg}$) applicative Herbrand model of a program $\mathcal{R}$.
Proof It is proved in the same way as the first-order case [GLMP91, Ham95].

The carrier of $\mathcal{AH}_{\mathcal{R}}{}'$ contains many elements that cannot be denoted by applicative terms. For example, consider a program $\mathcal{R} = \{\forall x : \mathsf{Nat}(\mathsf{id}\ x = x)\}$ and a query

$$\mathcal{R} \models \exists f : \mathsf{Nat} \Rightarrow \mathsf{Nat}\ .\ \mathsf{steq}\ (f\ 1)\ 1 = \mathsf{true}$$

where the signature Σ consists of $\textsc{Con} = \{1 : \mathsf{Nat}, \mathsf{true} : \mathsf{Bool}\}$ and $\textsc{Fun} = \{\mathsf{id} : \mathsf{Nat} \Rightarrow \mathsf{Nat}, \mathsf{steq}_{\mathsf{Nat}}, \ldots\}$. A witness of the above query in $\mathcal{AH}_{\mathcal{R}}{}'$ is $\xi : f \mapsto \mathsf{id}_{\mathcal{AH}_{\mathcal{R}}{}'}$ where $\mathsf{id}_{\mathcal{AH}_{\mathcal{R}}{}'} \in \mathrm{AH}^{\mathsf{Nat} \Rightarrow \mathsf{Nat}}$ is the identity function on $\mathrm{AH}^{\mathsf{Nat}}$. The witness ξ can be *represented* as a substitution $\theta : f \mapsto \mathsf{id}$. This means that the system of a applicative FLP can return an answer for the query as the form of the applicative term id.

On the other hand, there exits another witness for the above query, which is $\zeta : f \mapsto \iota$ where the function $\iota \in \mathrm{AH}^{\mathsf{Nat} \Rightarrow \mathsf{Nat}}$ is defined as $\iota(x) \stackrel{def}{=} 1$. But the witness ζ *cannot* be represented by any term substitution because the function $\iota \in \mathrm{AH}^{\mathsf{Nat} \Rightarrow \mathsf{Nat}}$ is not a denotation of any applicative term constructed from the signature Σ.

This unsatisfactory result is caused by the definition of the carrier of function types. Since the carrier $\mathrm{AH}^{\mathsf{Nat} \Rightarrow \mathsf{Nat}}$ is defined as the set of *all* continuous functions on $\mathrm{AH}^{\mathsf{Nat}}$, ι is an element of $\mathrm{AH}^{\mathsf{Nat} \Rightarrow \mathsf{Nat}}$ in this example. So ι is a witness of that query. However, in order to obtain the answer substitution corresponding to the witness of queries, we will exclude such ι from the admissible witnesses that cannot be represented by substitutions to applicative terms. By minimizing the carrier of $\mathcal{AH}_{\mathcal{R}}{}'$ to the set only containing elements that can be denoted by applicative terms, our intended model is obtained. Formally, this is done as follows.

Definition 4.3.7
Let $\mathcal{R}$ be a program. The S-typed Σ-algebra $\mathcal{AH}_{\mathcal{R}}$ is the homomorphic image of the unique homomorphism from $\mathcal{AT}_{\Sigma}$ to $\mathcal{AH}_{\mathcal{R}}{}'$.

Then, we obtain the following.

Corollary 4.3.8
$\mathcal{AH}_{\mathcal{R}}$ is a model of a program $\mathcal{R}$.

We call the Σ-algebra $\mathcal{AH}_{\mathcal{R}}$ *minimal applicative Herbrand model*. The terminology "minimal" comes from the notion of minimal algebra [Mei92], which is also called term generated in the literature. In other words, $\mathcal{AH}_{\mathcal{R}}$ has "no junk" [MG85].

4.4 Soundness and Completeness

In this section we show that the minimal applicative Herbrand model is sound and complete for the operational model of narrowing. The soundness and

completeness results stated here express the correspondence between a witness in the minimal applicative Herbrand model and a computed substitution by narrowing.

Theorem 4.4.1 (**Soundness of Narrowing for $\mathcal{AH}_\mathcal{R}$**)
Let $\mathcal{R}$ be a program and $\exists\Delta(\mathsf{steq}\ s\ t = \mathsf{true})$ a query. Then,

$$\begin{array}{rl} & (\mathsf{steq}\ s\ t) \leadsto_\theta^* \mathsf{true} \\ \Rightarrow & \mathcal{AH}_\mathcal{R} \models \exists\Delta(\mathsf{steq}\ s\ t = \mathsf{true}) \\ & \quad \text{with a witness } \delta : \Delta \to \mathcal{AH}_\mathcal{R} \end{array}$$

where $\theta : \Delta \to \mathcal{AT}_\Sigma(\Delta')$ and $\delta = \mathcal{AH}_\mathcal{R}[\![_]\!] \circ \rho^\# \circ \theta$ for arbitrary substitution $\rho : Y \to \mathcal{AT}_\Sigma$.

Proof Suppose $(\mathsf{steq}\ s\ t) \leadsto_\theta^* \mathsf{true}$. Then by the soundness of narrowing for rewriting, for any substitution $\rho : Y \to \mathcal{AT}_\Sigma$, $(\mathsf{steq}\ s\ t)\theta\rho \to_\mathcal{R}^* \mathsf{true}$. Then, $\mathcal{AQ}_\mathcal{R}[\![(\mathsf{steq}\ s\ t)\theta\rho]\!] = \mathcal{AQ}_\mathcal{R}[\![\mathsf{true}]\!]$. Let $\phi : \mathcal{AQ}_\mathcal{R} \to \mathcal{AH}_\mathcal{R}$ be the unique homomorphism. Since $\mathcal{AH}_\mathcal{R}[\![_]\!] = \phi \circ \mathcal{AQ}_\mathcal{R}[\![_]\!]$,

$$\mathcal{AH}_\mathcal{R}[\![_]\!] \circ \rho^\# \circ \theta\,(\mathsf{steq}\ s\ t) = \phi(\mathcal{AQ}_\mathcal{R}[\![(\mathsf{steq}\ s\ t)\theta\rho]\!]) = \phi(\mathcal{AQ}_\mathcal{R}[\![\mathsf{true}]\!]) = \mathsf{true}.$$

Hence $\mathcal{AH}_\mathcal{R} \models \exists\Delta(\mathsf{steq}\ s\ t = \mathsf{true})$. ■

We define an S-indexed subclass $\mathrm{AD} = \{\mathrm{AD}^\alpha \mid \alpha \in S\} \subseteq \mathcal{AT}_\Sigma$ as follows:

$$\begin{array}{rcl} \mathrm{AD}^\alpha & \stackrel{def}{=} & \{\text{ all ground applicative constructor terms of base type } \alpha\}, \\ \mathrm{AD}^{\alpha\Rightarrow\beta} & \stackrel{def}{=} & \mathcal{AT}_\Sigma^{\alpha\Rightarrow\beta} \text{ for each } \alpha, \beta \in S. \end{array}$$

This class D plays an intermediate role between $\mathcal{AH}_\mathcal{R}$ and $\mathcal{AQ}_\mathcal{R}$ for witnesses.

Lemma 4.4.2
Let $\mathcal{R}$ be a program and $\exists\Delta(\mathsf{steq}\ s\ t = \mathsf{true})$ a query.

$$\begin{array}{rl} & \mathcal{AH}_\mathcal{R} \models \exists\Delta(\mathsf{steq}\ s\ t = \mathsf{true}) \\ \Rightarrow & \mathcal{AQ}_\mathcal{R} \models \forall\varnothing((\mathsf{steq}\ s\ t)\xi = \mathsf{true}) \text{ for some } \xi : \Delta \to \mathrm{AD}. \end{array}$$

Proof Without loss of generality, we can assume that the typing context $\Delta = (x_1 : b_1, \ldots, x_m : b_m, y_1 : \alpha_1, \ldots, y_n : \alpha_n)$ satisfies that each b_i is base

type and each α_j a function type. Correspondingly, suppose δ is a witness in $\mathcal{AH}_\mathcal{R}$ such that

$$\delta : x_i \mapsto h_i \in \mathrm{AH}^{b_i} \text{ for } i = 1, \ldots, m$$
$$\delta : y_i \mapsto f_i \in \mathrm{AH}^{\alpha_i} \text{ for } i = 1, \ldots, n.$$

Then,

$$\begin{aligned}
& \delta^{\#}(\mathsf{steq}\ s\ t) \\
= \ & (\mathsf{steq}\ s\ t)_{\mathcal{AH}_\mathcal{R}}(h_1, ..., h_m, f_1, ..., f_n) \\
= \ & (\mathsf{steq}\ s\ t)_{\mathcal{AH}_\mathcal{R}}(\overrightarrow{\bigsqcup\{z_i \mid \text{compact } z_i \sqsubseteq h_i\}}, f_1, ..., f_n) \\
= \ & \bigsqcup\{(\mathsf{steq}\ s\ t)_{\mathcal{AH}_\mathcal{R}}(z_1, ..., z_m, f_1, ..., f_n) \mid \text{compact } z_i \sqsubseteq h_i \text{ for each } i = 1, ..., m\} \\
& \qquad \text{(by the continuity of } (\mathsf{steq}\ s\ t)_{\mathcal{AH}_\mathcal{R}}) \\
= \ & \mathsf{true} \quad \text{(by assumption)}
\end{aligned}$$

Therefore there exist compact elements $\hat{z_1} \in \mathrm{AH}^{b_1}, ..., \hat{z_m} \in \mathrm{AH}^{b_m}$ such that

$$(\mathsf{steq}\ s\ t)_{\mathcal{AH}_\mathcal{R}}(\hat{z_1}, ..., \hat{z_m}, f_1, \ldots, f_n) = \mathsf{true}.$$

Taking arbitrary elements $d_1 \in \mathrm{AD}^{b_1}, \ldots, d_n \in \mathrm{AD}^{b_m}$ that satisfy $d_1 \sqsupseteq \hat{z_1}, ..., d_m \sqsupseteq \hat{z_m}$, we obtain

$$(\mathsf{steq}\ s\ t)_{\mathcal{AH}_\mathcal{R}}(d_1, ..., d_m, f_1, ..., f_n) = \mathsf{true}.$$

By the construction of $\mathcal{AH}_\mathcal{R}$, for each f_i, there exists $u_i \in \mathcal{AT}_\Sigma$ such that $\mathcal{AH}_\mathcal{R}[\![u_i]\!] = f_i$. Define a substitution $\xi : \Delta \to \mathcal{T}_\Sigma$ as follows:

$$\xi \stackrel{def}{=} \{x_1 \mapsto d_1, ..., x_m \mapsto d_m, y_1 \mapsto u_1, ..., y_m \mapsto u_m\}.$$

Hence, we have

$$\mathcal{AQ}_\mathcal{R} \models \forall \varnothing\ ((\mathsf{steq}\ s\ t)\xi = \mathsf{true}).$$

■

Theorem 4.4.3 (**Completeness of Narrowing for $\mathcal{AH}_\mathcal{R}$**)
Let $\mathcal{R}$ be a program and $\exists\Delta(\mathsf{steq}\ s\ t = \mathsf{true})$ a query. If

$$\mathcal{AH}_\mathcal{R} \models \exists\Delta(\mathsf{steq}\ s\ t = \mathsf{true})$$

holds with a witness $\delta : \Delta \to \mathcal{AH}_\mathcal{R}$ then there exists

- a narrowing derivation $(\mathsf{steq}\ s\ t) \leadsto^*_\theta \mathsf{true}$ with an answer substitution $\theta : \Delta \to \mathcal{AT}_\Sigma(\Delta')$,
- an assignment $\phi : \mathcal{AH}_\mathcal{R} \to \mathrm{AD}$ that depends on δ and θ, and
- a substitution $\rho : Y \to \mathrm{AD}$ that depends on θ and ϕ

such that

$$\rho^\# \circ \theta = \phi \circ \delta.$$

Proof We will show that the following diagram commutes:

$$\begin{array}{ccccc} \Delta & = & \Delta & = & \Delta \\ {\scriptstyle\delta}\downarrow & & \downarrow{\scriptstyle\xi} & & \downarrow{\scriptstyle\theta} \\ \mathcal{AH}_\mathcal{R} & \xrightarrow{\phi} & \mathrm{D} & \xleftarrow{\rho^\#} & \mathcal{AT}_\Sigma(Y) \end{array}$$

where ξ is a mapping determined in the proof.

Suppose δ is a witness of $\exists\Delta(\mathsf{steq}(s,t) = \mathsf{true})$, i.e., $\delta : \Delta \to \mathcal{AH}_\mathcal{R}$ such that $\delta^\#(\mathsf{steq}(s,t)) = \mathsf{true}$. By Lemma 4.4.2, there exists an assignment

$$\xi : \Delta \to \mathrm{D} \text{ such that } \xi^\#(\mathsf{steq}\ s\ t) = \mathsf{true}.$$

By the Axiom of Choice, there exists a function Ψ such that

$$\Psi(\delta) = \xi.$$

Define a partial function $\phi : \mathcal{AH}_\mathcal{R} \to \mathrm{D}$ such that

$$\phi(h) \stackrel{def}{=} \begin{cases} \Psi(\delta)(x) & \text{if there exists } x \in \mathcal{V}(\Delta) \text{ such that } \delta(x) = h \\ \textit{undefined} & \text{otherwise.} \end{cases}$$

If there are many x satisfying the first clause of the above definition, we choose the minimal x with respect to the element's order in the sequence Δ. Then, clearly $\xi = \phi \circ \delta$. Since $(\mathsf{steq}\ s\ t)\xi \to^*_\mathcal{R} \mathsf{true}$, by the completeness of narrowing for rewriting, there exists a narrowing derivation $(\mathsf{steq}\ s\ t) \leadsto^*_\theta \mathsf{true}$ with an answer substitution $\theta : \Delta \to \mathcal{AT}_\Sigma(Y)$ such that $\theta \preccurlyeq \xi$, i.e. there exists a substitution

$$\rho : Y \to \mathrm{AD} \text{ such that } \rho^\# \circ \theta = \xi.$$

Hence $\rho^\# \circ \theta = \phi \circ \delta$. ■

Chapter 5

Conclusion

In this thesis, we have given axiomatic, algebraic, operational, and categorical semantics of first-order, interactive first-order, and higher-order functional-logic programming languages. We have established the correspondence between these semantics by using existing results on equational logic and proving the following *new* results:

- equivalence of the quotient term model and the least complete Herbrand model in algebraic semantics of many-sorted first-order FLP (Theorem 2.6.7),
- soundness of the translation from interactive functional-logic programs to CTRSs (Theorem 3.6.4, 3.6.5) in interactive first-order FLP,
- correspondence of the minimal applicative Herbrand model and conditional narrowing (Theorem 4.4.3) in simply-typed applicative FLP,

As a summary, we give intuitive interpretations of the constructs of functional-logic programming languages in several semantics we gave in this thesis.

First-order functional-logic programming language:

Axiomatic semantics

logic	many-sorted conditional equational logic
sorts	sorts
terms	terms
programs	universally quantified equations
queries	existentially quantified strict equations

Operational semantics

operation	conditional narrowing
sorts	sorts
terms	terms
programs	CTRSs
queries	steq-terms to be narrowed

Algebraic semantics

models	many-sorted Σ-algebras
sorts	indexed set by each sort
terms	elements in a Σ-algebra
programs	definitions of operations of a Σ-algebra
queries	questions asking values in a Σ-algebra for variables

Categorical semantics

models	categories with finite products
sorts	objects
terms	arrows
programs	equations between arrows
queries	questions asking values in a Σ-algebra for variables

Interactive first-order functional-logic programming language:
The interpretations of sort, term, program and query are the same as in the case of first-order functional-logic programming language.

Axiomatic semantics

logic	the computational metalanguage
sequential computation	nested let-terms

Operational semantics

operation	conditional narrowing
sequential computation	conditional part of a CTRS

Categorical semantics

models	cartesian closed categories with the strong monad for side-effects
sequential computation	composition of arrows

Simply-typed applicative functional-logic programming language:
Axiomatic semantics

logic	applicative conditional equational logic
sorts	types (including function types)
terms	applicative terms
programs	universally quantified applicative equations
queries	existentially quantified applicative equations

Operational semantics

operation	conditional narrowing
sorts	types (including function types)
terms	applicative terms
programs	CTRSs
queries	steq-terms to be narrowed

Algebraic semantics

models	typed Σ-algebras
sorts	indexed set by each sort
terms	elements (including function) in a typed Σ-algebra
programs	(higher-order) function definitions
queries	questions asking values in a Σ-algebra for variables

Bibliography

[AP95] P.M. Achten and M.J. Plasmeijer. The ins and outs of concurrent clean I/O. *Journal of Functional Programming*, 5(1):81–110, 1995.

[Bar84] H. Barendregt. *The Lambda Calculus, Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, revised edition, 1984.

[Bar96] H. Barendregt. Interactive functional programming. In *Proceedings of Second Fuji International Workshop on Functional and Logic Programming*, pages 192–193. World Scientific, 1996.

[Bar97] H. Barendregt. The impact of the lambda calculus in logic and computer science. *Bulletin of Symbolic Logic*, 3(2):181–215, June 1997.

[BK86] J. A. Bergstra and J. W. Klop. Conditional rewrite rules: Confluence and termination. *Journal of Computer and System Sciences*, 32(3):323–362, 1986.

[BS93] E. Barendsen and S. Smetsers. Conventional and uniqueness typing in graph rewrite systems. In *In Proceedings of 13th Conference on the Foundations of Software Technology and Theoretical Computer Science (FST & TCS13)*, volume 761 of *Lecture Notes in Computer Science*, pages 41–51, 1993.

[CF58] H.B. Curry and R. Feys. *Combinatory Logic, Volume I.* Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1958. Second printing 1968.

[CHS72] H.B. Curry, J.R. Hindley, and J.P. Seldin. *Combinatory Logic, Volume II.* Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1972.

[Cro93] R.L. Crole. *Categories for Types.* Cambridge Mathematical Textbook, 1993.

[Cur93] Curien, P.-L. *Categorical Combinators, Sequential Algorithms, and Functional Programming.* Birkhäuser, second edition, 1993.

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–320. The MIT Press/Elsevier, 1990.

[Fri85] L. Fribourg. SLOG: a logic programming language interpreter based on clausal superposition and rewriting. In *Proceedings of the 2nd IEEE Symposium on Logic Programming, Boston*, pages 172–184, 1985.

[GLMP91] E. Giovannetti, G. Levi, C. Moiso, and C. Palamidessi. Kernel-LEAF: A logic plus functional language. *Journal of Computer and System Sciences*, 42(2):139–185, 1991.

[GM85] J. Goguen and J. Meseguer. Completeness of many-sorted equational logic. *Houston Journal of Mathematics*, 11(3):307–334, 1985.

[GM86a] E. Giovannetti and C. Moiso. A completeness result for E-unification algorithms based on conditional narrowing. In *Proceedings of the Workshop on Foundations of Logic and Functional Programming, Lecture Notes in Computer Science 306*, pages 157–167, 1986.

[GM86b] J. A. Goguen and J. Meseguer. EQLOG: equality, types and generic modules for logic programming. In DeGroot and Lindstrom, editors, *Logic Programming: Functions, Relations and Equations*, pages 295–364. Prentice Hall, 1986.

[GM87] J. Goguen and J. Meseguer. Models and equality for logical programming. *LNCS 250*, pages 1–22, 1987.

[GMHGRA92] J.C. González-Moreno, M.T. Hortalá-González, and M. Rodríguez-Artalejo. Denotational versus declarative semantics for functional programming. *LNCS 626*, pages 134–148, 1992.

[Gor94] A. D. Gordon. *Functional Programming and Input/Output*. PhD thesis, University of Cambridge, 1994.

[GTEJ77] J. Goguen, J. W. Thatcher, E.G.Wangner, and J.B.Wright. Initial algebra semantics and continuous algebras. *Journal of the Association for Computing Machinery*, 24(1):68–95, 1977.

[GTW75] J. Goguen, J. Thatcher, and E. Wagner. Abstract data types as initial algebras and the correctness of data representations. In *Proceedings of Conference on Computer Graphics, Pattern Recognition and Data Structure*, pages 89–93, 1975.

[GTW76] J. Goguen, J. Thatcher, and E. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. Technical Report RC 6487, IBM T. J. Watson Research Center, 1976. Reprinted in: Yeh, R. (ed.): Current trends in programming methodology IV. Prentic-Hall 1987, pp. 80-149.

[Gun92] C.A. Gunter. *Semantics of programming languages*. The MIT Press, 1992.

[GWMF96] J. Goguen, T. Winkler, J. Meseguer, and K. Futatsugi. *Introducing OBJ*. Cambridge, 1996.

[Ham95] M. Hamana. Semantics of a functional-logic language. Master's thesis, University of Tsukuba, 1995.

[Han90] M. Hanus. Compiling logic programs with equality. In *Proceedings of the 2nd International Symposium on Programming Language Implementation and Logic Programming, Lecture Notes in Computer Science 456*, pages 387–401, 1990.

[Han94] M. Hanus. The integration of functions into logic programming: From theory to practice. *Journal of Logic Programming*, 19&20:583–628, 1994.

[Has97] M. Hasegawa. *Models of Sharing Graphs*. PhD thesis, University of Edinburgh, 1997.

[HKMN95] M. Hanus, H. Kuchen, and J.J. Moreno-Navarro. Curry: A truly functional logic language. In *Proceedings of ILPS'95 Workshop on Visions for the Future of Logic Programming*, pages 95–107, 1995.

[HNN+94] M. Hamana, T. Nishioka, K. Nakahara, A. Middeldorp, and T. Ida. A design and implementation of a functional-logic language based on applicative term rewriting systems. *Proceedings of WGSYM, Information Processing Society of Japan, 94-SYM-73, pp.25–32*, March 1994. In Japanese.

[Hul80] J. M. Hullot. Canonical forms and unification. *LNCS 87*, 1980.

[IO94] T. Ida and S. Okui. Outside-in conditional narrowing. *IEICE Transactions on Information and Systems*, E77-D(6), 1994.

[JMMGMA92] J.C. González-Moreno, M.T. Hortalá-González, and M. Rodríguez-Artalejo. On the completeness of narrowing as the operational semantics of functional logic programming. *LNCS 702*, pages 216–230, 1992.

[Kap84] S. Kaplan. Conditional rewrite rules. *Theoretical Computer Science*, 33(2):175–193, 1984.

[KKSdV96] J. R. Kennaway, J. W. Klop, M. R. Sleep, and F. J. de Vries. On comparing curried and uncurried rewrite system. *Journal of Symbolic Computation*, 21(1):15–39, 1996.

[Klo92] J.W. Klop. Term rewriting systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 1–116. Oxford University Press, 1992.

[Kow74] R. A. Kowalski. Predicate logic as a programming language. *Information Processing*, (74):569–574, 1974.

[Llo87] J.W. Lloyd. *Foundations of Logic Programming, second edition*. Springer-Verlag, second edition, 1987.

[LPB+87] G. Levi, C. Palamidessi, P.G. Bosco, E. Giovannetti, and C. Moiso. A complete semantics characterization of K-LEAF, a logic language with partial functions. In *Proceedings 1987 Symposium on Logic Programming*, pages 318–327, Rockville, MD, 1987. IEEE Computer Society Press.

[LS86] J. Lambek and P.J. Scott. *Introduction to Higher Order Categorical Logic*, volume 7 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, England, 1986.

[Mac71] S. Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1971.

[Man76] E.G. Manes. *Algebraic Theories*, volume 26 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1976.

[Mei92] K. Meinke. Universal algebra in higher types. *Theoretical Complete Science*, 100:385–417, 1992.

[MG85] J. Meseguer and J.A. Goguen. Initiality, induction, and computability. In M. Nivat and J. C. Reynolds, editors, *Algebraic methods in semantics*, pages 459–541. Cambridge University Press, 1985.

[MH94] A. Middeldorp and E. Hamoen. Completeness results for basic narrowing. *Applicable Algebra in Engineering, Communication and Computing*, 5(3/4):213–253, 1994.

[Mid90] A. Middeldorp. *Modular Properties of Term Rewriting Systems*. PhD thesis, Vrije Universiteit, Amsterdam, 1990.

[Mit96] J.C. Mitchell. *Foundations for Programming Languages*. The MIT Press, 1996.

[MNRA92] J.J. Moreno-Navarro and M. Rodríguez-Artalejo. Logic programming with functions and predicates: The language BABEL. *Journal of Logic Programming*, 12:191–223, 1992.

[Mog88] E. Moggi. Computational lambda-calculus and monads. LFCS ECS-LFCS-88-66, University of Edinburgh, 1988.

[Mog91] E. Moggi. Notions of computation and monads. *Information and Computation*, (93):55–92, 1991.

[NMI95] K. Nakahara, A. Middeldorp, and T. Ida. A complete narrowing calculus for higher-order functional logic programming. In *Proceedings of Seventh International Conference on Programming Languages: Implementations, Logics, and Programming 95 (PLILP'95), Lecture Notes in Computer Science 982*, pages 97–114, 1995.

[Pit95] A.M. Pitts. Categorical logic. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume VI, chapter ? Oxford University Press, 1995.

[Re91] J. Rees and W. Clinger (editors). Revised^4 report on the algorithmic language Scheme. *ACM Lisp Pointers IV*, July-September 1991.

[Sco71] D.S. Scott. Towards a mathematical semantics for computer languages. In J. Fox, editor, *Computers ans Automata*, pages 19–46. Polytechnic Institute of Brooklyn Press, 1971.

[SH97] T. Suzuki and M. Hamana. Logic programs as term rewriting systems. *Computer software*, 14(6):29–43, November 1997.

[SMI95] T. Suzuki, A. Middeldorp, and T. Ida. Level-confluence of conditional rewrite systems with extra variables in right-hand sides. In *Proceedings of the 6th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 914, pages 179–193, 1995.

[Suz98] T. Suzuki. *Studies on Conditional Narrowing for Rewrite Systems with Extra Variables.* PhD thesis, University of Tokyo, 1998.

[vBSB93] S. van Bakel, S. Smetsers, and S. Brock. Partial type assignment in left linear applicative term rewriting systems. In M. R. Sleep, M. J. Plasmeijer, and M. C. J. van Eekelen, editors, *Term Graph Rewriting: Theory and Practice*, pages 15–29. Wiley, Toronto, 1993.

[Wad90] P. Wadler. Comprehending monads. In *ACM Conference on Lisp and Functional Programming*, pages 61–78, Nice, France, June 1990.

[Wec92] W. Wechler. *Universal algebra for computer scientists.* Springer-Verlag, 1992.

[YALSM97] T. Yamada, J. Avenhaus, C. Loria-Saenz, and A. Middeldorp. Logicality of conditional rewrite systems. In *Proceedings of the 22nd International Colloquium on Trees in Algebra and Programming (CAAP'97)*, Lecture Notes in Computer Science 1214, pages 141–152, 1997.

Index

Σ-algebra 19
3-CTRS 23

algebraic 34
answer substitution 26
applicative constructor term . . . 80
applicative FLP 80
applicative Herbrand algebra . . . 83
applicative strict equation 80
arity 18
assignment 20
axiom 21

base type 78
built-in interactive axiom 65
built-in interactive function symbol 65

compact 34
complete Herbrand algebra 35
complete Herbrand universe 34
computational signature 53
computations 52
$\text{Con}_\perp$-tree 34
conditional equation 21
Conditional equational logic 21
conditional term rewriting system 23
confluent 24
constant symbol 18
constructor symbol 31
constructor term 31
context 23
conversion 24
cpo 34
CTRS 22

defined function symbol 31

empty 18
equivalent 74
existentially quantified equation 21

first-order FLP 31
function symbol 18, 31
function type 78

ground 18

initial algebra 20
interaction 13
interactive FLP 57
interactive FLP signature 57
interactive function symbol 66

Kleisli triple 55

linear 18

many-sorted first-order FLP . . . 31
minimal applicative Herbrand model 85
model 29, 44, 56
monad 55

normal form 24

normalizable 24
normalized 24, 28
null typing context 18

orthogonal 24

positions 23
program 80
properly oriented 24
proved term 18

query 9, 32, 81
quotient term model 30

raw terms 18
rewrite rule 23
right-stable 25

side-effects 59
signature 18
sorts 18
source sort 18
state 59, 62
strict equation 31
strong monad 55
structure 43
substitution 20
subterm 23

target sort 18
term 18
term algebra 20
terminating 24
theorem 21
total 34
type basis 78
typed signature 78
types 52
typing context 18

undefined 20

valid 29, 44
validity 56
value 52

well-typed term 18
witness 29