Term Graph Rewriting

syntax and semantics

IPA Dissertation Series, no. 2001-05

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

VRIJE UNIVERSITEIT

Term Graph Rewriting

syntax and semantics

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan
de Vrije Universiteit te Amsterdam,
op gezag van de rector magnificus
prof.dr. T. Sminia,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der Exacte Wetenschappen \ Wiskunde en Informatica
op dinsdag 6 maart 2001 om 13.45 uur
in het hoofdgebouw van de universiteit,
De Boelelaan 1105

door

Stefanus Cornelis Christoffel Blom

geboren te Rotterdam

Promotor:	prof.dr. J.W. Klop
Copromotor:	dr. Z.M. Ariola

Contents

Preface

The fields of Programming Languages and Rewriting Theory have in the past benefited from each other. For example, term rewriting systems have proven to be a useful tool for expressing specifications and reasoning about them. The lambda calculus, has proven useful in the study of parameter passing principles and provided a sound basis for the semantics of functional programming languages. In order to better model the implementations of programming languages, graphs are considered rather than terms. The oldest work on term graph rewriting concentrates on acyclic graphs. Recently, the focus is shifting towards cyclic graphs. In first order rewriting the results are excellent. The functional programming language Clean has an underlying model that is based on graph rewriting. This model provides the semantics for the language but it has one drawback: because a program is translated to a rewrite system plus an expression it is relatively hard to reason about the compilation process itself. If one uses higher order graph rewriting then it is possible to express functions as terms rather than as rewrite rules. As a result we can represent an entire program by a single term. This makes reasoning about the compilation process easier because the optimization of the program, which is the most complicated part of a compiler for a functional programming language, now can be expressed as a rewrite system. The Glasgow Haskell Compiler uses this approach.

Designing a confluent graph rewriting system, based on a confluent term rewrite system, is a non-trivial task if the graph rewrite system also has to satisfy efficiency conditions. In the first-order case it is often possible to find a confluent rewrite system that has the desired expressiveness. However, in the higher-order case there is a direct trade-off between expressiveness and confluence. This means that although working with non-confluent rewrite systems is inconvenient, we have to consider them.

In first order rewriting the important properties were termination (SN) and uniqueness of normal forms (UN). These properties guarantee that computation stops and that the answer is the same no matter how the computation was done. In the lambda calculus we also had meaningful infinite reductions. Rather than reducing terms to normal form, the Böhm Tree of a term was computed. The notion of Böhm tree extends that of normal form: if a term terminates then its Böhm tree is its normal form. But the Böhm tree can be computed for any term. Böhm trees have the same uniqueness property as normal forms: given two convertible terms

their Böhm trees are the same. We will also refer to this property as the uniqueness of infinite normal forms (UN^{∞}). The key observation is that confluence and uniqueness of normal forms are the same property in the presence of termination, but that confluence and uniqueness of infinite normal forms are not. Confluence implies uniqueness of infinite normal forms, but not the other way around. Thus, we searched for and found a new property that resembles confluence and that is equivalent to uniqueness of infinite normal forms: skew confluence. This notion is based on the intuition that some terms are better than others and that terms reduce to better terms. It states that if a term reduces to two other terms, the second of those terms can always be reduced to a term that is better than the first.

Chapter 1

Introduction

Graph rewriting is a rather intuitive subject, but unfortunately proofs can become very tedious. We have chosen to give an extensive informal introduction at the start, in order to make it easier to skip entire chapters of formal details later.

The introduction is split into four sections. The first two sections deal with the syntax and semantics of first and higher-order term graphs. We will introduce syntactic constructs, such as the let, μ, letrec and lambda-abstraction and for every syntactic construct that we add we will discuss the features of programming languages that it models. We will also discuss the expressive power of the constructs in terms of graphs and trees. In the third section we focus on the applications in the field of programming languages. Among others, we will consider the application that motivated the work in this thesis: *program transformations.* We will show that this application forces us to consider non-confluent rewrite systems in our treatment of semantics. In the fourth section we discuss our approach to the semantics of term graph rewriting. That is, we will discuss the notion of Böhm tree and how to generalize this notion to non-confluent rewrite systems, using the notion of *skew confluence.*

1.1 First-order term graphs

1.1.1 Term Rewriting Systems

Algebraic specifications are a well-known topic in computer science. These specifications consist of a signature, describing how we can build terms, and some equations that describe an equivalence relation on the terms. See for example the specification of the natural numbers with addition and multiplication in Table 1.1. More precisely, a *signature* consists of a set of function symbols with a fixed arity and an infinite set of variables. The set of *first order terms* over a signature allows only variables and function applications in the construction of terms.

If we replace the equal signs by left to right arrows then we get the rewrite rules of a *term rewriting system.* The idea is that by repeatedly matching a left-hand side

Table 1.1 Specification of the natural numbers with addition and multiplication

Signature	
0	constant
S	unary function symbol
$+, *$	binary function symbols
$X, Y, Z, \cdots$	variables
Equations	
$X + 0 = X$	
$X + S(Y) = S(X + Y)$	
$X * 0 = 0$	
$X * S(Y) = X + (X * Y)$	

of a rewrite rule and replacing it by the right-hand side we turn the equivalence into a computation or reduction sequence. For example,

$$S(0) + S(0) \to S(S(0) + 0) \to S(S(0)) \quad .$$

If we are not interested in the intermediate results, we will abbreviate this reduction sequence to

$$S(0) + S(0) \to\!\!\to S(S(0)) \quad .$$

The standard graphical representation of terms is by means of trees. In Fig. 1.1 we have drawn the picture of $2 + x$, which we refer to as the graph of $2 + x$. Note that we distinguish between the constant 2 and the variable x by drawing an arrow for the former and a labeled line for the latter. The reason for making this distinction is that we do not count a variable as a node. A variable is just a placeholder for something else. We think of the line with a variable at the end as a labeled edge that has a source node but not yet a destination node. The view of variables as unfinished nodes is especially important during the construction of a graph. Once the construction of a graph is finished, variables often behave just like constants, which are nodes. Therefore, variables are officially labels, but drawn just like nodes to help intuition. This difference will return in the formal definition of the graph of a term in Chap. 3.

1.1.2 Sharing

Term rewriting isn't very efficient. The reason is that terms often contain the same sub-term many times. For example, $(1+2)+(1+2)$ contains the sub-term $(1+2)$

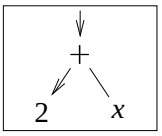


Figure 1.1: The graph of $2 + x$

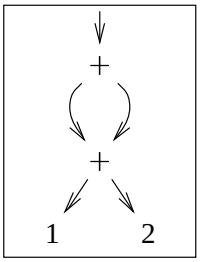


Figure 1.2: The graph of $\mathsf{let}\ x = 1+2\ \mathsf{in}\ x+x$

twice. It is not efficient to write these sub-terms down many times. It is much better to write it down once and *share* that single occurrence. For example, we could represent $(1+2)+(1+2)$ by

$$\mathsf{let}\ x = 1+2\ \mathsf{in}\ x+x\ .$$

We can also express this sharing in pictures. In Fig. 1.2 we have drawn the graph of this term.

Using the let syntax, we not only need less space to write down terms, but we can also evaluate them much more efficiently because instead of reducing each occurrence of the sub-term separately, we can reduce the shared occurrence once. For example, given $(1+2)+(1+2)$ an ordinary term rewrite system would add 1 and 2 then add 1 and 2 again and finally add 3 and 3. The efficient way of doing this computation is to add 1 and 2, remember the result and then add 3 and 3. That is, using the let construct we have the following reduction sequence:

$(1+2)+(1+2)$	$\to$	$\mathsf{let}\ x = 1+2\ \mathsf{in}\ x+x$	recognize the sharing,
	$\to$	$\mathsf{let}\ x = 3\ \mathsf{in}\ x+x$	compute $1+2$,
	$\to$	$3+3$	fill in the result,
	$\to$	6	and compute the final result.

Even though this reduction contains more steps than the reduction in the term rewrite system, we consider this reduction more efficient because it uses only two additions instead of three. In Fig. 1.3 we have done the reduction in the graph. From left to right we do two steps. In the first step we add 1 and 2 and replace the plus by the resulting 3. In the second step we add 3 and 3 and replace the plus by the resulting 6. If we translate this reduction sequence to terms then we get:

$$\mathsf{let}\ x = 1+2\ \mathsf{in}\ x+x \to \mathsf{let}\ x = 3\ \mathsf{in}\ x+x \to 6\ .$$

An important characteristic of the graphs obtained from terms with let is that they contain no cycles. Moreover, terms with let are sufficient to represent this class of graphs. That is, given any *directed acyclic graph* (dag for short) we can find a term with let that represents it. The term with let that represents a certain

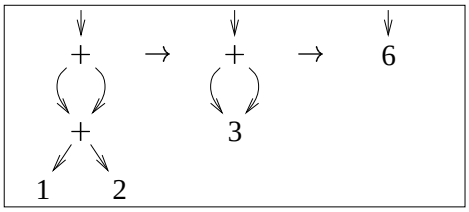


Figure 1.3: Shared evaluation of $(1+2)+(1+2)$

graph is not uniquely determined. For example, Fig. 1.2 is not only the graph of $(\mathsf{let}\ x = 1+2\ \mathsf{in}\ x+x)$, but also that of $(\mathsf{let}\ z = 1+2\ \mathsf{in}\ z+z)$, $(\mathsf{let}\ y = 1\ \mathsf{in}\ \mathsf{let}\ x = y+2\ \mathsf{in}\ x+x)$, $(\mathsf{let}\ z = 2\ \mathsf{in}\ \mathsf{let}\ x = 1+z\ \mathsf{in}\ x+x)$ and many more terms with let.

1.1.3 Recursion

In the theory of functional programming languages infinite objects play an important role, especially infinite trees. Some of these infinite trees have a finite representation. The simplest form of finitely representable infinite trees are the *recursively defined trees*. For example, the infinite list of ones can be recursively defined as the list that starts with a 1 and is followed by the list itself. By using the μ-construct we can express this in a term:

$$\mu x.1 :: x \ ,$$

where we use the infix operator :: as the cons operator on lists. The graph of this term is drawn in Fig. 1.4.

Like the let construct the μ-construct may occur anywhere in a term. Its general form is $\mu x.M$ where every free occurrence of x in M is bound by the μx. On μ-terms we have the following rewrite rule:

$$\mu x.M \underset{\mu}{\rightarrow} M[x := \mu x.M] \ ,$$

where the $M[x := N]$ denotes substituting N for every free occurrence of x in M.

A special μ-term is $\mu x.x$. If we follow the intuition of recursively defined trees then this is the term that is recursively defined as itself. This means that the intuition of this term is "undefined". As a matter of fact it is a special kind of "undefined", which in recent years has been denoted with the constant $\bullet$, called *black hole*. We have chosen to treat the terms $\mu x.x$ and $\bullet$ in the same way. Thus, we define the graph of $\mu x.x$ as that of $\bullet$. See Fig. 1.5.

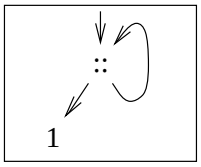


Figure 1.4: The graph of $\mu x.1 :: x$

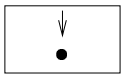


Figure 1.5: the graphs of $\mu x.x$ and $\bullet$

1.1.4 Letrec

The μ-construct is powerful in the sense that it can be used to represent all recursively defined infinite terms, or regular trees (see [Cou83]). This implies that the graphs that can be represented as μ-terms represent all recursively defined infinite trees. However, these representations are often not very efficient in the sense that they use more nodes than strictly necessary. For example, the graphs in Fig. 1.6 all represent the same infinite tree. Graph (a) is represented by the μ-term

$$F(\mu x.A(B(x)), \mu y.B(A(y))) \quad .$$

Graph (b) is slightly more efficient than (a), but we cannot represent it with only μ; we also need let to represent this graph:

$$\mathsf{let}\ x = \mu y.A(B(y))\ \mathsf{in}\ F(x, B(x)) \quad .$$

Graph (c) is the most efficient representation, but this graph cannot be represented by μ-terms with or without let. In order to represent this graph, we need the letrec. Using the letrec we can represent this graph by:

$$\langle F(x,y) \mid x = A(y), y = B(y) \rangle \quad .$$

The general form of the letrec is

$$\langle M \mid x_1 = M_1, \cdots, x_n = M_n \rangle \quad .$$

We will refer to M as the external part of the letrec and to the bindings $x_1 = M_1, \cdots, x_n = M_n$ as the internal part, or the declarations. We will often abbreviate this with $\langle M \mid D \rangle$. In the literature the letrec often occurs in a different syntactic form. For example,

$$M \text{ where } x_1 = M_1, \cdots, x_n = M_n$$

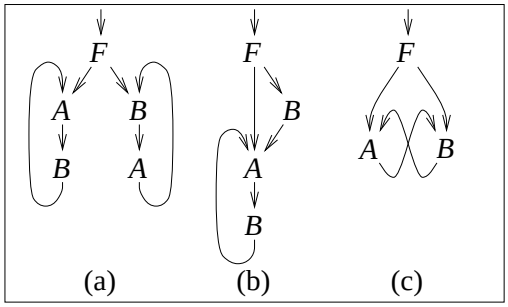


Figure 1.6: Three graphs that represent the same infinite term

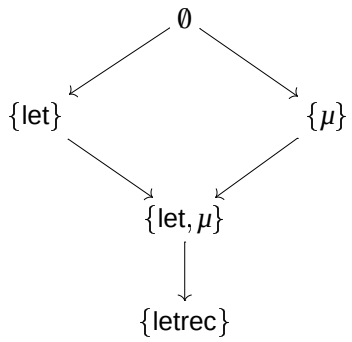


Figure 1.7: Expressiveness hierarchy of graph semantics

and

$$\text{letrec } x_1 = M_1, \cdots, x_n = M_n \text{ in } M \quad .$$

In Chapter 4 we study the classes of graphs that can be represented by (combinations of) the various constructs we have introduced so far. These classes form a hierarchy, which is drawn in Fig. 1.7. In this figure we have represented classes of terms by sets of constructs that are added to the basic variable and function application. An arrow in the picture means that the class at the source end of the arrow is strictly included in the class at the destination end.

1.1.5 Unwinding

One of the semantics of a graph is its unwinding. The unwinding of a graph is a possibly infinite tree, which can be obtained in several different ways. When we give the formal definition in Chap. 3, we will use a direct construction that involves paths in a graph. For now, we will describe the unwinding as the limit of the unwinding process. One step in the unwinding process is as follows: first, we choose a node with two or more in-going edges. Then, we add a copy of this node to the graph. Finally, we redirect some of the edges going to the original node to the newly created copy. To ensure that we do not introduce nodes that are not accessible from the root, we must do the redirection in such a way that there is at least one in-going edge to both the original and the copy. Fig. 1.8 illustrates this process. The unwinding of a graph is the possibly infinite tree that is

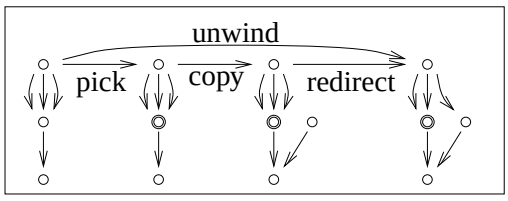


Figure 1.8: A single unwinding step.

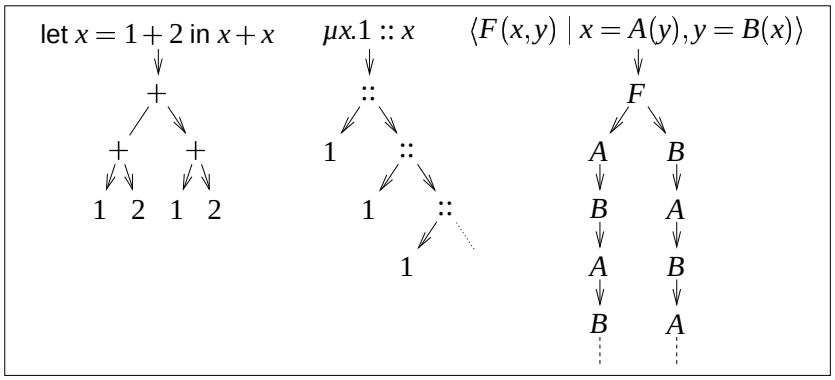


Figure 1.9: Examples of unwindings.

formed at the root of the graph, during a "fair" unwinding sequence. An unwinding sequence is "fair" if every node that has two or more in-going edges will eventually be chosen as the node that will be copied. For finite acyclic graphs, we have that every unwinding sequence is finite and hence that every unwinding sequence is "fair".

In Fig. 1.9 we have drawn the unwindings of some terms that we have used as examples before: let $x = 1+2$ in $x+x$, $\mu x.1 :: x$ and $\langle F(x,y) \mid x = A(y), y = B(x)\rangle$. The graphs of these terms are drawn in Fig. 1.2, 1.4 and 1.6(c), respectively.

The graphs that have the same unwinding can be characterized as the graphs that are *bisimilar*. Two graphs are bisimilar if we can play the following game without loosing. A state of the game is a pair of nodes, that are labeled with the same function symbol. The initial state of the game is the pair of root nodes of the graphs. Our opponent picks one node in the pair and one argument of that node. If that argument is a variable and the corresponding argument of the other node is the same variable we win otherwise we loose. If the argument of the chosen node is a node and the argument of the other node is also a node, such that the pair of arguments is a legal state the game continues in that state. Otherwise we loose.

One of the important properties of the unwinding of a graph is that it is the unique tree that is bisimilar with that graph. Thus, since they have the same unwinding all graphs in Fig. 1.6 are bisimilar.

Another way of constructing the unwinding of a graph is by means of rewriting the terms that represent graphs. The μ-rule is an example of a rule that expresses unwinding. For the let and letrec we can, for example, use the following rules:

$$\begin{array}{lll} \mathsf{let}\ x = M\ \mathsf{in}\ N & \to & N[x := M] \\ \langle M_0 \mid \underbrace{x_1 = M_1, \cdots, x_n = M_n}_{D}\rangle & \to & M_0[x_1 := \langle M_1 \mid D\rangle, \cdots, x_n := \langle M_n \mid D\rangle] \end{array}$$

In sections 7.1 and 8.1 rewrite systems for unwinding will be treated in detail.

Table 1.2 Combinatory Logic

$$\begin{array}{ll} Ix & \to x \\ Kxy & \to x \\ Sxyz & \to (xz)\,(yz) \end{array}$$

1.2 Higher-order term graphs

1.2.1 Lambda Calculus

One of the goals of this work is to provide a theoretically sound framework for reasoning about program optimization in particular and program transformations in general. So far, we have modeled the execution of a program by encoding the declarations of the program into a term rewrite system, which is used to reduce the main expression of the program. Another option is to encode the entire program into a term that is then reduced using a "universal" rewrite system. For reasoning about program execution both approaches work fine. However, reasoning about transformations on the translation of declarations to a rewrite system means that we have to reason about the transformation of a rewrite system. Whereas, reasoning about the translation of the program to a term means equational reasoning about a term. The latter is more convenient and better developed.

One of the properties of a "universal" rewrite system is that it is capable to encode functions as terms. For example, we must be able to encode the function

$$\texttt{twice} : x \mapsto x + x \ .$$

A typical example of a "universal" rewrite system is *Combinatory Logic* (CL), which is the term rewrite system given in Table 1.2, and the *lambda calculus* ([Bar84]), which is a *higher order* rewrite system. In CL we can encode the function `twice` as

$$S(+)I \ ,$$

where the parentheses around the $+$ turn it from an infix operator to a prefix operator. That is, $(+)\,1\,2 \equiv 1+2$. If we apply the encoding of `twice` to 2 then we get the following evaluation:

$$S(+)I2 \to 2 + (I2) \to 2+2 \to 4 \ .$$

In the lambda calculus we can encode the function `twice` as

$$\lambda x.x + x \ .$$

If we apply this encoding to 2 then we get:

$$(\lambda x.x + x)\,2 \underset{\beta}{\to} 2+2 \to 4 \ .$$

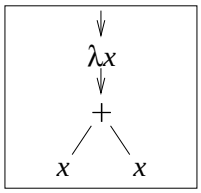


Figure 1.10: Encoding of the function `twice`

Note that because of the I in the encoding in Combinatory Logic we need an extra step in the reduction. If you compare the reduction of the encoding to the evaluation done in an actual implementation then these extra steps do not correspond to anything: they are really "extra". Also, the encoding in the lambda calculus is syntactically much closer to the real program. Therefore, we will work with the lambda calculus rather than Combinatory Logic.

In the example of the encoding, we have used a lambda calculus that we extended with natural numbers and addition. The pure lambda calculus has a signature that consists of variables and a single binary function symbol, called the application, which is denoted by both the function symbol @ and juxtaposition. That is, $@(M,N)$ and MN are two different notations for the same term. Lambda terms are formed from variables, function applications and the lambda abstraction:

$$\lambda x.M \ ,$$

where the x can be any variable and the M any lambda term. The most important rewrite rule of the lambda calculus is the β-rule:

$$(\lambda x.M)\,N \xrightarrow[\beta]{} M[x := N] \ .$$

It is simple to draw a picture of a lambda term. All one has to do is treat λx as a unary function symbol. In this way we have drawn the picture of the encoding of the function `twice` in Fig. 1.10.

1.2.2 Lambda Calculus and Sharing

The β-rule has one problem: if the bound variable occurs many times then the argument is copied many times in the reduction step. It is much more efficient to

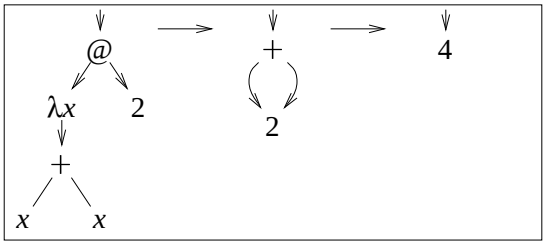


Figure 1.11: Graph reduction of `twice 2`

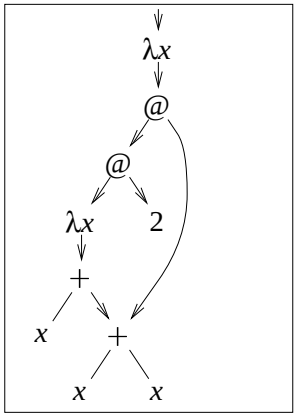


Figure 1.12: A confusing lambda graph with labels

share the argument rather than copy it. That is, instead of putting a substitution on the right-hand side we put a let:

$$(\lambda x.M)\,N \xrightarrow[\beta\mathsf{let}]{} \mathsf{let}\ x = N\ \mathsf{in}\ M \quad .$$

For `twice2` this leads to the reduction sequence depicted in Fig. 1.11. In this figure we have drawn lambda graphs by treating abstractions as function symbols. While this is common practice for trees, it is confusing in arbitrary graphs. For example, if we consider the graph in Fig. 1.12 then we have a graph with a single β-redex, which we want to reduce by replacing variables x with pointers to the 2. But, which variables should we replace? It is clear that we should replace the leftmost x, but how about the two other occurrences?

We can avoid this confusion by representing bound variables with arrows called *back-pointers*. Instead of labeling nodes with λx and using an x as a bound instance, we use nodes labeled λ and arrows, called *back-pointers*, to the lambda nodes to represent abstraction. To distinguish between arrows that denote arguments and arrows that denote back-pointers, we follow the convention that if an arrow points to a lambda node from above then it denotes an argument and if it points to it from below then it denotes a back-pointer. An example of a lambda graph with back-pointers is drawn in Fig. 1.13. In this graph the argument of the lambda is a back-pointer, both arguments of the top application node are normal pointers, as is the first argument of the second application node, and the second argument of the second application node is a back-pointer.

Apart from the confusion about which variables are bound, there is another problem: it is possible to have back-pointers from anywhere in the graph to anywhere else. This is very unlike what we have in terms, where an abstraction can only bind variables in its own sub-term. To reflect this sub-term relation in graphs, we will add the notion of *scope*. With every lambda node we will associate a set of nodes (of which that lambda node is a member), called the scope of the lambda node. We then add the restriction that a back-pointer from a node to a lambda node

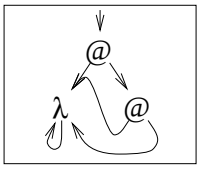


Figure 1.13: A lambda graph with back-pointers

is only allowed if that node is in the scope of the lambda node. In pictures, we will draw a line indicating the boundary of the scope. The lambda node to which the scope belongs is drawn on the boundary and every other node in the scope in drawn inside the boundary. As an example we have drawn the graph of `twice 2` in Fig. 1.14. To make scopes behave like sub-terms we also require that scopes are properly nested and that pointers can only traverse the scope boundary in one direction: from the inside to the outside. These restrictions rule out the graphs in Fig. 1.15. From left to right we have a back-pointer illegally crossing a scope boundary, a forward pointer illegally crossing a scope boundary and improper nesting. These restrictions also imply that only nodes in the scope of a lambda node are allowed to have back-pointers to that lambda node and that from the outside of a scope we can only have a pointer to the lambda node of that scope.

1.2.3 Lambda Calculus and Recursion

We have seen how we can use cycles to encode recursively defined objects. Cycles are also very useful in the representation of recursively defined functions. Traditionally recursively defined functions have been encoded in CL and the lambda calculus by means of *fixed-point combinators*. A fixed point combinator is a term Y such that

$$Y\,F = F\,(Y\,F) \quad .$$

In the lambda calculus such terms can be found, for example

$$Y_f \equiv \lambda f.(\lambda x.\,f\,(x\,x))\,(\lambda x.\,f\,(x\,x)) \quad .$$

If we also have the μ-construct then we can define

$$Y_\mu = \lambda f.\mu x.\,f\,x \quad .$$

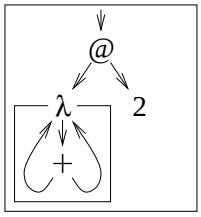


Figure 1.14: The graph of `twice 2`

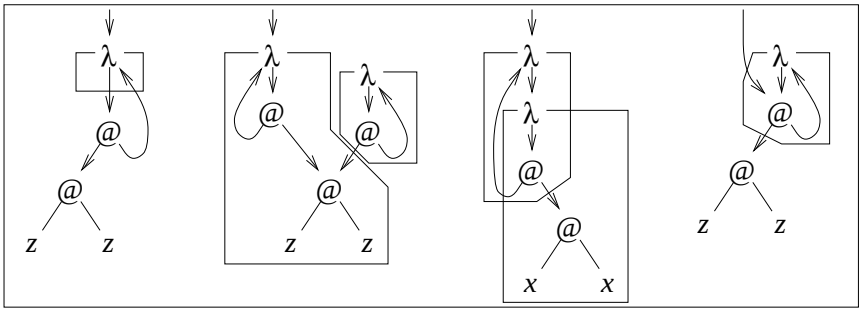


Figure 1.15: Lambda graphs with incorrect scopes

This is another fixed point combinator:

$$\begin{array}{rcl}(\lambda f.\mu x.f\,x)\,F & = & \mu x.F\,x \\ & = & F\,(\mu x.F\,x) \\ & = & F\,((\lambda f.\mu x.f\,x)\,F)\end{array}$$

The difference between the two fixed-point combinators can be shown in a little example. Let us consider the factorial function

$$\texttt{fac}\,n = \text{if } n = 0 \text{ then } 1 \text{ else } n * (\texttt{fac}\,(n-1)) \quad .$$

Using any fixed-point combinator we can encode `fac` as:

$$Y\,(\lambda f\,n.\text{if } n = 0 \text{ then } 1 \text{ else } n * (f\,(n-1))) \quad .$$

Note that if we use Y_μ then we can reduce the encoding of `fac` to

$$\mu x.\lambda n.\text{if } n = 0 \text{ then } 1 \text{ else } n * (x\,(n-1)) \quad .$$

This is much closer to the definition we gave. If we use the letrec then we can encode this function as a declaration and stay even closer to the definition:

$$\langle \cdot \mid \texttt{fac} = \lambda n.\text{if } n = 0 \text{ then } 1 \text{ else } n * (\texttt{fac}\,(n-1)), \cdots \rangle \quad .$$

1.2.4 Unwinding of Lambda Graphs

In the first-order case we had two ways to define the unwinding of a term with letrec: by means of unwinding the graph and by means of rewriting the term. The first definition yields an infinite tree and the second an infinite term. The same can be done for higher-order terms. In the first-order case we have that the set of infinite trees and infinite terms are isomorphic and that both definitions are the same up to isomorphism. However, in the higher order case the set of infinite trees is not isomorphic to the set of infinite terms: a single infinite term (infinite trees with labels) corresponds to many infinite trees with scopes. Thus, the unwinding of a graph yields an infinite tree with scopes and the rewriting of a term yields an

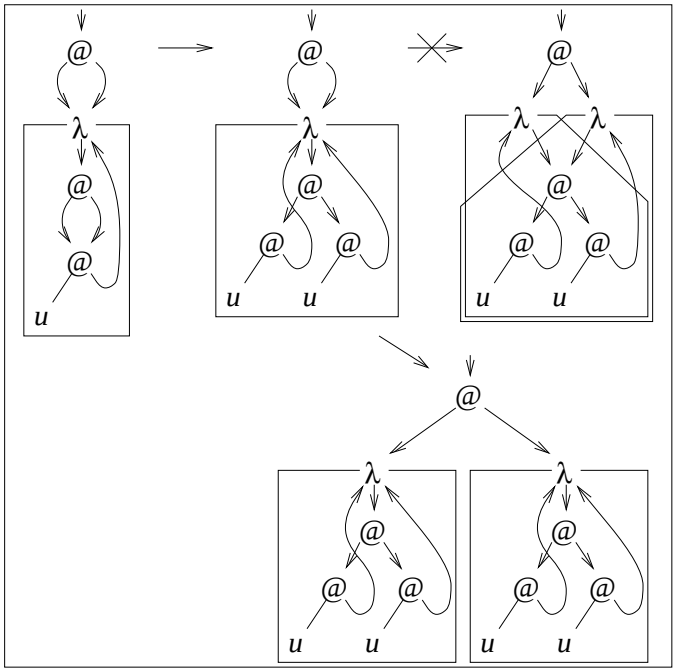


Figure 1.16: Unwinding the graph of let $y = (\lambda x.\text{let } z = ux \text{ in } zz)$ in yy

infinite tree with labels. Nevertheless, if we translate the scoped unwinding of a higher-order term to an infinite tree with labels we obtain the infinite normal form of the term.

Obtaining the unwinding by means of rewriting works exactly the same for higher-order terms as for first-order terms. Unwinding higher-order graphs is a little bit more difficult. If we choose to duplicate a lambda node then we do not copy just the lambda node, but we copy the entire scope of the lambda node. This is necessary because we need to preserve the proper structure of the graph. Fig. 1.16 shows what goes wrong if we only copy the lambda node and also shows a proper unwinding sequence for the graph of let $y = \lambda x.\text{let } z = xx \text{ in } uz \text{ in } yy$. In Fig. 1.17 we have drawn the scoped unwinding of let $z = \lambda y.y$ in $\lambda x.z$ on the left. The infinite normal form of this term is $\lambda xy.y$. We have drawn this term in the middle and on the right we have drawn the graph of this term. Note that the the graph of the infinite normal form is not the same as the scoped unwinding of the graph.

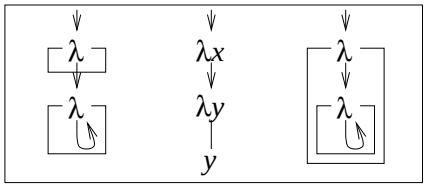


Figure 1.17: The unwindings of let $z = \lambda y.y$ in $\lambda x.z$

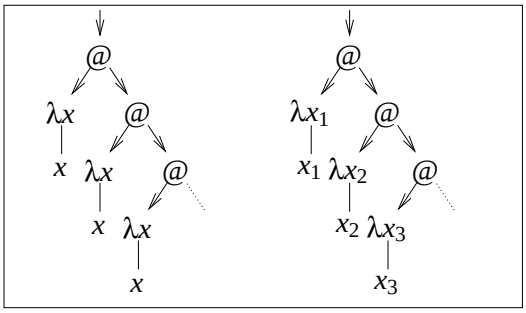


Figure 1.18: Examples of regular lambda trees

A natural question to ask is which infinite trees can be obtained as the unwinding of a term with letrec. In the first-order case the answer was all regular trees. We would like to give the same answer here, but what does it mean for a higher-order tree to be regular? In the remainder of this section, we will answer that question. Because the answer has no impact on the rest of the thesis and because it is a rather complicated answer, some readers may wish to skip to the next section.

In the first-order case a term is regular if the set of all sub-terms is finite. We will try to use this definition in the higher-order case as well, but taking sub-terms is much more difficult in the higher-order case because of bound variables or back-pointers.

For labeled trees we can just forget that some variables are bound by lambda abstractions, but this leads to problems. For example, in Fig. 1.18 we have drawn two alpha convertible infinite lambda trees. Both trees are the unwinding of $\langle x \mid x = yx, y = \lambda x.x \rangle$. However, the left graph has a finite number of subtrees and the right graph has infinitely many subtrees. Worse than that, the tree in Fig. 1.19 has only a finite number of subtrees, but it is not the unwinding of a term with letrec.

To prove that this tree is not the unwinding of a term with letrec, we must

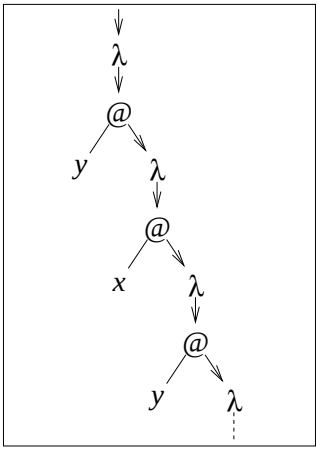


Figure 1.19: Example of a lambda tree

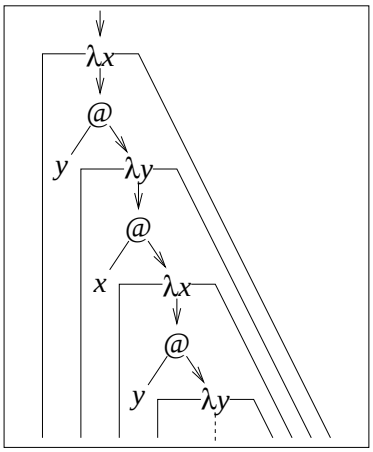


Figure 1.20: A labeled tree with scopes.

observe that the scoped unwinding preserves the depth of nesting of scopes. That is, if the maximal depth of nesting of the scopes in a graph is n then it is at most n in the scoped unwinding of that graph. (It can be less if the maximal nesting depth occurs in an inaccessible part of the graph.) With this observation we can explain why we cannot get the graph in Fig. 1.19 as the unwinding of a term with letrec. The only legal scoping of this tree is given in Fig. 1.20. Because this scoping is nested infinitely deep it is impossible that this tree was obtained as the unwinding of a term with letrec. To prove that this is the only possible scoping let us consider an arbitrary node in the graph. We must prove that this node is in the scope of every lambda node above it. We will do so by induction on the number of lambda nodes above the given node. If there are no lambda nodes above the given node then we are done. If there is a lambda node above us then the nearest lambda node above us binds a variable that is somewhere below the given node. Hence, we must be in the scope of this lambda node. By induction hypothesis this lambda node has to be in the scope of every lambda node above it. Because scopes have to be properly nested this means that the given node is also in the scope of every lambda node above the lambda node and thus in the scope of every lambda node above it.

The conclusion is that taking sub-trees of labeled trees doesn't work. The same holds for sub-terms of trees with de Bruijn labels. In Fig. 1.21 we have drawn the same term as in Fig. 1.19 but this time with de Bruijn labels.

The solution is to consider sub-trees of scoped trees. At first sight, taking sub-trees of scoped trees looks impossible because back-pointers would be left without a lambda to point to. This problem can be overcome by introducing nodes labeled ν that are also allowed to bind. We then define the sub-tree at a certain point as the tree starting with as many ν-nodes as we have scopes at that point followed by the real sub-tree at that point, where we connect every back-pointer to a node that is not in the sub-tree to the corresponding ν-node. See for example Fig. 1.22, where we have draw the graph of $x\lambda xy.yx$ and indicated all sub-trees. Note that we take

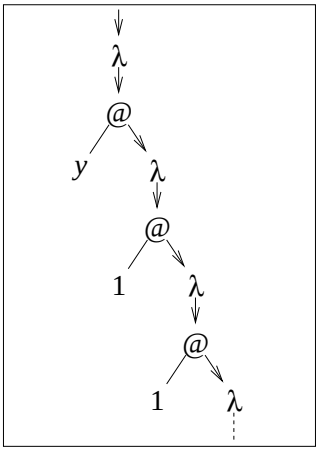


Figure 1.21: An infinite lambda term with de Bruijn indices

sub-trees at edges rather than sub-tree at nodes. Using this notion of subtree we can define regular lambda-trees. We conjecture that the set of regular lambda-trees is precisely the set of lambda-trees that can be obtained as the unwinding of terms with letrec.

1.3 Programming Languages

In the past sections we have often motivated the introduction of a new concept by referring to an application in the field of programming languages. In this section we will elaborate on these applications. More precisely, we will show how to use rewriting on terms with letrec to describe program execution and program

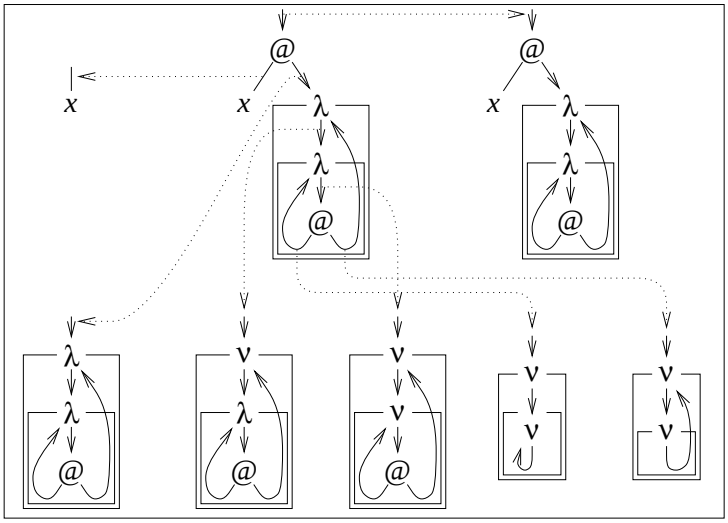


Figure 1.22: A lambda-tree and all its sub-trees.

transformations. We will also show the limitations of this approach.

1.3.1 Program Execution

We have given many examples, where we reduced a term and claimed that the reduction modeled the execution of the program. We will now discuss the modeling of program execution in detail. Let us consider the term

$$\langle F(xy,xy) \mid x = (\lambda xy.x)A, y = (\lambda x.xx)(\lambda x.xx)\rangle \quad .$$

We can think of this term as a program in a functional programming language. Three possible implementations of a functional programming language are *call-by-name, call-by-need* and *call-by-value.* By using different rewrite rules we can model these implementations.

The simplest implementation is call-by-name. To evaluate a function call using call-by-name we pass the expressions that are given as arguments and start to evaluate the body of the function. Every time we need an argument we make a copy of the expression that was passed for that argument and evaluate it. For our example we can model this with the following reduction sequence:

$$\begin{aligned}
&\langle F(\underline{x}y,xy) \mid x = (\lambda xy.x)A, y = (\lambda x.xx)(\lambda x.xx)\rangle \\
&\xrightarrow[\text{es}]{} \quad \langle F(\underline{(\lambda xy.x)Ay},xy) \mid x = (\lambda xy.x)A, y = (\lambda x.xx)(\lambda x.xx)\rangle \\
&\xrightarrow[\beta]{}^{2} \quad \langle F(A,\underline{x}y) \mid x = (\lambda xy.x)A, y = (\lambda x.xx)(\lambda x.xx)\rangle \\
&\xrightarrow[\text{es}]{} \quad \langle F(A,\underline{(\lambda xy.x)Ay}) \mid x = (\lambda xy.x)A, y = (\lambda x.xx)(\lambda x.xx)\rangle \\
&\xrightarrow[\beta]{}^{2} \quad \langle F(A,A) \mid x = (\lambda xy.x)A, y = (\lambda x.xx)(\lambda x.xx)\rangle = F(A,A)
\end{aligned}$$

Similar, but much more efficient is call-by-need. To evaluate a function call using call-by-need we pass the expressions that are given as arguments and start to evaluate the body of the function. The first time we need a particular argument we evaluate that argument and store the result. If we need the same argument again, we use the stored result. For our example we can model this with the following reduction sequence:

$$\begin{aligned}
&\langle F(xy,xy) \mid x = \underline{(\lambda xy.x)A}, y = (\lambda x.xx)(\lambda x.xx)\rangle \\
&\xrightarrow[\beta\circ]{} \langle F(xy,xy) \mid x = \underline{\langle \lambda y.x' \mid x' = A\rangle}, y = (\lambda x.xx)(\lambda x.xx)\rangle \\
&\xrightarrow[\text{im}]{} \langle F(\underline{x}y,xy) \mid x = \lambda y.x', x' = A, y = (\lambda x.xx)(\lambda x.xx)\rangle \\
&\xrightarrow[\text{es}]{} \langle F(\underline{(\lambda y.x')y},xy) \mid x = \lambda y.x', x' = A, y = (\lambda x.xx)(\lambda x.xx)\rangle \\
&\xrightarrow[\beta\circ]{} \langle F(\langle x' \mid y' = y\rangle,\underline{x}y) \mid x = \lambda y.x', x' = A, y = (\lambda x.xx)(\lambda x.xx)\rangle \\
&\xrightarrow[\text{es}]{} \langle F(\langle x' \mid y' = y\rangle,\underline{(\lambda y.x')y}) \mid x = \lambda y.x', x' = A, y = (\lambda x.xx)(\lambda x.xx)\rangle \\
&\xrightarrow[\beta\circ]{} \langle F(\langle x' \mid y' = y\rangle,\langle x' \mid y' = y\rangle) \mid x = \lambda y.x', x' = A, y = (\lambda x.xx)(\lambda x.xx)\rangle \\
&= \quad F(A,A)
\end{aligned}$$

Another name for call-by-need is *lazy evaluation.* Call-by-value is completely different from call-by-name and call-by-need. To evaluate a function call using call-by-need we first evaluate each of the arguments and only after having successfully

evaluated the argument we will start evaluating the body of the function. For our example we can model this with the following reduction sequence:

$$\begin{array}{l}\langle F\,(xy,xy) \mid x = (\lambda xy.x)\,A, y = \underline{(\lambda x.xx)\,(\lambda x.xx)}\rangle \\ \quad \xrightarrow[\beta]{} \langle F\,(xy,xy) \mid x = (\lambda xy.x)\,A, y = \underline{(\lambda x.xx)\,(\lambda x.xx)}\rangle \\ \quad \xrightarrow[\beta]{} \cdots\end{array}$$

1.3.2 Program Transformations

One of the purposes of program transformation is the optimization of programs. For example, let us consider the program

$$\langle\, f\,\mathtt{true} \mid\ f = \lambda x.x :: (f\,(\mathtt{not}\,x))\rangle\ \ ,$$

where we have the predefined rewrite rules

$$\begin{array}{l}\mathtt{not\,true}\ \ \to \mathtt{false} \\ \mathtt{not\,false} \to \mathtt{true}\end{array}$$

This program computes an alternating list of booleans starting with `true`. The same infinite list is computed by the program

$$\langle\, f\,\mathtt{true} \mid\ f = \lambda x.\langle y \mid y = x :: \mathtt{not}\,x :: y\rangle\rangle\ \ .$$

However, the performance of the two programs is radically different: the first program needs an infinite number of β-step to compute the list, while the second terminates after a single β-step. In several steps, we will now transform the first program into the second.

As the first step in the transformation we *inline* the definition of f into itself. That is, we substitute the definition of f for an occurrence of f in the definition of f and we let this substitution be followed by a β-step.

$$\begin{array}{ll}\langle\, f\,\mathtt{true} \mid\ f = \lambda x.x :: (f\,(\mathtt{not}\,x))\rangle & \\ \xrightarrow[\mathrm{cs}]{} & \langle\, f\,\mathtt{true} \mid\ f = \lambda x.x :: ((\lambda x.x :: (f\,(\mathtt{not}\,x)))\,(\mathtt{not}\,x))\rangle \\ \xrightarrow[\beta]{} & \langle\, f\,\mathtt{true} \mid\ f = \lambda x.x :: (\mathtt{not}\,x) :: (f\,(\mathtt{not}\,(\mathtt{not}\,x)))\rangle\ \ .\end{array}$$

Because we know that for any x it holds that

$$(\mathtt{not}\,(\mathtt{not}\,x)) = x\ \ ,$$

the next step in the transformation is

$$\begin{array}{l}\langle\, f\,\mathtt{true} \mid\ f = \lambda x.x :: (\mathtt{not}\,x) :: (f\,(\mathtt{not}\,(\mathtt{not}\,x)))\rangle \\ \quad = \langle\, f\,\mathtt{true} \mid\ f = \lambda x.x :: (\mathtt{not}\,x) :: (f\,x)\rangle\ \ .\end{array}$$

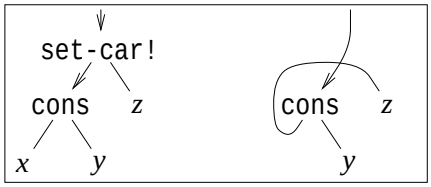


Figure 1.23: The function `set-car!` in Scheme

In this last term we can now see that a function is recursively called with the same arguments. This allows us to replace that particular function call with a cycle using the following axiom:

$$\langle M \mid f = \lambda x.C[f\,x], D\rangle = \langle M \mid f = \lambda x.\langle y \mid y = C[y]\rangle, D\rangle \quad .$$

Thus, the last step in the transformation is:

$$\begin{aligned}&\langle f\,\mathtt{true} \mid f = \lambda x.x :: (\mathtt{not}\,x) :: (f\,x)\rangle \\ &\quad = \langle f\,\mathtt{true} \mid f = \lambda x.\langle y \mid y = x :: (\mathtt{not}\,x) :: y\rangle\rangle \quad .\end{aligned}$$

Much more about the use of terms with letrec in the implementation of the pure functional programming language Haskell can be found in [dMS95].

1.3.3 Limitations

Because terms with letrec represent graphs, we can see rewriting terms with letrec as graph rewriting. One of the advantages of this approach is that term rewriting can be directly applied to graphs. Thus, we have a notion of graph rewriting that is relatively easy to work with. However, in some cases we must define a *native* graph rewriting notation. That is, we must define a rewrite relation directly on graphs. For example, we may describe the behavior of the `set-car!` function in Scheme with the rewrite rule in Fig. 1.23.[1] Syntactically we can write this graph rewrite rule as a rewrite rule on lists of declarations:

$$u = \mathtt{set\text{-}car!}(v,z), v = \mathtt{cons}(x,y) \rightarrow u = v, v = \mathtt{cons}(z,y) \quad .$$

However, we cannot express this rule a term rewrite rule on terms with letrec. We will not pursue this notion of graph rewriting any further. Instead, we continue by discussing a serious technical problem: the fact that higher-order term graph rewriting is not confluent.

Because we want to study program optimization as a rewrite system, we will need rewrite systems that can express the inlining transformation. As we have seen in our example the inlining transformation for the lambda calculus consisted of a substitution followed by several β-steps. The conclusion is that in order to be able

[1] This example was taken from [LDLR99].

to inline every function everywhere we need (among others) the following three substitution rules:

$$\begin{array}{ll} \langle C[x] \mid x = M, D\rangle & \xrightarrow[\text{es}]{} \langle C[M] \mid x = M, D\rangle \\ \langle M \mid x = C[y], y = N, D\rangle & \xrightarrow[\text{is}]{} \langle M \mid x = C[N], y = N, D\rangle \\ \langle M \mid x = C[x], D\rangle & \xrightarrow[\text{cs}]{} \langle M \mid x = C[C[x]], D\rangle \end{array}$$

These rules are called external substitution, internal substitution and cyclic substitution.[2] Unfortunately, these three rules form a non-confluent rewrite system. For example, consider the term $\langle x \mid x = F(x)$. We can reduce this term as follows:

$$\begin{array}{ccc} \langle x \mid x = F(x)\rangle & \xrightarrow{\text{cs}} & \langle x \mid x = F(F(x))\rangle \\ \downarrow{\scriptstyle \text{es}} & & \\ \langle F(x) \mid x = F(x)\rangle & & \\ \downarrow{\scriptstyle \text{cs}} & & \\ \langle F(x) \mid x = F(F(x))\rangle & & \end{array} \tag{1.1}$$

By further contracting substitution redexes this diagram cannot be closed: every reduct of $\langle x \mid x = F(F(x))\rangle$ will have an even number of F symbols and every reduct of $\langle F(x) \mid x = F(F(x))\rangle$ will have an odd number of F symbols. If we extend the rewrite system with a set of rules that allows us to rewrite any term to a flat term with the same graph then we can close the diagram as follows:

$$\begin{array}{ccc} \langle x \mid x = F(x)\rangle & \xrightarrow{\text{cs}} & \langle x \mid x = F(F(x))\rangle \\ \downarrow{\scriptstyle \text{es}} & & \downarrow{\scriptstyle \text{flatten}} \\ \langle F(x) \mid x = F(x)\rangle & & \langle x \mid x = F(y), y = F(x)\rangle \\ \downarrow{\scriptstyle \text{cs}} & & \downarrow{\scriptstyle \text{es}} \\ \langle F(x) \mid x = F(F(x))\rangle & & \langle F(y) \mid x = F(y), y = F(x)\rangle \\ & \xrightarrow[\text{flatten}]{} & \downarrow{\scriptstyle \text{flatten}} \\ & & \langle u \mid u = F(v), v = F(w), w = F(v)\rangle \end{array}$$

Note that during the flattening we also apply α-conversion. It is an open question if this extension is confluent. For example, there is no known way to close the following diagram in this system:

$$\begin{array}{ccc} \langle x \mid x = F(x,x)\rangle & \longrightarrow & \langle x \mid x = F(F(x,x),x)\rangle \\ \downarrow & & \\ \langle x \mid x = F(x,F(x,x))\rangle & & \end{array}$$

[2] In [AK96, AK97] cyclic substitution was deemed to be a special case of internal substitution. In this thesis we assume that internal substitution involves two different equations, because it is easier to understand and because the distinction is needed in proofs.

However, it is known that if we add a rule known as "copying", the result is a confluent rewrite system. Anyway, we hope to have made it clear that there are potential confluence problems as soon as the three substitution rules are allowed.

Adding first-order orthogonal term rewriting doesn't pose very big problems. A fundamental result is that on terms with letrec we have that orthogonal first-order term rewriting modulo unwinding is confluent. However, if we also have lambda abstraction and β-reduction then the non-confluence problem has a much more fundamental nature.

The following example is adapted from an example in [AK97]. Consider the term

$$\langle f\,O \mid f = \lambda x.c_1\,x\,(g\,(S\,x)), g = \lambda y.c_2\,y\,(f\,(S\,y))\rangle \ .$$

Starting with this term we have the following two reductions:

$$\begin{aligned}
&\langle f\,O \mid f = \lambda x.c_1\,x\,(g\,(S\,x)), g = \lambda y.c_2\,y\,(f\,(S\,y))\rangle \\
&\quad\rightarrow \langle f\,O \mid f = \lambda x.c_1\,x\,((\lambda y.c_2\,y\,(f\,(S\,y)))\,(S\,x))\rangle \\
&\quad\rightarrow \langle f\,O \mid f = \lambda x.c_1\,x\,(c_2\,(S\,x)\,(f\,(S\,(S\,x))))\rangle
\end{aligned}$$

and

$$\begin{aligned}
&\langle f\,O \mid f = \lambda x.c_1\,x\,(g\,(S\,x)), g = \lambda y.c_2\,y\,(f\,(S\,y))\rangle \\
&\quad\rightarrow \langle (\lambda x.c_1\,x\,(g\,(S\,x)))\,O \mid f = \lambda x.c_1\,x\,(g\,(S\,x)), g = \lambda y.c_2\,y\,(f\,(S\,y))\rangle \\
&\quad\rightarrow \langle c_1\,O\,(S\,O) \mid f = \lambda x.c_1\,x\,(g\,(S\,x)), g = \lambda y.c_2\,y\,(f\,(S\,y))\rangle \\
&\quad\rightarrow \langle c_1\,O\,(S\,O) \mid g = \lambda y.c_2\,y\,((\lambda x.c_1\,x\,(g\,(S\,x)))\,(S\,y))\rangle \\
&\quad\rightarrow \langle c_1\,O\,(S\,O) \mid g = \lambda y.c_2\,y\,(c_1\,(S\,y)\,(g\,(S\,(S\,y))))\rangle \ .
\end{aligned}$$

Even if we use β-reduction modulo unwinding equivalence the terms in which these two reductions end have no common reduct.

The conclusion is that if we want to consider a higher-order graph rewrite system that contains all three substitution rules then we will have to deal with a non-confluent rewrite system. From the program transformation, we gave as an example, we can conclude that any rewrite system that is capable of expressing many such transformations will contain the three substitution rules and will therefore be non-confluent. Thus, when we consider the semantics of graph rewriting we will have to deal with rewrite systems that are non-confluent and non-terminating.

1.4 The semantics of graph rewriting

Given a confluent and terminating term rewrite system, we can easily define the semantics of a term to be the unique normal form of the term with respect to the rewrite system. Unfortunately, we want to consider graph rewrite systems that are neither confluent nor terminating. The problem posed by the non-termination can be solved by considering constructions similar to that of the Böhm tree in the lambda calculus. This solves one problem, but the classical way of developing a

Böhm tree theory depends on confluence. Therefore, we will need to introduce a new property that is strong enough to do the Böhm tree theory and weaker than confluence: *skew confluence.*

1.4.1 The Böhm tree

The lambda calculus is an example of a rewrite system that is non-terminating. Because the lambda calculus is confluent, we have that a term reduces to at most one normal form. We can still use normal forms as semantics by defining the semantics of a lambda term as the unique normal form the term reduces to if it exists and undefined otherwise. However, this semantics is much too coarse. A more refined approach is that of the Böhm tree. We will present this approach now in the form in which Lévy presented it in his thesis [Lév78].

The key observation behind the definition of the Böhm tree and similar notions is that if we reduce a term then some parts of that term can become stable. That is, some parts of the term contain no redexes and the term cannot be reduced in a way such that the stable parts will again contain redexes. The stable part of a lambda term M is the normal form of M with respect to the following rewrite rules:

$$\begin{array}{lll} (\lambda x.M)\,N & \xrightarrow[\omega_{\mathrm{BT}}]{} & \Omega \ ; \\ \lambda x.\Omega & \xrightarrow[\omega_{\mathrm{BT}}]{} & \Omega \ ; \\ \Omega M & \xrightarrow[\omega_{\mathrm{BT}}]{} & \Omega \ . \end{array}$$

This normal form is denoted with $\omega_{\mathrm{BT}}(M)$. The stable part of a term is also referred to as the *direct approximation* of the term or the *information content* of the term. During a reduction the information content increases. That is, we have that

$$M \xrightarrow[\beta]{} N \implies \omega_{\mathrm{BT}}(M) \leq_{\Omega} \omega_{\mathrm{BT}}(N) \ .$$

In other words, we have that the information content is monotonic with respect to reduction.

The Böhm tree of infinite normal form of a term is the set of all information that can be found in reducts of that term. We formalize the Böhm tree $\mathrm{BT}(M)$ of a lambda term M as follows:

$$\mathrm{BT}(M) = \{a \in \omega_{\mathrm{BT}}(\Lambda) \mid M \twoheadrightarrow N, a \leq_{\Omega} \omega_{\mathrm{BT}}(N)\} \ .$$

In effect the Böhm tree of M is the downward closure of the set

$$\{\omega_{\mathrm{BT}}(N) \mid M \twoheadrightarrow N\} \ .$$

The use of a downward closure is necessary to ensure that every two convertible lambda terms have the same Böhm tree. We will refer to this important property as the uniqueness of the infinite normal form UN^{∞}.

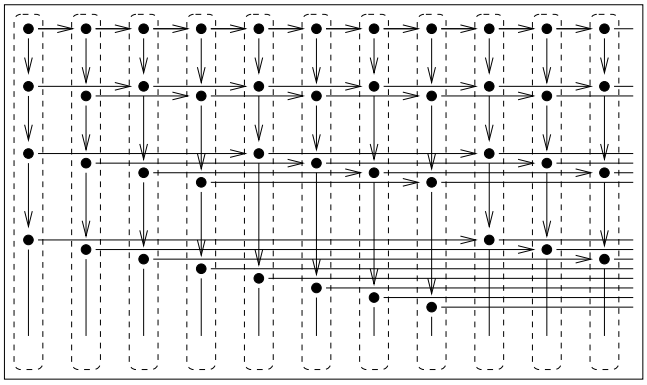


Figure 1.24: The $\xrightarrow[\mu,\mu\mu]{}$ reduction graph of $\mu x.F(x)$

1.4.2 Skew Confluence

The μ-calculus is a rewrite system with a single rule:

$$\mu x.M \xrightarrow[\mu]{} M[x := \mu x.M] \ .$$

We can define infinite normal forms for the μ-calculus by defining the information content $\omega_\mu(M)$ of a term M as the normal form of M with respect to the rewrite rule:

$$\mu x.M \xrightarrow[\omega_\mu]{} \Omega \ .$$

Because the μ-calculus is a confluent CRS, we can easily prove that the infinite normal forms are unique. However, if we add the rewrite rule

$$\mu x.C[x] \xrightarrow[\mu\mu]{} \mu x.C[C[x]]$$

and keep the notion of information content then we have a problem: the rewrite relation $\xrightarrow[\mu,\mu\mu]{}$ is not confluent. For example, we have the following reductions:

$$\begin{array}{ccccc} \mu x.F(x) & \longrightarrow & F(\mu x.F(x)) & \longrightarrow & F(\mu x.F(F(x))) \\ \downarrow & & & & \\ \mu x.F(F(x)) & & & & \end{array}$$

The two terms $\mu x.F(F(x))$ and $F(\mu x.F(F(x)))$ do not have a common reduct, because any reduct of $\mu x.F(F(x))$ will contain an even number of F symbols and every reduct of $F(\mu x.F(F(x)))$ will contain an odd number of F symbols. The reduction graph of $\mu x.F(x)$ is drawn in Fig. 1.24. The horizontal reductions are μ-steps and the vertical reductions are $\mu\mu$-steps. The dashed 'sticks' indicate the terms that have the same information content.

Although $\xrightarrow[\mu,\mu\mu]{}$ is not confluent, it does have unique infinite normal forms. This is because having unique infinite normal forms does not directly depend on

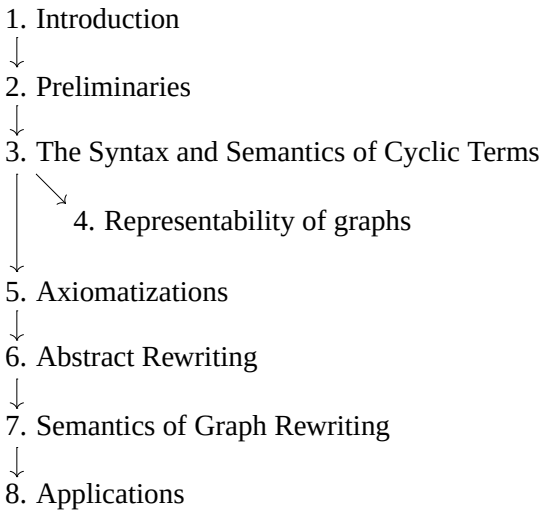


Figure 1.25: Overview

confluence, but on the ability to derive the same information from every reduct of a term. That is, if M reduces to both N_1 and N_2 then we need to be able to derive from N_2 all the information that we can derive from N_1. Deriving information from a term means reducing that term and taking the information content. Therefore, it is sufficient that if M reduces to both N_1 and N_2, we are able to derive the information content of N_1 from N_2. That is, it is sufficient if for some reduct N of N_2 the following diagram holds:

$$\begin{array}{ccc} M & \longrightarrow\!\!\!\!\!\rightarrow & N_2 \\ \downarrow & & \vdots \\ N_1 & \leq_{\omega_\mu} & N \end{array}$$

where $P \leq_{\omega_\mu} Q$ if $\omega_\mu(P) \leq_\Omega \omega_\mu(Q)$. This diagram is a special case of a property known as skew confluence. We say that $\xrightarrow[\alpha]{}$ is skew confluent with respect to $\xrightarrow[\beta]{}$ if the following diagram holds:

$$\begin{array}{ccc} & \overset{\alpha}{\longrightarrow\!\!\!\!\!\rightarrow} & \\ {\scriptstyle\alpha}\downarrow & & \vdots\,{\scriptstyle\alpha} \\ N_1 & \underset{\beta}{-\!\!\!\!\!\rightarrow} & N \end{array}$$

In the new terminology we can say that $\xrightarrow[\mu,\mu\mu']{}$ has unique infinite normal forms, because $\xrightarrow[\mu,\mu\mu']{}$ is skew confluent with respect to $\leq_{\omega_\mu}$.

1.5 Overview

Before we finish the introduction and continue with the technical part of this thesis let us give an overview of the thesis. (See Fig. 1.25.) With only a few exceptions, the following chapters build towards the goal of a generic theory of infinite normal forms for higher order graph rewrite systems. Rather than just the lambda calculus we use CRSs as higher-order rewrite systems[3].

After the preliminaries, we continue with the formal definition of the syntax and semantics of terms with letrec. This includes the introduction of higher order term graphs, which are a formal representation of the pictures we have drawn.

In the next chapter we study the expressive power of the let, μ and letrec. More precisely, we give a characterization of the graphs that can be represented by terms, which use these constructs. In this chapter we motivate the use of the letrec, by showing that the letrec is more powerful than the let and μ together. Technically speaking this chapter is an intermezzo as none of its definitions or results are used in the remainder of the thesis.

In Chap. 5 we continue building towards the main goal by axiomatizing several equivalence relations on cyclic terms. Among others, we axiomatize the terms that have the same graph and the terms that have the same unwinding. As the application to the theory of graph rewrite systems, we will give rewrite systems that can be used as the basis of graph rewrite systems.

We continue with a study of the principles of Böhm trees and infinitary rewriting in Chap. 6. The main technical vehicles in this study are complete partial orders and abstract rewriting systems on them. A crucial role is played by the finite elements of complete partial orders.

In Chap. 7 we concentrate on rewrite systems defined on terms with letrec. Most of these systems can be decomposed into two sets of rules. The first set of rules dealing with 'bookkeeping' and the second set dealing with the real work. For example, in an explicit substitution lambda calculus the rules dealing with substitution are bookkeeping rules and the β-rule is the rule that does the real work. Intuitively, a bookkeeping rule is a rule that doesn't change the unwinding. We apply the theory of the previous chapter to terms with letrec to provide a framework for defining Böhm semantics. This theory is extended with properties of contexts, which we could not study in the previous chapter.

In Chap. 8 we apply the theory of Chap. 7. More precisely, we give a few more examples of unwinding calculi, we define a generic notion of infinite normal form on certain combinatory reduction systems and we do a case study of cyclic lambda calculi.

[3]For the reader that doesn't know CRS's. CRS's are a form of higher order rewriting. It is best to think of a CRS as lambda calculus plus term rewriting plus complicated rules involving lambda abstractions.

Chapter 2

Preliminaries

2.1 Sets

On sets we will use the normal notation for union, intersection and cartesian product. For the disjoint union we use the symbol $\uplus$. The formal definition of disjoint union is

$$\uplus_{i\in I} S_i = \bigcup_{i\in I} (\{i\} \times S_i) \quad .$$

However, if we have disjoint sets A and B then we will make no difference between $A \cup B$ and $A \uplus B$. We define the binary operators set minus ($\setminus$) and set difference (Δ) by

$$A \setminus B = \{a \in A \mid a \notin B\}$$
$$A \Delta B = (A \setminus B) \cup (B \setminus A)$$

The following notation is probably known, but we want to avoid any possible confusion:

Set	Definition	Description
$\mathcal{P}(S)$	$\{A \mid A \subset S\}$	the power set of S;
A^n	$\Pi_{i\in n}$	
S^*	$\cup_{i\in\mathbb{N}} S^i$	the set of words over S.
$\mathbb{N}$	$\{0,1,2,\cdots\}$	The set of the natural numbers.
∞		Infinity.
$\overline{\mathbb{N}}$	$\mathbb{N} \cup \{\infty\}$	The natural numbers plus infinity.

2.2 Relations

Though it is clear what a partial order is, the formal definition is not unique. We use the following definitions:

Definition 2.2.1 A partial order is a pair $(S, \leq)$, where $\leq$ is a transitive, reflexive and anti-symmetric binary relation over S. A set $D \subset S$ is a directed set if every finite subset D' of D there exists $d \in D$ such that d is an upper bound of D'. A set

$D \subset S$ is downward closed if $\forall d, d' \in D : d \leq d' \in D \implies d \in D$. A set $I \subset S$ is an ideal if I is downward closed and directed. The set of ideals over S is denoted by $I(S)$. A partial order $(S, \leq)$ is complete if every directed subset has a least upper bound.

The notions in the previous definition were all very well-known. Somewhat less well-known are the following two:

Definition 2.2.2 Given a complete partial order $(A, \leq)$. An element $a \in A$ is *finite* if for every directed set $D \subset A$, such that $a \leq \mathrm{lub} D$, we have that there exists $d \in D$, such that $a \leq d$. The set of finite elements in A is denoted by $\mathcal{F}(A)$. A CPO is algebraic if $\forall a \in A : a = \mathrm{lub}\{a' \in \mathcal{F}(A) \mid a' \leq a\}$.

If $\mathcal{A} \equiv (A, \leq)$ is a partial order then $\mathcal{A}_I \equiv (I(A), \subset)$ is a complete partial order, called the ideal completion of $\mathcal{A}$. If moreover all elements of $\mathcal{A}$ are finite then $\mathcal{F}(I(A)) = \{\downarrow \{a\} \mid a \in A\}$ and $\mathcal{A}_I$ is algebraic.

The downward closure $\downarrow S$ of a set S is a well-known notion. In addition, we will also define the finite element downward closure $\downarrow_{\mathcal{F}} S$ of S. This definition uses the set of finite approximation $\downarrow_{\mathcal{F}} (s)$ of an element s;

Definition 2.2.3 Given a complete partial order $(A, \leq)$, we define

$$\begin{array}{ll} \downarrow_{\mathcal{F}} (s) = \{a \in \mathcal{F}(A) \mid a \leq s\} & , \forall s \in A \\ \downarrow_{\mathcal{F}} (S) = \cup_{s \in S} \downarrow_{\mathcal{F}} (s) & , \forall S \subset A \end{array}$$

Given a partial order $(A, \leq)$, an element $a \in A$ and a subset $S \subset A$.

- We say that a is a lower bound of S if $\forall s \in S : \ a \leq s$.
- We say that a is the greatest lower bound of S, denoted $\mathrm{glb} S$, if a is a lower-bound of S and every lowerbound a' of S satisfies the condition that $a' \leq a$.
- We say that a is an upperbound of S if $\forall s \in S : \ s \leq a$.
- We say that a is the least upperbound of S, denoted $\mathrm{lub} S$, if a is an upper bound of S and every upper bound a' of S satisfies the condition that $a \leq a'$.

Based on the notions of least upper bound and greatest lower bound we define the following three partial functions form sequences of limit ordinal length over A to A:

$$\begin{array}{ll} \mathrm{liminf}(a_i)_{i \in \alpha} & = \mathrm{lub}\{a \in A \mid \exists \beta \in \alpha : \forall i \in \alpha \setminus \beta : a \leq a_i\} \ ; \\ \mathrm{limsup}(a_i)_{i \in \alpha} & = \mathrm{glb}\{a \in A \mid \exists \beta \in \alpha : \forall i \in \alpha \setminus \beta : a_i \leq a\} \ ; \\ \mathrm{lim}(a_i)_{i \in \alpha} & = \mathrm{liminf}(a_i)_{i \in \alpha}, \text{ if } \mathrm{liminf}(a_i)_{i \in \alpha} = \mathrm{limsup}(a_i)_{i \in \alpha} \ . \end{array}$$

For the special case of sets we have:

$$\begin{array}{lll} \mathrm{liminf}_{\gamma \in \alpha} A_\gamma & = \cup_{\gamma \in \alpha} \cap_{\delta \in \alpha \setminus \gamma} A_\delta & \\ \mathrm{limsup}_{\gamma \in \alpha} A_\gamma & = \cap_{\gamma \in \alpha} \cup_{\delta \in \alpha \setminus \gamma} A_\delta & \\ \mathrm{lim}_{\gamma \in \alpha} A_\gamma & = \mathrm{liminf}_{\gamma \in \alpha} A_\gamma & \text{if } \mathrm{liminf}_{\gamma \in \alpha} A_\gamma = \mathrm{limsup}_{\gamma \in \alpha} A_\gamma \end{array}$$

and

$$\begin{aligned}\liminf_{\gamma\in\alpha} A_\gamma &= \{x \mid \exists\gamma : \forall\delta, \gamma < \delta < \alpha : x \in A_\delta\}\\ \limsup_{\gamma\in\alpha} A_\gamma &= \{x \mid \forall\gamma : \exists\delta, \gamma < \delta < \alpha : x \in A_\delta\}\end{aligned}$$

Proposition 2.2.4 Given an algebraic complete partial order $(A, \leq)$ and a subset S of A, we have that if the least upper bound of S exists then the least upper bound of $\downarrow_{\mathcal{F}} S$ exists and

$$\operatorname{lub} S = \operatorname{lub} \downarrow_{\mathcal{F}} S \ .$$

Proof. We have that $\operatorname{lub} S$ is an upper bound of $\downarrow_{\mathcal{F}} S$, because for every element $a \in \downarrow_{\mathcal{F}} S$, we have that $a \leq a'$ for some element $a' \in S$ and by definition $a' \leq \operatorname{lub} S$. By contradiction we can prove that $\operatorname{lub} S$ is the least upper bound of $\downarrow_{\mathcal{F}} S$: assume that b is an upper bound of $\downarrow_{\mathcal{F}} S$. We claim that b is an upper bound of S. By this claim we have that $\operatorname{lub} S \leq b$. To show the claim let a' be an arbitrary element of S. We must now show that $a' \leq b$. Because the partial order is algebraic we have that $a' = \operatorname{lub} \downarrow_{\mathcal{F}} a'$. Because $\downarrow_{\mathcal{F}} a' \subset \downarrow_{\mathcal{F}} S$ and b is an upper bound of $\downarrow_{\mathcal{F}} S$, we have that $\operatorname{lub} \downarrow_{\mathcal{F}} a' \leq b$. □

2.3 Abstract Reduction Systems

Introductions to term rewriting and abstract reduction systems can be found in [Klo92, DJ90]. Here, we will only mention some non-standard notation we use and introduce reduction and confluence modulo, for which there does not seem to be a standard notation and terminology.

The usual symbol for the conversion relation is $=$. Because this symbol $=$ is also used for the declaration of let and letrec, we use a different symbol. Because in diagrams arrows look best, we use a double sided arrow. By putting an order above the arrow we indicate that every step in the conversion is also an ascending sequence:

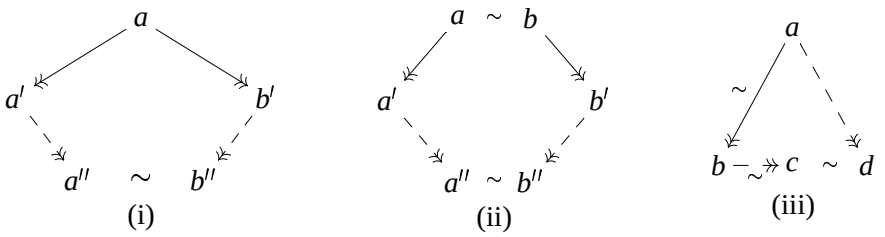
Definition 2.3.1 Given an ARS $(A, \to)$ and an order $(A, \leq)$, we define

$$\begin{aligned}\leftrightarrow &= (\to \cup \leftarrow)^*\\ \overset{\leq}{\leftrightarrow} &= ((\to \cup \leftarrow) \cap \leq)^*\end{aligned}$$

In addition we will need to indicate in diagrams that we are rewriting an object or a term to normal form. We will indicate this with a special arrow:

Definition 2.3.2 Given an ARS $(A, \to)$, we write $a \twoheadrightarrow b$ if $a \to\!\!\!\!\!\to b$ and b is a normal form.

Different versions of confluence modulo and reduction modulo have been proposed [Hue80, DJ90]. Our definitions are given in pictorial form in Fig. 2.1 and in algebraic form below:



(i) $\to$ is weakly confluent modulo $\sim$
(ii) $\to$ is confluent modulo $\sim$
(iii) $\to$ is complete for $\underset{\sim}{\longrightarrow}$ modulo $\sim$

Figure 2.1: Pictorial definitions

Definition 2.3.3 Given an ARS $(A,\to)$ and an equivalence relation $\sim$ on A. Then
- reduction modulo (written as $\underset{\sim}{\longrightarrow}$) is defined by:

$$a \underset{\sim}{\longrightarrow} b, \text{ if } \exists a',b' \in A:\ a \sim a' \to b' \sim b \ .$$

- $\to$ is weakly confluent modulo $\sim$ if

$$\forall a,a',b' \in A:\ (a \to\!\!\!\!\to a', a \to\!\!\!\!\to b' \implies \exists a'',b'':\ a' \to\!\!\!\!\to a'', b' \to\!\!\!\!\to b'', a'' \sim b'') \ .$$

- $\to$ is confluent modulo $\sim$ if

$$\forall a,a',b,b' \in A:\ (a \sim b, a \to\!\!\!\!\to a', b \to\!\!\!\!\to b' \implies \exists a'',b'' : a' \to\!\!\!\!\to a'', b' \to\!\!\!\!\to b'', a'' \sim b'') \ .$$

- $\to$ is complete for $\underset{\sim}{\longrightarrow}$ modulo $\sim$ if

$$\forall a,b \in A:\ (a \underset{\sim}{\longrightarrow}\!\!\!\!\to b \implies \exists c,d \in A:\ b \underset{\sim}{\longrightarrow}\!\!\!\!\to c, c \sim d, a \to\!\!\!\!\to d) \ .$$

2.4 Terms

Definition 2.4.1 A signature Σ is a triple $(\mathsf{Fun},\mathsf{arity},\mathsf{Var})$, where Fun and Var are disjoint sets of function symbols and variables respectively and where $\mathsf{arity}:\mathsf{Fun}\to\mathbb{N}$ is a function giving the arity of its argument. For every $n\in\mathbb{N}$ we define the set of n-ary function symbols as $\mathsf{Fun}_n = \{f \in \mathsf{Fun} \mid \mathsf{arity}(f) = n\}$

Definition 2.4.2 The set of terms over Σ is given by

$$M ::= x \mid F_n(M_1,\cdots,M_n) \ ,$$

where $x\in\mathsf{Var}$ and $F_n \in \mathsf{Fun}_n$.

If $@\in\mathsf{Fun}_2$ then the set of lambda terms over Σ is given by:

$$M ::= x \mid F_n(M_1,\cdots,M_n) \mid \lambda x.M \ ,$$

where $x\in\mathsf{Var}$ and $F_n \in \mathsf{Fun}_n$.

Typical elements of Var are x, y, z, u, v, w, typical elements of Fun are $@, F, G, H$. The requirements that $x \in \mathsf{Var}$ and $F_n \in \mathsf{Fun}_n$ in the definition of terms will be left implicit in the future. That is, the set of terms will be specified as:

$$M ::= x \mid F_n(M_1, \cdots, M_n) \ .$$

As usual we will write MN for $@(M,N)$. Also as usual the only correct way to parse $\lambda x.xyz$ is as $\lambda x.@(@(x,y),z)$.

Definition 2.4.3 The set of positions in a (lambda) term M, denoted $\mathsf{Pos}(M)$, is a set of words over the natural numbers recursively defined by

$$\begin{array}{lcl} \mathsf{Pos}(x) & = & \{\varepsilon\} \\ \mathsf{Pos}(F(M_1, \cdots, M_n)) & = & \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{i\,p \mid p \in \mathsf{Pos}(M_i)\} \\ \mathsf{Pos}(\lambda x.M) & = & \{\varepsilon\} \cup \{1\,p \mid p \in \mathsf{Pos}(M)\} \end{array}$$

For $p \in \mathsf{Pos}(M)$ we define the subterm of M at position p, $M|_p$, as

$$\begin{array}{lcl} M|_\varepsilon & = & M \\ F(M_1, \cdots, M_n)|_{i\,p} & = & M_i \\ \lambda x.M|_{1\,p} & = & M \end{array}$$

For $p \in \mathsf{Pos}(M)$ we define the context of M at position p, $M[]_p$, as

$$\begin{array}{lcl} M[]_\varepsilon & = & \Box \\ F(M_1, \cdots, M_n)[]_{i\,p} & = & F(M_1, \cdots, M_{i-1}, M_i[]_p, M_{i+1}, \cdots, M_n) \\ (\lambda x.M)[]_{i\,p} & = & \lambda x.(M[]_p) \end{array}$$

Definition 2.4.4 The set of terms Λ of the lambda calculus with positions $(\Lambda, \xrightarrow[p]{})_{p \in \mathbb{N}^*}$ is given by:

$$M ::= x \mid \lambda x.M \mid MM$$

The reduction relation $\xrightarrow[p]{}$ is given by:

$$\frac{}{(\lambda x.M)N \xrightarrow[\varepsilon]{} M[x := N]} \qquad \frac{M \xrightarrow[p]{} N}{\lambda x.M \xrightarrow[1p]{} \lambda x.N}$$

$$\frac{M \xrightarrow[p]{} N}{MP \xrightarrow[1p]{} NP} \qquad \frac{M \xrightarrow[p]{} N}{PM \xrightarrow[2p]{} PN}$$

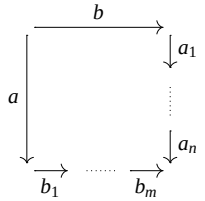
2.5 Diagrams

A powerful technique, decreasing diagrams, to prove commutativity from local commutativity was developed by van Oostrom (see [vO94]). This technique consists of associating a label to each reduction step and giving a well-founded order on these labels. If all local diagrams turn out to be of a specific kind, namely *decreasing*, then commutativity is guaranteed.

Definition 2.5.1 Let $|.|$ be the measure from strings of labels to multisets of labels defined by:

$$|a_1 \dots a_n| = \{\{a_i | \text{ there is no } j < i \text{ with } a_j > a_i\}\} \quad .$$

Then, the diagram



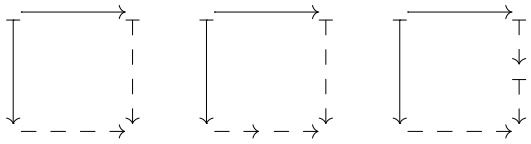
is *decreasing* if $\{\{a,b\}\} \geq |ab_1 \dots b_m|$ and $\{\{a,b\}\} \geq |ba_1 \dots a_n|$.

Theorem 2.5.2 If two labeled reduction systems are locally commutative and all local diagrams are decreasing with respect to a well founded order on labels then the systems are commutative.

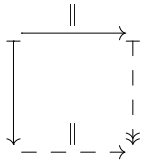
Proof. See [vO94]. □

We remark that we are free to give any label to any occurrence of a step. It is especially useful to be able to give different labels to the same step occurring horizontally and vertically. However, once we give a label to an occurrence of a step then it is fixed.

The decreasing diagrams technique sometimes fails if there is duplication in both the horizontal and vertical direction, *e.g.,* there is no possible labeling that makes the following diagrams all decreasing:



It is often possible to solve this problem by introducing a form of parallel reduction in, for example, the horizontal direction. With respect to parallel reduction the three diagrams should then collapse into the single diagram



which can be made decreasing by ordering the parallel reduction larger than the standard reduction.

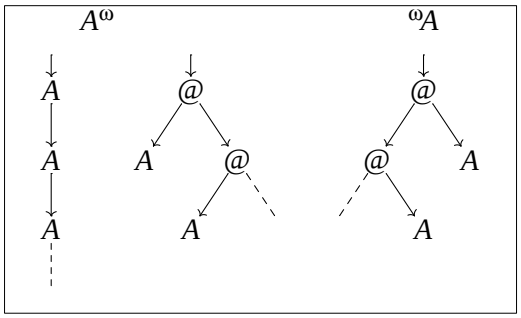


Figure 2.2: Notation for infinite terms

2.6 Infinite Terms

An actual direct definition of a set of infinite terms is:

Definition 2.6.1 A set $T \subset \mathbb{N}^* \times (\mathsf{Fun} \oplus \mathsf{Var} \oplus (\lambda\mathsf{Var}))$ is a term if

- T is a partial function on *Pos*.
- If $T(p)$ is defined then for all $p' \sqsubset p$ we have that $T(p')$ is defined.
- If $T(p) \in \mathsf{Var}$ and $T(pq)$ is defined then $q = \varepsilon$.
- If $T(p) \in \lambda\mathsf{Var}$ and $T(pq)$ is defined then $q = \varepsilon$ or $q = 1q'$.
- If $T(p) \in \mathsf{Fun}$ and $T(pq)$ is defined then $q = \varepsilon$ or $q = iq'$ for $1 \leq i \leq arity(T(p))$.

The set of terms is a CPO, if it is ordered by the subset relation. Also an ultra-metric can be defined on terms as follows:

$$D(T_1, T_2) = \max\{2^{-|p|} \mid T_1(p) \neq T_2(p)\} \ ,$$

where $\max \emptyset = 0$.

For infinite terms we have some notation, which is illustrated in Fig. 2.2. For a unary function symbol A we have:

$$A^\omega = A(A^\omega) = A(A(A(\cdots))) \ .$$

Given any term A and the application with its usual infix notation we have:

$$A^\omega = AA^\omega = A\,(A\,(A\,(\cdots))) \text{ and } {}^\omega A = {}^\omega AA = (((\cdots A)\,A)\,A) \ .$$

2.7 Combinatory Reduction Systems

Combinatory Reductions Systems are a form of higher-order rewriting systems, defined by Klop ([Klo80]). A more recent introduction to combinatory reduction systems can be found in ([KvOvR93]). Because CRSs are not as well-known as term rewriting systems and the lambda calculus, we will spend some time introducing them.

In order to introduce combinatory reduction systems we need a new class of syntactic objects: *metavariables.* Like function symbols meta variables have a fixed arity. The usual notation for a metavariable of arity k is Z_n^k. Also if a metavariable is used as $Z(M_1,\cdots,M_k)$ then it is assumed that Z has arity k. We also use a special binary constructor $[.].$, called the abstraction operator .

Definition 2.7.1 The set of metaterms (MT) is defined by

$$MT ::= x \mid [x]MT \mid F(MT_1,\cdots,MT_n) \mid Z(MT_1,\cdots,MT_n) \ .$$

A meta-term is a term if no metavariable occurs in the term.

For (meta)terms we have the following notation: $[x_1]\cdots[x_n]M$ is denoted $[x_1,\cdots,x_n]$ and $F([x_1,\cdots,x_n]M)$ is denoted $F\,x_1\cdots x_n.M$. If we have both application and abstraction in a (meta)term then application binds stronger. That is, $[x]xx$ is parsed as $[x](xx)$. All binding conventions of the lambda calculus are applicable if we read $\lambda x.M$ for $[x]M$ and if we count metavariables as function symbols.

Definition 2.7.2 Given a signature $\Sigma \equiv (\mathsf{Fun}, \mathsf{arity}, \mathsf{Var})$ and a set of unary function symbols $\mathcal{B} \subset \mathsf{Fun}_1$, the set of CRS terms with binders $\mathcal{B}$ is the subset of the set of term specified by

$$M ::= x \mid Bx.M \mid F_n(M_1,\cdots,M_n) \ ,$$

where $B \in \mathcal{B}$.

Definition 2.7.3 A rewrite rule is a CRS is a pair (M,N), written as $M \to N$, where M and N are metaterms such that

- M and N are closed metaterms;
- M is of the form $F(M_1,\cdots M_N)$;
- Any metavariable occurring in N occurs in M;
- The metavariables in M occur only as subterms of the form $Z(x_1,\cdots,x_n)$.

If no metavariable occurs twice in M then the rule is *left-linear.*

Definition 2.7.4 Given a rule $M \to N$ and a substitution from metavariables to terms σ, such that for every metavariable Z^k we have that $\sigma(Z^k) \equiv [x_1, \cdots, x_k]M$. We define single step reduction on terms as:

$$C[M^\sigma] \to C[N^\sigma] \ ,$$

where substitution is extended from terms to metaterms by the equation

$$Z(M_1, \cdots, M_k)^\sigma = M[x_1 := M_1^\sigma, \cdots, x_k := M_k^\sigma], \text{ if } \sigma(Z) = [x_1, \cdots, x_k]M \ .$$

2.8 Context restricted rewriting

Usually in term rewriting a subterm is a redex no matter where in the term this subterm occurs. In this thesis many rewrite systems will be defined where a redex is a subterm occurring at the right place. That is, depending on the context in which it occurs a sub-term is a redex or not. Hence the name *context restricted rewriting.*

Definition 2.8.1 A *context restricted rewrite system* $\mathcal{C}$ over a signature Σ consists a set of CRS rules $\mathcal{R}_\mathcal{C}$ and a map $\phi_\mathcal{C} : \mathcal{R}_\mathcal{C} \to C(\Sigma)$. The reduction relation $\underset{\mathcal{C}}{\to}$ is defined by

$$C[M] \underset{\mathcal{C}}{\to} C[N] \ ,$$

where $M \to N$ is an instance of a rule $L \to R$ in $\mathcal{R}_\mathcal{C}$ and where $C \in \phi_\mathcal{C}(L \to R)$.

Usually we will indicate the values of the function $\phi_\mathcal{C}$ by writing this value as a label of the rule. For example, if we write

$$L \underset{S}{\to} R$$

then we intend that $L \to R$ is a rule in $\mathcal{R}_\mathcal{C}$ and $\phi_\mathcal{C}(L \to R) = S$. The default label for a rewrite rule in a context restricted rewrite system is the set of all contexts. That is, if the label is missing then a rewrite rule can be applied in any context.

In order to specify a context restricted rewrite system we must specify contexts. For the specification of contexts we will use two tools: EBNF-like specifications and regular expressions over contexts. The first method is strictly more powerful than the second, but the second method allows a more compact notation. This is useful because EBNF specification do not fit as a subscript and regular expressions do. For example, if we have unary function symbols A and B then we can specify the set C_a of contexts where the hole occurs at the root of the term or directly below an A as follows:

$$C_a ::= \Box \mid C[A(\Box)] \ .$$

The same definition can be obtained with a regular expression as follows:

$$C_a = \Box \mid CA(\Box) \ .$$

The set

$$C_{ab} = \{\Box, A(B(\Box)), A(B(A(B(\Box)))), \cdots\}$$

is specified by both

$$C_{ab} ::= \Box \mid A(B(C_{ab}))$$

and

$$C_{ab} = A(B(\Box))^* \ .$$

When used as a set of context C stands for the set of all contexts.

Chapter 3

The Syntax and Semantics of Cyclic Terms

This chapter contains precise formal definitions of several notions, which have been discussed in the introduction. These notions include the syntax of terms with letrec and the translation of those terms to graphs. The reader who is not interested in extensive formal details may wish to skip this chapter and use the index to look up definitions if necessary.

3.1 Syntax

In this section we will define the syntax of several sets of terms and some of the terminology related to those terms. To simplify these definitions we will define them as the restriction of a larger set of *pre-terms*. A pre-term can be a variable, a function application, an abstraction, a let, a μ or a letrec. Formally:

Definition 3.1.1 Given a signature $\Sigma \equiv (\mathsf{Fun}, \mathsf{arity}, \mathsf{Var})$. The set of *pre-terms* over Σ is given by

$$\begin{array}{rl} M ::= & x \mid F_n(M_1, \cdots, M_n) \mid [x]M \mid \mathsf{let}\ x = M_1\ \mathsf{in}\ M_2 \\ \mid & \mu x.M \mid \langle M_0 \mid x_1 = M_1, \cdots, x_n = M_n \rangle \ , \end{array}$$

where $x, x_1, \cdots, x_n \in \mathsf{Var}$, $F_n \in \mathsf{Fun}_n$ and $\{x_1, \cdots, x_n\}$ is a set of n distinct variables.

In Table 3.1 we have defined a number of interesting subsets of the set of pre-terms. Like for CRS terms we have also identified a subset of terms with a set of binders. (See 2.7.2). Thus, if we consider terms over a signature with $\mathsf{Fun}_1 = \{\mu, \lambda\}$ and $\mathsf{Fun}_2 = \{@\}$ then

$$\langle @(\lambda(x), \mu(x)) \mid x = [y]@(y,y) \rangle$$

is a higher order term with letrec. The picture of this term is drawn in Fig.3.1. This somewhat bizarre term is not present in the set of higher order terms with letrec with binders $\{\lambda, \mu\}$.

Table 3.1 subsets of the set of pre-terms

First order terms ...	
	$M ::= x \mid F_n(M_1,\cdots,M_n)$
with let	$M ::= x \mid F_n(M_1,\cdots,M_n) \mid \mathsf{let}\ x = M_0\ \mathsf{in}\ M_1$
with μ	$M ::= x \mid F_n(M_1,\cdots,M_n) \mid \mu x.M$
with let, μ	$M ::= x \mid F_n(M_1,\cdots,M_n) \mid \mathsf{let}\ x = M_0\ \mathsf{in}\ M_1 \mid \mu x.M$
with letrec	$M ::= x \mid F_n(M_1,\cdots,M_n) \mid \langle M_0 \mid x_1 = M_1,\cdots,x_n = M_n\rangle$
Higher order terms ...	
	$M ::= x \mid F_n(M_1,\cdots,M_n) \mid [x]M$
with let	$M ::= x \mid F_n(M_1,\cdots,M_n) \mid [x]M \mid \mathsf{let}\ x = M_0\ \mathsf{in}\ M_1$
with μ	$M ::= x \mid F_n(M_1,\cdots,M_n) \mid [x]M \mid \mu x.M$
with let, μ	$M ::= x \mid F_n(M_1,\cdots,M_n) \mid [x]M \mid \mathsf{let}\ x = M_0\ \mathsf{in}\ M_1 \mid \mu x.M$
with letrec	$M ::= x \mid F_n(M_1,\cdots,M_n) \mid [x]M \mid \langle M_0 \mid x_1 = M_1,\cdots,x_n = M_n\rangle$
Higher order terms with binders $\mathcal{B}$ ...	
	$M ::= x \mid F_n(M_1,\cdots,M_n) \mid Bx.M$
and let	$M ::= x \mid F_n(M_1,\cdots,M_n) \mid Bx.M \mid \mathsf{let}\ x = M_0\ \mathsf{in}\ M_1$
and μ	$M ::= x \mid F_n(M_1,\cdots,M_n) \mid Bx.M \mid \mu x.M$
and let, μ	$M ::= x \mid F_n(M_1,\cdots,M_n) \mid Bx.M \mid \mathsf{let}\ x = M_0\ \mathsf{in}\ M_1 \mid \mu x.M$
and letrec	$M ::= x \mid F_n(M_1,\cdots,M_n) \mid Bx.M \mid \langle M_0 \mid x_1 = M_1,\cdots,x_n = M_n\rangle$

We refer to $x_1 = M_1,\cdots,x_n = M_n$ as a list of declarations. If D_1 and D_2 are the lists of declarations $x_1 = M_1,\cdots,x_m = M_m$ and $y_1 = N_1,\cdots,y_n = M_n$, respectively, such that $\forall i,j : x_i \not\equiv y_j$ then we denote the list of declarations $x_1 = M_1,\cdots,x_m = M_m, y_1 = N_1,\cdots,y_n = M_n$ by D_1,D_2. When it is convenient to do so we sometimes denote a list of declarations as a set. For example, if $D_1 = \{x = y\}, D_2 = \{y = x\}$ then D_1,D_2 is a list of declarations, but D_1,D_1 is not. Usually, we will take the set of terms modulo permutation of declarations. That is, we will consider cyclic terms modulo the equation:

$$\langle M \mid D_1, x = N, y = P, D_2\rangle = \langle M \mid D_1, y = P, x = N, D_2\rangle \ .$$

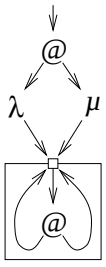


Figure 3.1: Example of a higher-order graph

We will often refer to a term with letrec as a *cyclic term.* A cyclic term is *flat* if it is of the form

$$\begin{array}{lll} M & ::= & \langle x \mid EQ_1, \cdots, EQ_n \rangle \\ EQ & ::= & x = F(x_1, \cdots, x_n) \mid x = [x]M \end{array}$$

For flat terms is is assumed that the black hole $\bullet$ is a constant in the signature. For arbitrary terms we may choose to include it as a constant or define it as $\bullet = \langle x \mid x = x \rangle$.

Definition 3.1.2 The *recursion variables* of a term M, denoted by $\mathsf{RecVar}(\mathsf{M})$, are given by:

$$\begin{array}{rcl} \mathsf{RecVar}(\mathsf{x}) & = & \emptyset \\ \mathsf{RecVar}(\mathsf{F}(\mathsf{M}_1, \cdots, \mathsf{M}_n)) & = & \cup_{i=1}^{n} \mathsf{RecVar}(\mathsf{M_i}) \\ \mathsf{RecVar}([\mathsf{x}]\mathsf{M}) & = & \mathsf{RecVar}(\mathsf{M}) \\ \mathsf{RecVar}(\langle \mathsf{M_0} \mid \mathsf{x}_1 = \mathsf{M}_1, \cdots, \mathsf{x}_n = \mathsf{M}_n \rangle) & = & \{x_1, \cdots, x_n\} \cup (\cup_{i=0}^{n} \mathsf{RecVar}(\mathsf{M_i})) \end{array}$$

Next, we define contexts with any finite non-zero number of holes. Because we will also take contexts up to permutation of declarations, we will have to number the holes in order to be able to fill the right hole with the right term.

Definition 3.1.3 (context) Given a signature Σ, such that $\Box_i \notin \Sigma$ $(i \in \mathbb{N})$. Let Σ' be obtained from Σ by adding $\Box_i$ $(i \in \mathbb{N})$ as constants. For $n \in \mathbb{N}$ an n-holed context over Σ is a term over Σ' with exactly one occurrence of each constant $\Box_1, \cdots, \Box_n$ and no occurrences of $\Box_{n+1}, \cdots$. Given an n-holed context C and cyclic terms $M_1, \cdots, M_n$, we define the term $C[M_1, \cdots, M_n]$ as the term obtained by syntactically replacing each symbol $\Box_i$ by M_i in C.

Even though formally we have numbered holes we can always omit the subscripts when we write down a context because in writing we do have a natural left to right order. Thus, by $F(\Box, \Box, \Box)$ we mean $F(\Box_1, \Box_2, \Box_3)$. Note that

$$C_1 = \langle F(x,y) \mid x = \Box, y = \Box \rangle \text{ and } C_2 = \langle F(x,y) \mid y = \Box, x = \Box \rangle$$

do *not* denote the same context. This is easily seen by explicitly adding the numbers of the holes.

$$C_1 = \langle F(x,y) \mid x = \Box_1, y = \Box_2 \rangle \text{ and } C_2 = \langle F(x,y) \mid y = \Box_1, x = \Box_2 \rangle$$

On cyclic terms we have two types of alpha conversion: alpha conversion of the abstraction operator and alpha conversion of the letrec itself. The definition of alpha conversion uses *syntactic replacement of variables*, which is a sort of substitution defined as:

Definition 3.1.4 Let τ be a function from variables to terms. For cyclic terms M we recursively define the *syntactic replacement* M^τ as follows:

$$\begin{array}{lcl} x^\sigma & = & \sigma(x) \\ F(M_1,\cdots,M_n)^\sigma & = & F(M_1^\sigma,\cdots,M_n^\sigma) \\ ([x]M)^\sigma & = & [x^\sigma]M^\sigma \\ \langle M_0 \mid x_1 = M_1,\cdots,x_n = M_n\rangle^\sigma & = & \langle M_0^\sigma \mid x_1^\sigma = M_1^\sigma,\cdots,x_n^\sigma = M_n^\sigma\rangle \end{array}$$

We can then define alpha conversion as the conversion relation generated by a rewrite relation:

Definition 3.1.5 (alpha conversion) Given cyclic terms $M_0,\cdots,M_n$ and n distinct variables $z_1,\cdots,z_n$, we define

$$\begin{array}{lcl} [x_1]M_1 & \xrightarrow[\alpha]{} & ([x_1]M_1)^\tau \\ \langle M_0 \mid x_1 = M_1,\cdots,x_n = M_n\rangle & \xrightarrow[\alpha]{} & \langle M_0 \mid x_1 = M_1,\cdots,x_n = M_n\rangle^\tau \ , \end{array}$$

where

$$\tau(y) = \begin{cases} z_i, \text{ if } y = x_i \\ y \text{ , otherwise} \end{cases}$$

By $\xleftrightarrow[\alpha]{}$ we denote the transitive, reflexive and compatible closure of $\xrightarrow[\alpha]{}$.

Substitution is usually defined as a function from alpha equivalence classes to alpha equivalence classes. This means that one can also define substitution as a *relation* between terms. Given a substitution σ we will define $M\sigma N$. This should be read as N can be obtained by applying the substitution σ to N.

Definition 3.1.6 (substitution) Let σ be a partial function from variables to terms. We extend σ to a relation on cyclic terms as follows:

$$\begin{array}{rll} x\ \sigma\ \sigma(x) & \text{, if } x \in \mathrm{Dom}_\sigma & \\ x\ \sigma\ x & \text{, if } x \notin \mathrm{Dom}_\sigma & \\ F(N_1,\cdots,N_k)\ \sigma\ F(P_1,\cdots,P_k) & \text{, if } N_i\sigma P_i & (i = 1\cdots k) \\ M\ \sigma\ N & \text{, if } M \xleftrightarrow[\alpha]{} M'\sigma N' \xleftrightarrow[\alpha]{} N & \\ [x]M\ \sigma\ [x]N & \text{, if } x \notin \mathrm{Dom}_\sigma \text{ and } M\sigma N & \\ \langle N_0 \mid y_i = N_i\rangle_{i=1}^k\ \sigma\ \langle P_0 \mid y_i = P_i\rangle_{i=1}^k & \text{, if } y_i \notin \mathrm{Dom}_\sigma,\ N_i\sigma P_i & (i = 0\cdots k) \end{array}$$

The partial function $[x_1 := M_1,\cdots,x_n := M_n]$ is given by:

$$x \mapsto \begin{cases} M_i, \text{ if } x = x_i \\ \bot \text{ , otherwise} \end{cases}$$

In order to be able to see a substitution as a function on α-equivalence classes, we define

$$M\sigma = \{N \mid M\sigma N\} \ .$$

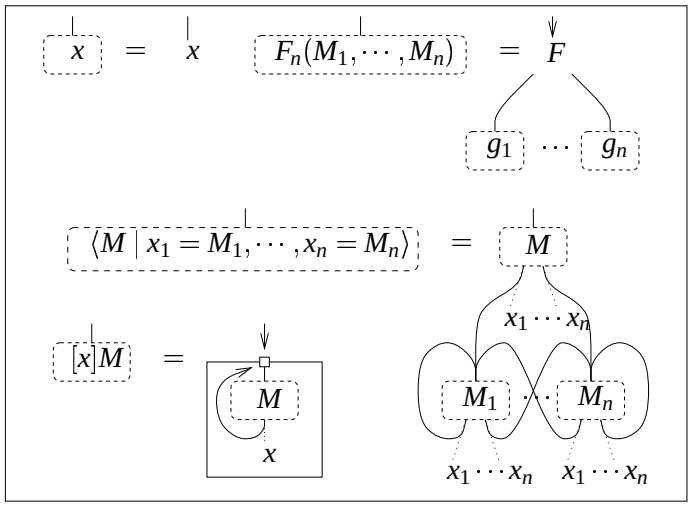


Figure 3.2: Graphical description of the graph constructions

Note the difference between M^{σ} and $M\sigma$. The former is a syntactic replacement, the latter a substitution.

So far we have drawn the graphs of cyclic terms without being precise about how to draw these pictures. We will now work towards a formal definition. What we would like is to define the graph of a term recursively. We have given such a definition pictorially in Fig. 3.2. Most constructions in this picture are self explaining, except for the dotted lines ending in variables. Such a dotted line ending in a variable indicates that every free variable in a graph is replaced by something else. In particular, it does not reflect any restriction on the amount of occurrences of that

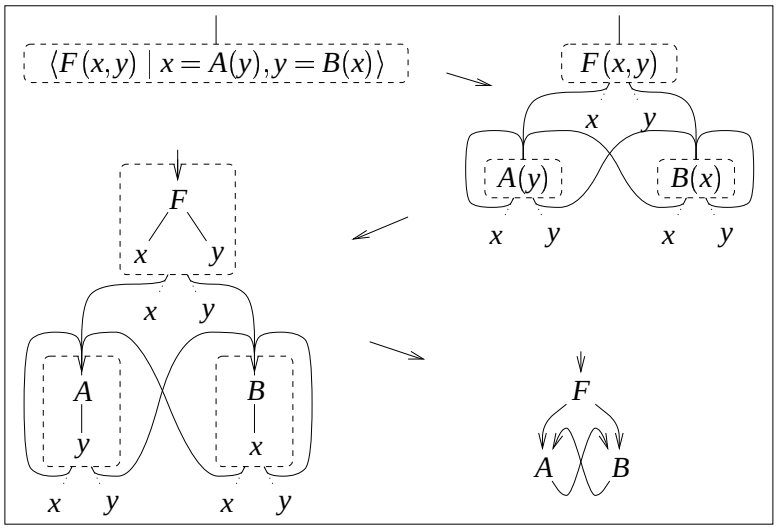


Figure 3.3: The construction of the graph of $\langle F(x,y) \mid x = A(y), y = B(x)\rangle$

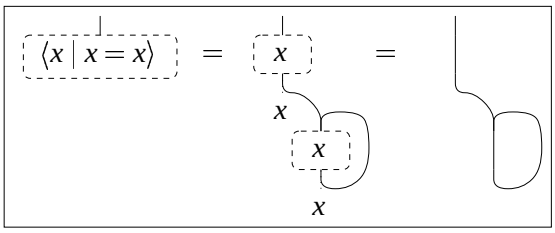


Figure 3.4: The construction of the graph of $\langle x \mid x = x \rangle$

free variable. In fig. 3.3 we have drawn many steps in the construction of the graph of $\langle F(x,y) \mid x = A(y), y = B(x) \rangle$. To give a completely formal definition, we will first need a formal definition of the graphs in the pictures. Afterwards we will need some tools to deal with a little problem in the definition, which shows when we try to construct the graph of, for example, $\langle x \mid x = x \rangle$. In Fig. 3.4 we have drawn this construction. The result is not a graph!

3.2 Term Graphs

A *first order term graph* consists of a set of nodes V, a function L which gives the labels of the nodes, a function A that gives the arguments of the nodes and a root. The root and the arguments can be either a node or a variable:

Definition 3.2.1 Given a signature Σ. A *first order term graph* over Σ is a tuple (V, L, A, r), where

$$\begin{array}{lcl} V & & \text{is a set;} \\ L & : & V \to \mathcal{F} \ ; \\ A & : & \{(v,i) \mid v \in V, 1 \leq i \leq \mathit{arity}(L(v))\} \to (V \oplus \mathcal{V}) \ ; \\ r & \in & V \oplus \mathcal{V} \ . \end{array}$$

Higher-order term graphs add the feature of abstraction. This feature is implemented with abstraction nodes, back-pointers (to represent bound variables) and scopes. An abstraction node is a node labeled $\Box$, which may have two types of edges pointing to it: normal edges and back-pointers. We denote back-pointers by extending the set of possible arguments with a new symbol $\overline{v}$ for every abstraction node v in the graph. For every abstraction node we also define a set of nodes that depends on this abstraction: the scope of the abstraction node. We formalize the scope with a function from abstraction nodes to sets of nodes. This function must satisfy several conditions:

(self) Every abstraction node is a member of its own scope.

(nest) Scopes are properly nested. That is, if an abstraction node is a member of the interior of the scope of another abstraction node then the entire scope of

the first abstraction node is a subset of the interior of the scope of the second abstraction node.

(scope) If a node has a back-pointer to an abstraction node then that node is a member of the scope of the abstraction node.

(closed) Every path from a node outside the scope of an abstraction node to a node inside the scope must pass through the abstraction node. One could also say that, if the argument of a node is a node in the interior of the scope of an abstraction node then the first node is a member of the scope of the abstraction node.

(root) The root should not be in the interior of any scope.

Definition 3.2.2 Given a signature Σ, without the symbol $\Box$. A *higher order term graph* over Σ is a tuple (V,L,A,S,r), such that

$$
\begin{array}{ll}
V \text{ is a set;} & \\
L : V \to \mathcal{F} \cup \{\Box\} & \\
A : \{(v,i) \mid v \in V, 1 \leq i \leq \mathit{arity}(L(v))\} \to V \to (V \oplus \{\overline{v} \in V \mid L(v) = \Box\} \oplus \mathcal{V}) & \\
S : \{v \in V \mid L(v) = \Box\} \to \mathcal{P}(V) \text{ such that} & \\
\quad \forall v \in V, L(v) = \Box : v \in S(v) & (\mathit{self}) \\
\quad \forall v,w \in V, L(v) = L(w) = \Box, v \in S^-(w) : S(v) \subset S^-(w) & (\mathit{nest}) \\
\quad \forall v,w \in V, A(v,i) = \overline{w} : \Longrightarrow \ v \in S(w) & (\mathit{scope}) \\
\quad \forall u,v,w \in V, w \in S^-(u), A(v,i) = w : v \in S(u) & (\mathit{closed}) \\
\quad \forall v \in V, L(v) = \Box : r \notin S^-(v) & (\mathit{root}) \\
r \in V \oplus \mathcal{V} &
\end{array}
$$

where the interior S^- of the scope of an abstraction node v is defined as

$$S^-(v) = S(v) \setminus \{v\} \quad .$$

Definition 3.2.3 In a graph $(V,L,A(,S),r)$ an *annotated path* of length n from v_0 to v_n is a sequence $v_0 i_1 v_1 \cdots i_n v_n$ of nodes interleaved with integers, such that for all $0 \leq j \leq n-1$ we have that $A(v_j, i_j) = v_{j+1}$. A *path* of length n from v_0 to v_n is a sequence of nodes $v_0 \cdots v_n$, such that there exists an annotated path $v_0 i_1 v_1 \cdots i_n v_n$.

Definition 3.2.4 A higher-order graph is a *tree* if the graph is acyclic and if every node in the graph is referenced exactly once by a normal pointer, where the root counts as a normal pointer.

Paths and annotated paths are usually denoted with the letter p . We denote the length of a path by $|p|$.

Definition 3.2.5 A graph $(V,L,A(,S),r)$ is *garbage free* if for all $v \in V$ there exists a path $r \cdots v$.

Definition 3.2.6 A graph $(V,L,A(,S),r)$ is *acyclic* if every path of the form $v\cdots v$ has length 0.

For cyclic higher order term graphs we need scopes. However, for higher order trees we prefer a notion with labels and without scopes. To bridge the gap we will now introduce the notion of labeled higher order term graph:

Definition 3.2.7 Given a signature Σ. A *labeled higher order term graph* over Σ is a tuple (V,L,A,r), where

$$\begin{array}{lcl} V & & \text{is a set;} \\ L & : & V \to \mathcal{F} \oplus [\mathcal{V}] \ ; \\ A & : & \{(v,i) \mid v \in V, 1 \leq i \leq \mathit{arity}(L(v))\} \to (V \oplus \mathcal{V}) \ ; \\ r & \in & V \oplus \mathcal{V} \ . \end{array}$$

where $[\mathcal{V}] = \{[x] \mid x \in \mathcal{V}\}$ is a set of unary function symbols.

Any scoped graph can be translated to a labeled graph, simply by omitting the scopes and by introducing fresh variables to replace the back-pointers to abstraction nodes. Formally:

Definition 3.2.8 Given a higher order graph $g \equiv (V,L,A,S,r)$. The *labeled translation* g^l of g is constructed as follows. Let $\{x_v \mid v \in V, L(v) = \Box\} \subset \mathcal{V}$ be given such that $x_v = x_w \implies v = w$. Then $g^l = (V,L',A',r)$, where

$$L'(v) = \begin{cases} [x_v] & , L(v) = \Box \\ L(v), & \text{otherwise} \end{cases}$$

$$A'(v,i) = \begin{cases} x_w & , A(v,i) = \overline{w} \\ A(v,i), & \text{otherwise} \end{cases}$$

Not every labeled term graph can be obtained as the translation of a scoped graph. The labeled graphs that can be obtained in that way are called scopable graphs.

Definition 3.2.9 Given a labeled graph G. We say that g is *scopable* if there exists a scoped graph g, such that G is isomorphic to g^l. We say that G is *well-formed* if G is garbage free, every label $[x]$ occurs at most once and if $[x]$ occurs then every path from the root to a node that has x as an argument passes through the node labeled $[x]$.

3.3 Indirection nodes

We saw in the introduction that the black hole was very important for graph rewriting and graph semantics of terms with μ and/or letrec. In the definition of graph semantics it was used to solve the problem of having to connect the end of an edge

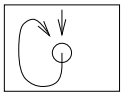


Figure 3.5: A self referencing indirection node

to the beginning of the edge itself. We will now introduce the concept of *indirection node*. We will use indirection nodes to make constructions easier by providing a natural way of introducing black holes where needed, but the concept has other uses as well.

An indirection node is a node labeled ○ with one argument. The informal semantics of an indirection node is that when there is a pointer to the indirection node it should be interpreted as a pointer to whatever the indirection node is pointing to. To this general case there is one exception: if an indirection node points to itself (see Fig. 3.5) then the semantics of that indirection node is a black hole. Note that the outgoing edge of the indirection node starts in the middle of the node rather than just outside it.

Indirection nodes have several practical and theoretical applications. One of the theoretical applications of indirection nodes is, that their use allows us to define a somewhat more refined notion of graph semantics than the notion informally defined in Fig.3.2. For example, the graphs of terms $F(x,x)$ and $\langle F(y,y) \mid y = x\rangle$ both are the middle graph of Fig. 3.6 according to the informal definition in Fig. 3.2. One of the possible modifications is to add an indirection node for every variable (See Fig. 3.7). If we do this then the graph of $F(x,x)$ is the left graph of Fig. 3.6 and the graph of $\langle F(y,y) \mid y = x\rangle$ is the right graph. Another possible modification is to add an indirection node for every equation (see Fig. 3.8). This yields the middle and right graphs, respectively. When we formalize the definition in Fig. 3.2, we will also use indirection nodes to solve the problem which arose in the construction of the graph of $\langle x \mid x = x\rangle$ (see Fig. 3.4).

One of the practical applications of indirection nodes is in the implementation of graph rewriting. In graph rewriting collapsing rules, such as

$$I(X) \to X$$

are hard to implement. The implementation is eased by encapsulating the X on the

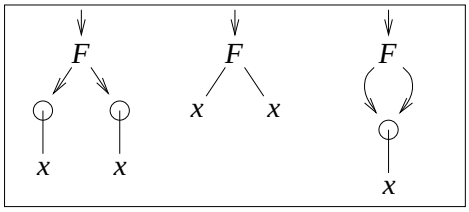


Figure 3.6: The enhanced resolution of indirection nodes.

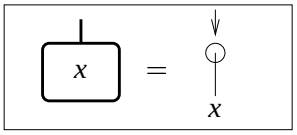


Figure 3.7: Graph semantics with indirection nodes for variables

right-hand side by an indirection node:

$$I(X) \to \mathsf{O}(X) \ .$$

Theoretically one then immediately removes this indirection node, but this is expensive to implement. It is better to leave the indirection nodes in the graph and to resolve references when they are needed.

In order to compute the semantics or *simplification* of a graph with indirection nodes we can remove the indirection nodes one by one. For every indirection node there are three possibilities, depicted in Fig. 3.9. First, the indirection node can have itself as its argument. In this case we replace the node by a black hole. Second, the indirection node can have a variable as an argument. In this case we delete the indirection node and replace all references to it by the variable. Third, the indirection node may have an arbitrary other node as its argument. Again we delete the indirection node and we replace every reference to the indirection node by a reference to its argument. If we simplify each graph in Fig. 3.6 then in each case the result is the middle graph in the figure. Formally simplification is defined as the normal form with respect to a rewrite relation:

Definition 3.3.1 The labeled rewrite relation $\xrightarrow[v]{\text{Sim}}$ on graphs is defined as follows: let (V,L,A,S,r) be a graph and let $v \in V$ be an indirection node with $A(v,1) = a$. Then we define:

$$\begin{array}{lll} (V,L,A,S,r) & \xrightarrow[v]{\text{Sim}} & (V,L' \cup \{(v,\bullet)\},A',S,r) \quad v \equiv a \\ (V,L,A,S,r) & \xrightarrow[v]{\text{Sim}} & (V \setminus \{v\},L',A',S,r') \qquad v \not\equiv a \end{array}$$

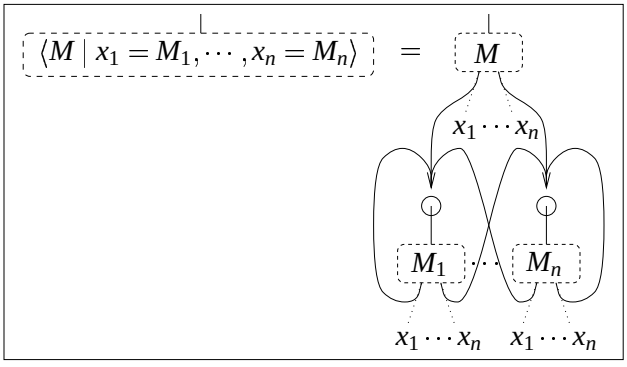


Figure 3.8: Graph semantics with indirections nodes for declarations

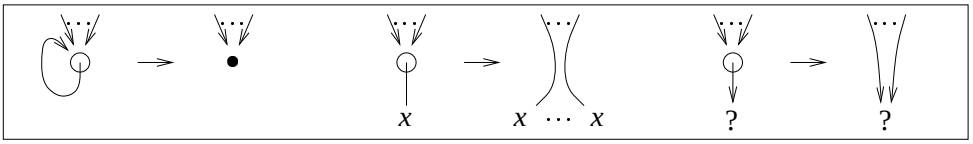


Figure 3.9: Removing indirection nodes

where

$$\begin{array}{lcl} L' & = & L \setminus \{(v, \circ)\} \\ A'(u,i) & = & \begin{cases} a & \text{, A(u,i)=v} \\ A(u,i) & \text{, otherwise} \end{cases} \\ r' & = & \begin{cases} a & \text{, r=v} \\ r & \text{, otherwise} \end{cases} \end{array}$$

The normal forms of $\xrightarrow[v]{\text{Sim}}$ defines a function Sim on equivalence classes of isomorphic graphs.

Note that if in this definition we have that $a \equiv v$ then we have that A' is nothing more than the restriction of A to the proper domain. Modulo isomorphism the rewrite relation $\xrightarrow{\text{Sim}}$ is confluent. As a matter of fact, it is almost confluent. The only problem is that there is a critical pair if we have two indirection nodes that have the other as argument. In this case each node is a redex and the contraction of the redex influences the other redex. We can see in Fig. 3.10 that the results of contracting the redexes are not identical. However, they are isomorphic, which is sufficient.

Lemma 3.3.2 The function Sim is well-defined.

Proof. We will show that $\xrightarrow{\text{Sim}}$ is terminating and confluent modulo isomorphism. Termination is easy because every step decreases the number of indirection nodes in the graph. Confluence up to isomorphism follows from the following two dia-

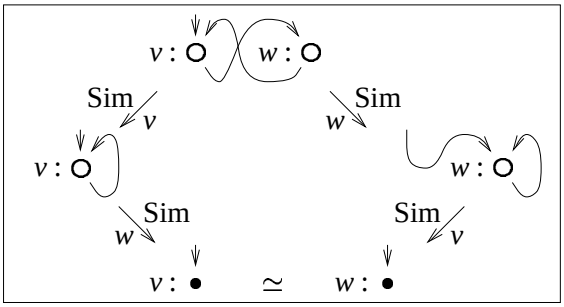
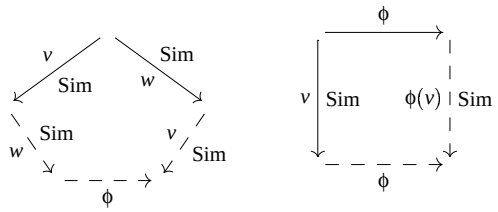


Figure 3.10: the critical pair of $\xrightarrow{\text{Sim}}$

grams:



where $\xrightarrow[\phi]{}$ denotes the image under an isomorphism ϕ. □

3.4 Semantics

In this section we define the graph semantics of cyclic terms and the unwinding semantics of graphs. The definition of the graph semantics of a term, also known as the graph of that term, follows the intuition of Fig. 3.2. All clauses involve straightforward constructions, except the one for letrec. The clause for letrec is composed of the construction in Fig. 3.8, followed immediately by simplification.

Definition 3.4.1 Given a cyclic term M, we recursively define the graph of M, denoted $[\![M]\!]$ as follows:

$$\begin{array}{ll}
[\![x]\!] & = x \\
[\![F(M_1,\cdots,M_n)]\!] & = F([\![M_1]\!],\cdots,[\![M_n]\!]) \\
[\![[x]M]\!] & = [x][\![M]\!] \\
[\![\langle M_0 \mid x_1 = M_1,\cdots,x_n = M_n\rangle]\!] & = \mathrm{Sim}(\langle [\![M_0]\!] \mid x_1 = [\![M_1]\!],\cdots,x_n = [\![M_n]\!]\rangle)
\end{array}$$

where the right-hand sides use the constructions from Table 3.2.

Example 3.4.2 In Fig. 3.3 we gave a pictorial construction of the graph of $\langle F(x,y) \mid x = A(y), y = B(x)\rangle$. We will now construct it using the formal definition. In Fig. 3.11 we give the same construction in a quasi pictorial format, because pictures of graphs are more easily parsed than syntax.

$$\begin{array}{l}
[\![\langle F(x,y) \mid x = A(y), y = B(x)\rangle]\!] \\
\quad = \langle [\![F(x,y)]\!] \mid x = [\![A(y)]\!], y = [\![B(x)]\!]\rangle \\
\quad = \langle F([\![x]\!],[\![y]\!]) \mid x = [\![A(y)]\!], y = [\![B(x)]\!]\rangle \\
\quad = \langle F((\emptyset,\emptyset,\emptyset,\emptyset,x),(\emptyset,\emptyset,\emptyset,\emptyset,y)) \mid x = [\![A(y)]\!], y = [\![B(x)]\!]\rangle \\
\quad = \langle (\{u\},\{(u,F)\},\{(u,xy)\},\emptyset,u) \mid x = [\![A(y)]\!], y = [\![B(x)]\!]\rangle \\
\quad = \langle (\{u\},\{(u,F)\},\{(u,xy)\},\emptyset,u) \mid x = (\{v\},\{(v,A)\},\{(v,y)\},\emptyset,v), \\
\qquad\qquad\qquad\qquad\qquad\qquad y = (\{w\},\{(w,B)\},\{(w,x)\},\emptyset,w)\rangle \\
\quad = \mathrm{Sim}(\{u,v,w,v',w'\},\{(u,F),(v,A),(w,B),(v',\mathsf{O}),(w',\mathsf{O})\}, \\
\qquad\qquad \{(u,v'w'),(v,w'),(w,v'),(v',v),(w',w)\},\emptyset,u) \\
\quad = (\{u,v,w\},\{(u,F),(v,A),(w,B)\},\{(u,vw),(v,w),(w,v)\},\emptyset,u)
\end{array}$$

Table 3.2 Graph constructions

$$
\begin{array}{lll}
x & = (\emptyset,\emptyset,\emptyset,\emptyset,x) & \\
F(g_1,\cdots,g_n) & = (V,L,A,S,r)\ , & \\
 & \text{where for some node } w & \\
 & V & = \{w\} \oplus V_{g_1} \oplus \cdots \oplus V_{g_n} \\
 & L(v) & = \begin{cases} F & , v \equiv w \\ L_{g_i}(v), & v \in V_{g_i} \end{cases} \\
 & A(v,i) & = \begin{cases} r_{g_i} & , v \equiv w \\ A_{g_i}(v,i), & v \in V_{g_i} \end{cases} \\
 & S(v) & = S_{g_i}(v),\ v \in V_{g_i}, L(v) = \square \\
 & r & = w \\
[x]g & = (V,L,A,S,r)\ , & \\
 & \text{where for some node } w & \\
 & V & = \{w\} \oplus V_g \\
 & L(v) & = \begin{cases} \square & , v \equiv w \\ L_g(v), & v \in V_g \end{cases} \\
 & A(v,i) & = \begin{cases} r_g & , v \equiv w \\ \overline{x} & , v \in V_g, A_g(v,i) = x \\ A_g(v,i), & v \in V_g, otherwise \end{cases} \\
 & S(v) & = \begin{cases} V & , v \equiv w \\ S_g(v), & v \in V_g, L(v) = \square \end{cases} \\
 & r & = w \\
\langle g_0 \mid x_1 = g_1,\cdots,x_n = g_n \rangle & = (V,L,A,S,r)\ , & \\
 & \text{where for some distinct nodes } w_1,\cdots,w_n & \\
 & V & = V_{g_0} \oplus \cdots \oplus V_{g_n} \oplus \{w_1,\cdots,w_n\} \\
 & L(v) & = \begin{cases} \bigcirc & , v \equiv w_i \\ L_{g_i}(v), & v \in V_{g_i} \end{cases} \\
 & A(v,i) & = \begin{cases} r_{g_i} & , v \equiv w_i \\ w_i & , v \in V_{g_i}, A_{g_i}(v,i) = x_i \\ A_{g_i}(v,i), & v \in V_{g_i} \end{cases} \\
 & S(v) & = S_{g_i}(v),\ v \in V_{g_i}, L(v) = \square \\
 & r & = r_{g_0}
\end{array}
$$

We now continue with the definition of the unwinding of a graph. Usually, the unwinding of a graph is obtained by ”unrolling” the graph to obtain a possibly infinite tree. Following this intuition, the unwinding of a first order term graph becomes a possibly infinite first order term. However, the ”unrolling” of a higher order term graph does not yield a possibly infinite higher order term. Because “unrolling” preserves the scopes, we get a possibly infinite higher order tree with scopes instead. We can translate this tree to a term by taking its labeled translation.

As usual the set of nodes of the tree unwinding is defined as the set of all paths starting at the root. Usually these paths are represented by strings of integers.

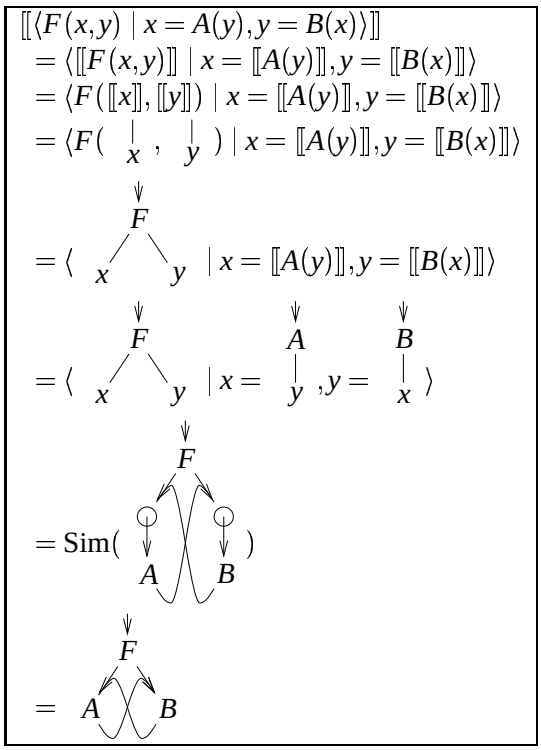


Figure 3.11: Quasi pictorial computation of $[\![\langle F(x,y) \mid x = A(y), y = B(x)\rangle]\!]$

Here, for convenience, we use a more verbose representation which keeps track of the nodes visited. For example, consider the following graph g (depicted on the left of Fig. 3.12):

$$\begin{array}{ll} (& \{v_1, v_2, v_3\}, \\ & \{v_1 \mapsto @, v_2 \mapsto \lambda, v_3 \mapsto @\}, \\ & \{v_1 \mapsto v_2\ v_2, v_2 \mapsto v_3, v_3 \mapsto \overline{v_2}\ \overline{v_2}\}, \\ & v_1 \\ ) & . \end{array} \tag{3.1}$$

The nodes of the unwinding of g (see the graph on the right of Fig. 3.12) are

$$\{v_1, v_1 1 v_2, v_1 1 v_2 1 v_3, v_1 2 v_2, v_1 2 v_2 1 v_3\}\ ,$$

instead of $\{\varepsilon, 1, 11, 2, 21\}$.

Definition 3.4.3 Given a graph $G \equiv (V, L, A, S, r)$. The *scoped unwinding* of g is the tree $g_u \equiv (V_u, L_u, A_u, S_u, r_u)$, where

- $V_u = \{r \mid r \in V\} \cup \{p\ v\ i\ w \mid p\ v \in V_u, A(v)_i = w, w \in V\}$.
- $L_u(p\ v) = L(v)$.

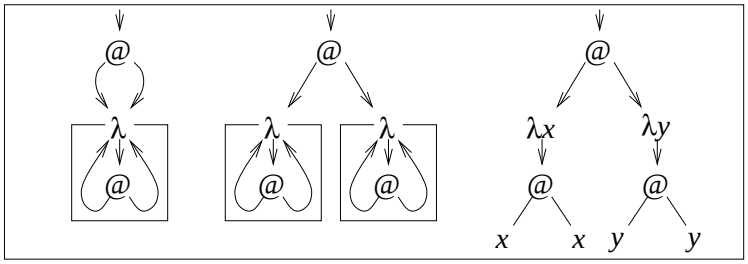


Figure 3.12: The unwindings of a higher-order graph

- $A_u(p\ v)_n = \begin{cases} p\ v\ n\ w, \text{ if } A(v)_n = w \\ \overline{p_1 w} \quad \text{, if } A(v)_n = \overline{w} \text{ and } p \equiv p_1\ w\ p_2 \text{ ,} \\ \qquad\qquad \text{where } w \text{ does not occur in } p_2 \\ x \qquad \text{, if } A(v)_n = x \end{cases}$

- $S_u(p\ v) = \{p\ v\} \cup \{p\ w\ i\ A(w)_i \mid p\ w \in S(p\ v), A(w)_i \in S^-(v)\}$

- $r_u = r$.

The *labeled unwinding* of g is the labeled translation g_u^l of g_u.

Note that the unwinding of a graph only depends on the accessible part, as in the first order case.

Example 3.4.4 The unwinding of graph g given in equation 3.1 and displayed in Fig. 3.12 is given below.

$$\begin{array}{rcl} V_u & = & \{v_1, v_1 1 v_2, v_1 1 v_2 1 v_3, v_1 2 v_2, v_1 2 v_2 1 v_3\} \\ L_u & = & \{v_1 \mapsto @, v_1 1 v_2 \mapsto \lambda, v_1 1 v_2 1 v_3 \mapsto @, v_1 2 v_2 \mapsto \lambda, v_1 2 v_2 1 v_3 \mapsto @\} \\ A_u & = & \{ \quad v_1 \mapsto v_1 1 v_2\ v_1 2 v_2, \\ & & \quad v_1 1 v_2 \mapsto \underline{v_1 1 v_2} 1 v_3, \\ & & \quad v_1 1 v_2 1 v_3 \mapsto \overline{v_1 1 v_2}\ \overline{v_1 1 v_2}, \\ & & \quad v_1 2 v_2 \mapsto \underline{v_1 2 v_2} 1 v_3, \\ & & \quad v_1 2 v_2 1 v_3 \mapsto \overline{v_1 2 v_2}\ \overline{v_1 2 v_2} \\ & & \} \\ S_u & = & \{v_1 1 v_2 \mapsto \{v_1 1 v_2, v_1 1 v_2 1 v_3\}, v_1 2 v_2 \mapsto \{v_1 2 v_2, v_1 2 v_2 1 v_3\}\} \\ r_u & = & v_1 \ . \end{array}$$

3.5 Conclusion

We have defined the syntax of terms with letrec and some of its important subsets. Also, we have defined first order term graphs and scoped/labeled higher order term graphs. We have defined what we mean by the graph of a term and by the unwinding of a graph. These two definitions may be composed to define the unwinding of a term. The unwinding of terms will be considered again in sections 7.1 and 8.1. We will now continue with a study of the expressive power of the let, μ and letrec constructs.

Chapter 4

Representability of graphs

In this chapter we will characterize the classes of first order graphs that can be represented by terms with let, μ, letrec or a combination of these three. Unless stated otherwise, we assume that the graphs in this chapter are first order, finite and garbage free.

4.1 Homeomorphic embedding

For terms with let and for μ-terms it is possible to give a short and intuitive description of the represented graphs (See [AK96]). We will take a different approach: rather than describing the graphs that are representable, we will describe the graphs that are not representable. To do this we will use the notion of *homeomorphic embedding*. To say that a certain graph is homeomorphically embedded in another graph means that in some way the second graph contains a subgraph that has the same structure as the first graph.

To explain what we mean by structure, it is convenient to view a term graph as a labeled rooted directed graph. As far as representability is concerned, function symbols and the order of the arguments don't matter. Thus, in order to decide if a certain term graph is representable, we can forget the labels and consider the underlying rooted directed graph of the term graph, which we will call the *skeleton* of the term graph. We draw pictures of skeleton graphs in the same way we draw pictures of term graphs, except that we draw every node as $\circ$. This can be seen in Fig. 4.1, where we introduce the graphs $\mathcal{H}$, $\mathcal{V}$ and $\mathcal{T}$.

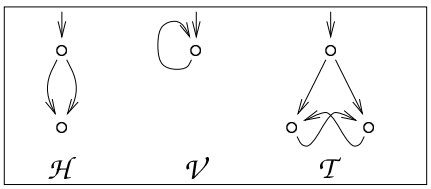


Figure 4.1: the graphs that define sharing

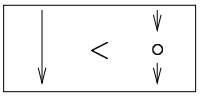


Figure 4.2: edge expansion

Informally, a skeleton graph is homeomorphically embedded in a term graph if we can find a subgraph of the skeleton of the term graph that can be obtained from the first skeleton graph by repeatedly inserting a node into an edge. (See Fig. 4.2 for a pictorial definition of the insertion operation.) In Fig. 4.3 we show how $\mathcal{V}$ is homeomorphically embedded in the graph of $\langle F(x,y) \mid x = A(z), y = F(z,y), z = B(x)\rangle$. Note that in the first step we insert a node into the edge that represents the root. This is expressed with the same pictorial definition as insertion into normal edges, but in the formal definition of node insertion ($<$) we get two cases: insertion of a node into the root edge ($<_{\text{root}}$) and insertion into an edge e ($<_e$).

Definition 4.1.1 A *skeleton graph* is a structure (V,E,s,d,r), where
- V is the non-empty set of nodes;
- E is the set of edges;
- $s : E \to V$ is a function indicating the source of every edge;
- $d : E \to V$ is a function indicating the destination of every edge;
- $r \in V$ is the root.

Definition 4.1.2 We define the *edge expansion* relation $\ll$ as the transitive reflexive closure of $<$, which is in turn defined as the union of the following two relations:

$$\begin{array}{ll} & (V,E,s,d,r) <_{\text{root}} (V \uplus \{v\}, E \uplus \{e\}, s \cup \{(e,v)\}, d \cup \{(e,r)\}, v) \ ; \\ \forall e \in E : & (V,E,s,d,r) <_e \quad (V \uplus \{v\}, (E \setminus \{e\}) \uplus \{e_1,e_2\}, s', d', r) \ , \end{array}$$

where

$$s'(e') = \begin{cases} s(e'), & e' \neq e \\ s(e), & e' = e_1 \\ v, & e' = e_2 \end{cases}$$

$$d'(e') = \begin{cases} d(e'), & e' \neq e \\ v, & e' = e_1 \\ d(e), & e' = e_2 \end{cases}$$

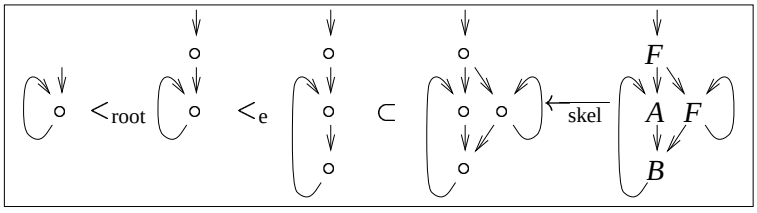


Figure 4.3: homeomorphic embedding

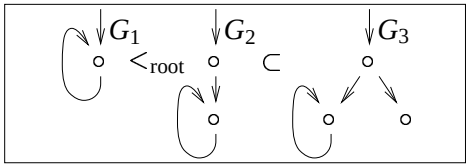


Figure 4.4: An example of expansion to a subgraph

Definition 4.1.3 Given two skeleton graphs $G_1 \equiv (V_1, E_1, s_1, d_1, r_1)$ and $G_2 \equiv (V_2, E_2, s_2, d_2, r_2)$, we say that G_1 is a *rooted subgraph* of G_2 ($G_1 \subset G_2$) if $V_1 \subset V_2$, $E_1 \subset E_2$, $s_1 \subset s_2$, $d_1 \subset d_2$ and $r_1 = r_2$.

Definition 4.1.4 Given a term graph $g \equiv (V, L, A, S, r)$ with a non-empty set of nodes, we define the skeleton of g as

$$\text{skel}(g) = (V, E, s, d, r) \ ,$$

where

$$\begin{array}{lcl} E & = & \{(v,i) \mid v \in V, 1 \leq i \leq \text{arity}(L(v)), A(v,i) \in V\} \\ s((v,i)) & = & v \\ d((v,i)) & = & A(v,i) \end{array}$$

We now have all the components to define homeomorphic embedding.

Definition 4.1.5 Given a skeleton graph G and a term graph h, we say that G is *homeomorphically embedded* in h, denoted $G \ll h$, if there exists a skeleton graph G', such that $G \ll G' \subset \text{skel}(h)$.

Note that the order of the edge expansion and the rooted subgraph matters. For example, in Fig. 4.4 we have drawn three graphs, of which the first graph can be transformed into the second using an root edge expansion, the second is a rooted sub-graph of the third. However, no graph G_4 such that $G_1 \subset G_4 \ll G_3$ exists. Supposing such a graph existed, we cannot have that $G_4 \equiv G_3$, because G_1 is not a rooted subgraph of G_3. That means that from G_4 to G_3 we must expand at least one edge. But that is impossible because the node that is added to a graph by an edge expansion will always have exactly one in-going edge and exactly one outgoing edge. Such nodes do not exist in G_3. Contradiction.

We do have that $g \subset \circ \ll h \implies g \ll \circ \subset h$. Given $g \subset \circ < h$ we can have two cases. The first case is if we expand an edge that was present in g. In this case the edge expansion and embedding are independent and we have that $g < \circ \subset h$. The second case is if we expand an edge that was added by the embedding. In this case we have that $g \subset h$. From this we may conclude that $g \subset \circ \ll h \implies g \ll \circ \subset h$.

From these two facts we can conclude that we have that

$$g \ll \circ \subset h \iff g(\ll \cup \subset)^* h \ .$$

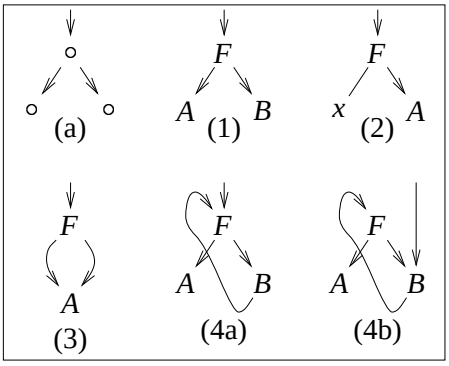


Figure 4.5: A collection of graphs.

Example 4.1.6 In Fig. 4.5 we have drawn a collection of graphs. There is a skeleton graph (a) and five term graphs (1,2,3,4ab). For all five term graphs we will now answer the question if (a) is homeomorphically embedded in it and why.

(1) We have that (a)=skel(1). Hence (a) is homeomorphically embedded in (1).

(2) The variable x is not a node. Therefore (a) is not homeomorphically embedded in (2).

(3) Even though there exists a rooted homomorphism from (a) to this graph, we do not have that (a) is homeomorphically embedded in this graph. The reason is that every node in the skeleton graph has to correspond to a unique node in the term graph.

(4a) The extra edge with respect to (1) does not matter, we have that (a)$\subset$skel(4a) and hence that (a) is homeomorphically embedded in (4a).

(4b) We do not have that (a) is homeomorphically embedded in (4b). The reason is that the node with B is needed as both the image of the bottom right node of (a) and as a node in the path from the root to the image of the top node of (a).

With the notion of homeomorphic embedding and the graphs $\mathcal{H}$, $\mathcal{V}$ and $\mathcal{T}$ we can precisely define what horizontal, vertical and twisted sharing are:

Definition 4.1.7 A term graph has *horizontal sharing* if it has a homeomorphic embedding of $\mathcal{H}$, a term graph has *vertical sharing* if it has a homeomorphic embedding of $\mathcal{V}$ and a term graphs has *twisted sharing* if it has a homeomorphic embedding of $\mathcal{T}$.

Note that if a graph has twisted sharing, it also has both horizontal and vertical sharing. The converse does not hold: if a graph has both horizontal sharing and vertical sharing then it does not necessarily have twisted sharing. See for example Fig. 4.6. The homeomorphic embeddings of $\mathcal{H}$ and $\mathcal{V}$ are easily recognized, but

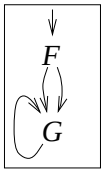


Figure 4.6: A graph that has horizontal and vertical sharing, but no twisted sharing

since the graph has only two nodes it is impossible that $\mathcal{T}$ is homeomorphically embedded in this graph.

It would have been simpler to consider directed graphs instead of rooted directed graphs. However, we need the roots for the following reasons: in Fig. 4.7 we have drawn the graph of $\mu x.A(F(x,x))$ twice. This should be a graph that has only vertical sharing. According to definition 4.1.5 we have that neither $\mathcal{H}$ nor $\mathcal{T}$ is homeomorphically embedded: there simply are not enough nodes to embed $\mathcal{T}$ and if we try to embed $\mathcal{H}$ then we find that the wrong node has to become the root. However, from the second picture of the graph it is clear that the unrooted version of $\mathcal{H}$ is homeomorphically embedded in this graph. This explains both the difference between rooted and non-rooted homeomorphic embedding and why we choose the rooted version. Similarly, the pictures in Fig. 4.8 show that the root in the definition of twisted sharing makes a difference and is needed. In the definition of vertical sharing the root makes no difference. Nevertheless, we include it for greater uniformity.

Distinguishing which kind of sharing is present in a graph with garbage is hard. Consider the two graphs in Fig. 4.9. Intuitively, the first graph is a graph with horizontal sharing and the second a graph with vertical sharing. Nevertheless, these graphs are isomorphic.

4.2 Representation by let

The class of term graphs that is represented by terms with let is the class of term graphs with only horizontal sharing. In other words, it is the class of term graphs that contain no vertical sharing:

Theorem 4.2.1 A term graph can be represented by a term with let if and only if the graph contains no homeomorphic embedding of $\mathcal{V}$.

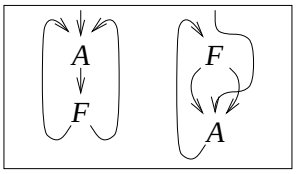


Figure 4.7: $\mu x.A(F(x,x))$

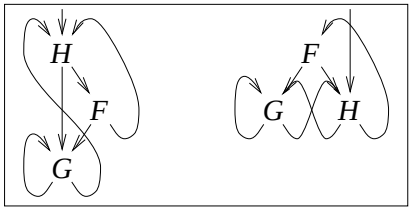


Figure 4.8: $\mu x.\mathsf{let}\ y = \mu z.G(z,x)\ \mathsf{in}\ H(y,F(y,x))$

Proof. Given a term graph. We have two cases, depending on whether or not this term graph has a homeomorphic embedding of $\mathcal{V}$:

- If the term graph has a homeomorphic embedding of $\mathcal{V}$ then it contains a cycle. By structural induction one can show that it is impossible that a cycle shows up in the graph semantics of a term with let. Thus, it is impossible to represent the term graph with a term with let.

- If the graph has no homeomorphic embedding of $\mathcal{V}$ then by induction on the number of nodes of the graph we will construct a term with let that represents the graph.

  (i) If the graph has no nodes then its root must be a variable and this variable is also a representation of the graph.

  (ii) If the graph g has one or more nodes then there must be a leaf node v in the graph. (Because the graph is acyclic and finite there must be a leaf node.) Assume that the label of the leaf node v is F and that its arguments are $x_1,\cdots,x_n$. Let y be a fresh variable. Let g' be the graph obtained from g by removing the node v and by using y in all places where v was used as an argument. Then g' still has no homeomorphic embedding of $\mathcal{V}$, so by induction hypothesis g' can be represented by a term with let. If M is such a representation of g' then g is represented by

$$\mathsf{let}\ y = F(x_1,\cdots,x_n)\ \mathsf{in}\ M\ .$$

□

Example 4.2.2 In Fig. 4.10 we have shown how to apply the construction of a term with let to represent a graph. Starting with the leftmost graph we have only

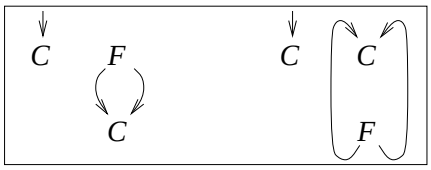


Figure 4.9: Graph with sharing and garbage

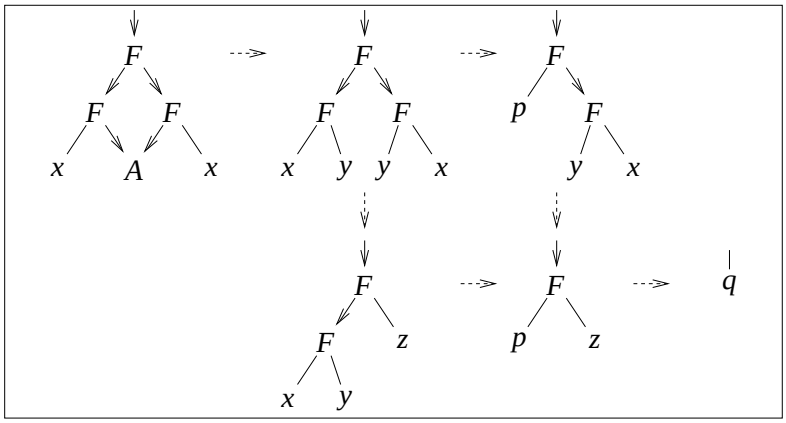


Figure 4.10: Sequential decomposition of a term graph

one choice for a leaf-node: the bottom node. We choose y as fresh variable and proceed. In the next step we have two choices for the leaf-node. After another step, we end up with the same graphs due to a smart choice of fresh variables. In the end the representation we get if we follow the top decomposition path is

$$\text{let } y = A \text{ in let } p = F(x,y) \text{ in let } z = F(y,x) \text{ in let } q = F(p,z) \text{ in } q$$

and the one we get if we follow the bottom path is

$$\text{let } y = A \text{ in let } z = F(y,x) \text{ in let } p = F(x,y) \text{ in let } q = F(p,z) \text{ in } q \quad .$$

Note that a much more efficient representation is possible by eliminating the lets whose equation is only used once:

$$\text{let } y = A \text{ in } F(F(x,y),F(y,x)) \quad .$$

4.3 Representation by μ

The class of term graphs that is represented by μ-terms is the class of term graphs with only vertical sharing. In other words, it is the class of term graphs that contain no horizontal sharing:

Theorem 4.3.1 A term graph can be represented by a term with μ if and only if the graph contains no homeomorphic embedding of $\mathcal{H}$.

Proof. Given a term graph. We have two cases, depending on whether or not this term graph has a homeomorphic embedding of $\mathcal{H}$:

- If the term graph has a homeomorphic embedding of $\mathcal{H}$ then there must be two nodes u and v in the graph that correspond to the top and bottom nodes of $\mathcal{H}$. There also must be disjoint acyclic paths p from the root to u and

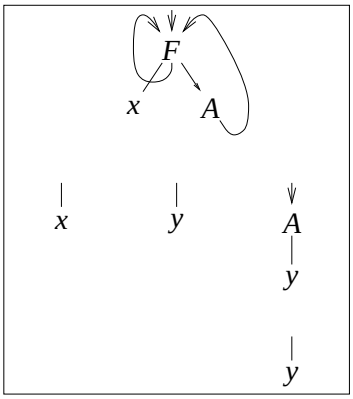


Figure 4.11: recursive decomposition of a graph

p_1, p_2 from u to v. Thus, there are two distinct acyclic paths $p; p_1$ and $p; p_2$ from the root to v. By structural induction one can prove that in the graph semantics of a term with μ there is a unique acyclic path from the root to every node. Therefore, the term graph cannot be represented by a term with μ.

- If the graph has no homeomorphic embedding of $\mathcal{H}$ then by induction on the number of nodes of the graph we will construct a term with μ that represents the graph.

  (i) If the graph has no nodes then its root must be a variable x. The term x represents the graph.

  (ii) If the graph g has one or more nodes then the root is a node v. Let x be a fresh variable. The graphs g_i are obtained from g by removing the node v, using x as argument where v was used before and by taking the subgraph starting with $A(v,i)$. If $A(v,i) = v$ then the root is x. Due to the fact that that there is no horizontal sharing in g we have that there is no horizontal sharing in each g_i. Because we remove the node v we have that each g_i is smaller than g. By induction hypothesis we can represent every g_i with a μ-term M_i. Let F be the label of v. The graphs g_i do not have any nodes in common because there is no horizontal sharing in g. Therefore, we have that g is represented by

$$\mu x.F(M_1, \cdots, M_n) \quad .$$

□

Example 4.3.2 Let us follow the construction in the previous proof to obtain a representation of the graph g on the first row of Fig. 4.11.

We choose y as the fresh variable and then compute g_1, g_2 and g_3 which are drawn on the second row of the figure. The first two graphs on the second row are

represented by x and y, respectively. In order to obtain a representation of the third graph, we have to decompose it. Now we choose z as our fresh variable. The result if the graph shown on the third row. This graph is represented by y. Thus, the last graph in the second row is represented by $\mu z.A(y)$ and the graph on the first row by

$$\mu y.F(x,y,\mu z.A(y)) \quad .$$

Note that the μz is superfluous. That is, $\mu y.F(x,y,A(y))$ is also a representation. Adding an occur check would make the algorithm optimal, we have omitted this check because we are only interested in the existence of a representation and not in the most efficient representation possible.

4.4 Representation by let and μ

The class of term graphs that is represented by μ-terms with let is the class of term graphs that only have horizontal and/or vertical sharing. In other words, it is the class of term graphs that do not have twisted sharing:

Theorem 4.4.1 A term graph can be represented by a term with let and μ if and only if the graph contains no homeomorphic embedding of $\mathcal{T}$.

Proof. Given a term graph g. We have two cases, depending on whether or not this term graph has a homeomorphic embedding of $\mathcal{T}$:

- If g has a homeomorphic embedding of $\mathcal{T}$ and is the graph of a term with μ and let then we can derive a contradiction.

  Using the recursive definition of graph semantics, we can tag all the edges. The edges that were introduced with the clause for μ are tagged "up"; all the others are tagged "down". The tagged graph satisfies the following two properties:

  - Every cyclic path includes at least one edge that has an "up" tag.
  - Every acyclic path from the root to a node contains only edges with the "down" tag.

  Because g contains a homeomorphic embedding of $\mathcal{T}$, there exist three nodes u,v,w in g and five disjoint acyclic paths p_1 from the root to u, p_2 form u to v, p_3 from u to w, p_4 from v to w and p_5 from w to v. Because the paths are disjoint, we have that $p_1p_2p_4$ is an acyclic path from the root to w. Such a path cannot contain "up" edges, so in particular p_4 contains no "up" edges. Similarly $p_1p_3p_5$ is an acyclic path from the root to v. Hence, p_5 contains no "up" edges. However, p_4p_5 is a cyclic path, which must contain at least one "up" edge. Contradiction.

- If g has no homeomorphic embedding of $\mathcal{T}$ then we will inductively construct a term with let and μ that represents the graph.

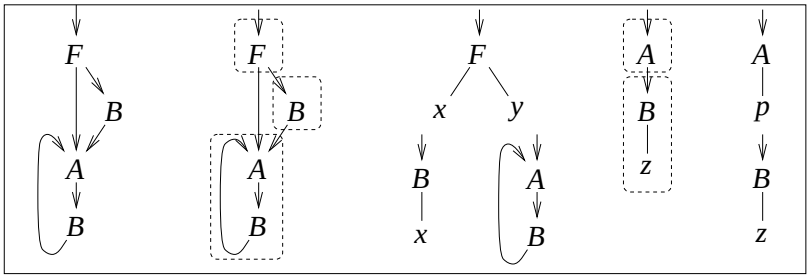


Figure 4.12: Recursive planar decomposition of a graph.

(i) If the graph has no nodes then its root must be a variable x. The term x represents the graph.

(ii) If the graph g has one or more nodes then the root is a node v. Let x be a fresh variable. The graph g' is obtained from g by replacing every occurrence of g as an argument by x. We can decompose g' into maximal strongly connected components. That is, we can decompose g' into subgraphs called planes, such that between every two nodes in the plane there exists a path and such that if a node is in the plane and that node lies on a cycle then every node on the cycle is in the plane. Every plane has a single root. That is, there is exactly one node in the plane that is referred to from outside the plane. Therefore, we can view a plane as a node. If g' consists of a single plane then this plane must be a function symbol F with arguments $x_1, \cdots x_n$ and we can represent the plane with the term $M = F(x_1, \cdots, x_n)$. If g' consists of more than one plane then we can apply our induction hypothesis to every plane. By using the algorithm we used in the proof of Thm. 4.2.1, we can combine the representations of the planes into a representation M of g'. If x does not occur free in M then M also represents g. Otherwise, we can represent g with $\mu x.M$.

□

Example 4.4.2 In Fig. 4.12 we have applied the construction to a small graph. We start with the leftmost graph. First, we must replace every reference to the root with a variable w. Then, we must decompose the graph into planes. The result of these steps is drawn in the second picture, where a plane is indicated with a dashed line. We must now consider the planes themselves, which means that we have to consider the group of graphs which form the third picture. The top and left graphs of this group are trivial and represented by $F(x,y)$ and $B(x)$, respectively. The right graph must be decomposed again, which is done in the fourth an fifth pictures. The resulting representation of the graph in the first picture is

$$\mathsf{let}\ x = \mu z.\mathsf{let}\ p = B(z)\ \mathsf{in}\ A(p)\ \mathsf{in}\ \mathsf{let}\ y = B(x)\ \mathsf{in}\ F(x,y) \quad .$$

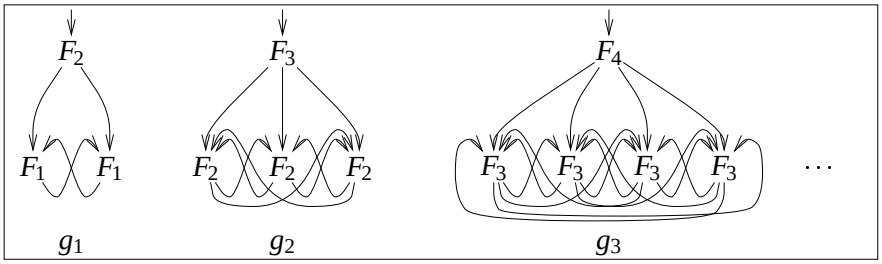


Figure 4.13: A set of graphs that can only be represented by terms with letrec

4.5 Representation by letrec

The class of term graphs represented by terms with letrec is the class of all graphs.

Theorem 4.5.1 Every term graph can be represented by a term with letrec.

Proof. Given a term graph $g \equiv (V, L, A, r)$. Let $x_v, v \in V$ be distinct variables, such that for no v we have that x_v occurs free in g. Let $x_y = y$ for every free variable in g and let n_v be the arity of $L(v)$. We can represent g with

$$\langle x_r \mid x_v = L(v)(x_{A(v,1)}, \cdots, x_{A(v,n_v)}); v \in V \rangle \quad .$$

□

Unlike the let, the letrec allows a random number of equations. A natural question to ask is what effect the number of equations has on the expressive power the letrec. To give a partial answer to this question let us consider the letrec with n bindings:

$$\mathsf{letrec}_n\ x_1 = M_1, \cdots, x_n = M_n\ \mathsf{in}\ M \quad .$$

For the special case of letrecs with a single binding we can easily give a complete answer:

Proposition 4.5.2 The class of graphs that are representable by terms with letrec_1 is the class of graphs that is representable by terms with let and μ.

Proof. All of the following rewrite rules preserve the graph semantics:

$$\begin{array}{lcl} \mathsf{letrec}_1\ x = M\ \mathsf{in}\ N & \rightarrow & \mathsf{let}\ x = \mu x.M\ \mathsf{in}\ N \\ \mathsf{let}\ x = M\ \mathsf{in}\ N & \rightarrow & \mathsf{letrec}_1\ x = M\ \mathsf{in}\ N \\ \mu x.M & \rightarrow & \mathsf{letrec}_1\ x = M\ \mathsf{in}\ x \end{array}$$

With these rules we can translate a given term with let and μ to a term with letrec_1 and vice versa. □

In general we can say that the letrec_n is not as powerful as the letrec_{n+1}. For example, if we look at the graphs $g_1, g_2, \ldots$ that are defined in Fig. 4.13 then we can prove that each g_i can be represented by a term with letrec_j if and only if

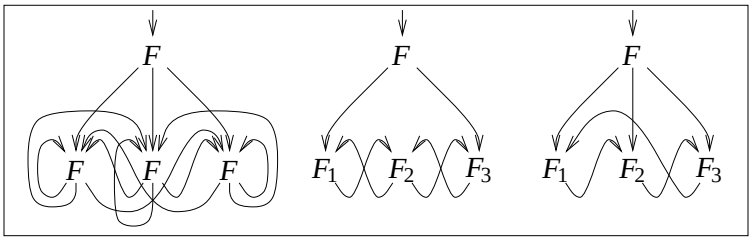


Figure 4.14: Graphs that are not representable by terms with letrec_2

$i \leq j$. A complete characterization is difficult. For example, the graphs in Fig. 4.14 cannot be represented using letrecs with at most 2 equations. Moreover, if one leaves out any edge in one of the last two graphs then the resulting graph can be represented by a term with letrec_2. Also, it is impossible to homeomorphically embed the skeleton of one graph into the other. The conclusion is that we cannot characterize letrec_2 with a single skeleton graph, but that we will have to use a set of graphs.

In such a set of graphs we may have to include skeletons, which have a node with three or more outgoing edges. The presence of such nodes poses a problem with the definition of homeomorphic embedding. Consider the graphs in Fig. 4.15. The skeleton graph on the left is not homeomorphically embedded in the term graph on the right. However, they look very similar. The problem is that the node with three arguments in the skeleton graph has to correspond to the two nodes labeled F in the term graph. To solve the problem, we define *weak homeomorphic embedding* by replacing the relation $\ll$ with $\lll$ in the definition of homeomorphic embedding. The relation $\lll$ is the least transitive and reflexive relation that includes $<_{\text{root}}$ and allows the expansion of nodes into trees, as depicted in Fig.4.16. Note that the edge expansion relation $<_e$ is contained in $<_{\text{node}}$: to extend an edge that begins in a certain node you can also extend the starting node.

Definition 4.5.3 Given a skeleton graph (V,E,s,d,r) and a node $v \in V$, we define

$$(V,E,s,d,r) <_{\text{node}} (V \uplus \{v'\}, E \uplus \{e\}, s', d' \cup \{(e,v')\}, r) \ ,$$

where s' is a function satisfying

$$s' \subset s \cup \{(e,v)\} \cup \{(e',v') \mid (e',v) \in s\} \ .$$

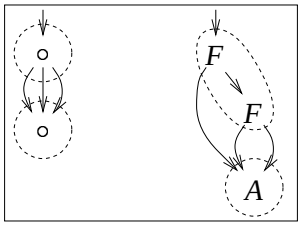


Figure 4.15: An example of generalized embedding

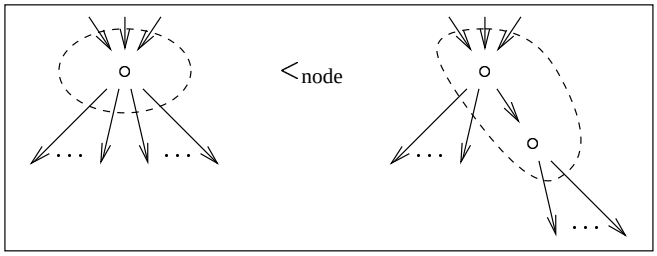


Figure 4.16: node expansion

Out intuition for the general case is that a graph which consists of a plane with $n+1$ nodes plus a root node cannot be represented by a letrec with n equations as long as that graph is not the result of a node expansion of a smaller graph. We can enforce that condition by requiring that every node in the plane has two distinct in-going edges. This intuition leads to the following educated guess:

Conjecture 4.5.4 Let $v_0, v_1, \cdots$ be a set of nodes. For $n > 1$ let $\mathcal{G}_n$ be the set of graphs g that satisfy the conditions:

- The nodes of g are $v_0, \cdots, v_n$.
- The root of g is v_0.
- There is no edge from v_i to v_0.
- For $i, j > 0$ there exists a path from v_i to v_j.
- There is no edge from v_i to v_i
- For $i > 0$ there are two distinct j_1, j_2 such that there are edges from v_{j_1} to v_i and from v_{j_2} to v_i.

The class of term graphs that is representable by terms with letrec_n is the class of graphs which contain no weak homeomorphic embedding of any element of $\mathcal{G}_{n+1}$.

In Fig. 4.17 we have drawn a skeleton graph, that is a member of $\mathcal{G}_4$, and a graph, that is not representable by letrec_3. The skeleton graph is not homeomorphically embeddable in the term graph. However, it is weakly homeomorphically embeddable. Thus, it is essential to use weak homeomorphic embedding. Note that we have that $\mathcal{G}_2 = \{\mathcal{T}\}$. So for $n = 1$ we already have a proof. Also note that given a graph $g \in \mathcal{G}_n$ we can find a graph $g' \in \mathcal{G}_{n+1}$, such that g is a subgraph of g'.

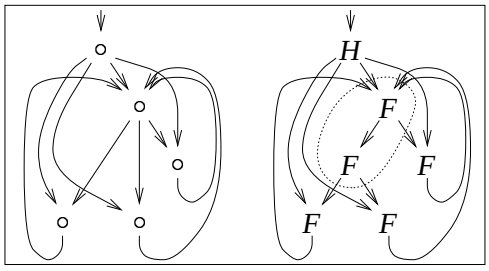


Figure 4.17:

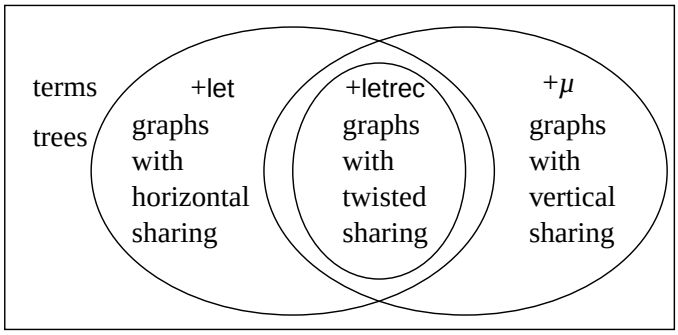


Figure 4.18: The classification as a Venn diagram

4.6 Conclusion

We now have a complete set of descriptions of the classes of graphs represented by terms with let, μ and/or letrec. This classification is nicely expressed in Fig. 4.18.

Chapter 5

Axiomatizations

In this chapter we will axiomatize the equivalence relations on terms, induced by the following equivalence relations on the graph semantics:

- isomorphism
- isomorphism modulo garbage collection
- bisimilarity
- isomorphism of labeled graphs
- isomorphism of tree unwinding

These axiomatizations are derived from the axiomatizations found in [AB97a, AB97b]. The differences are mostly in presentation and the technical details.

The first section of this chapter is devoted to the common component of all our axiomatizations: *equational logic*. This section is followed by several sections that are devoted to the axiomatization of one of the equivalence relations. Each of these sections begins with an introduction of the necessary new axioms and an axiom system. This introduction is followed by some technical lemmas and the section will end with a theorem that summarizes the important results about the axiom system. In the sections about isomorphic graph semantics, graph semantics modulo garbage and isomorphic labeled graph semantics we also provide a complete rewrite system, whose conversion is precisely the provable equality of the axiom system and whose normal forms are flat terms.

5.1 Equational logic

All of the proof systems in this chapter have a small set of inference rules in common: *equational logic*. The inference rules of equational logic are given in Table 5.1. In this table M, N and P stand for terms and C for a context.

With just the inference rules of equational logic we cannot do very much. However, if we add axioms then the resulting proof system can prove precisely those

Table 5.1 Equational logic

Reflexivity	$\dfrac{}{M = M}$ (refl)
Symmetry	$\dfrac{M = N}{N = M}$ (symm)
Transitivity	$\dfrac{M = N \quad N = P}{M = P}$ (trans)
Compatibility	$\dfrac{M = N}{C[M] = C[N]}$ (comp)

terms equal that are convertible in the rewrite relation that is obtained from those axioms by replacing the equal signs with arrows. For example, if we extend equational logic with the axioms

$$\begin{array}{ll} \text{A1} & A(x,0) \quad = x \\ \text{A2} & A(x,S(y)) = S(A(x,y)) \end{array}$$

and refer to the result as $\mathcal{R}_{\mathbb{A}}$ then we can prove two terms M and N equal in $\mathcal{R}_{\mathbb{A}}$ if and only if M and N are convertible in the following term rewrite system:

$$\begin{array}{l} A(x,0) \quad \to x \\ A(x,S(y)) \to S(A(x,y)) \end{array}$$

If we can prove an equation $M = N$ in a proof system $\mathcal{R}$ then we write $\vdash_{\mathcal{R}} M = N$. For example we have that $\vdash_{\mathcal{R}_{\mathbb{A}}} A(O,S(O)) = A(S(O),O)$. A completely formal derivation of this fact is the following proof tree:

$$\dfrac{\dfrac{\dfrac{}{A(O,S(O)) = S(A(O,O))}\ \text{A2} \quad \dfrac{\dfrac{}{A(O,O) = O}\ \text{A1}}{S(A(O,O)) = S(O)}\ (\text{comp})}{A(O,S(O)) = S(O)}\ (\text{trans}) \quad \dfrac{\dfrac{}{A(S(O),O) = S(O)}\ \text{A1}}{S(O) = A(S(O),O)}\ (\text{symm})}{A(O,S(O)) = A(S(O),O)}\ (\text{trans})$$

We can also give a proof in a much more compact style:

$$\begin{array}{rll} A(O,S(O)) &= S(A(O,O)) & A2 \\ &= S(O) & A1 \\ &= A(S(O),O) & A1 \end{array}$$

Essentially this proof is a conversion, which means that a proof in this style exists if and only if a proof tree exists.

Besides the provable equality we will also have semantical truth. The semantical truth will depend on semantics for the terms $[\![.]\!]$ and an equivalence relation R on those semantics. We will write $\models_R M = N$ if we have that $[\![M]\!] R [\![N]\!]$. This gives rise to two properties of a proof system $\mathcal{R}$. We say that $\mathcal{R}$ is sound with respect to R if

$$\vdash_{\mathcal{R}} M = N \implies \models_R M = N$$

and that $\mathcal{R}$ is complete with respect to R if

$$\models_R M = N \implies \vdash_{\mathcal{R}} M = N \ .$$

The usual semantics for the terms in our example would be parameterized with a valuation (a function from variables to natural numbers) and yield a natural number. As this scheme doesn't fit our theory, we cheat a bit by using a non-parameterized semantics that yields a function from valuations to natural numbers. Apart form this, the definition of the semantics is the usual recursive one:

$$\begin{array}{lcl} [\![x]\!] & = & f \mapsto f(x) \\ [\![O]\!] & = & f \mapsto 0 \\ [\![S(M)]\!] & = & f \mapsto 1 + [\![M]\!](f) \\ [\![A(M,N)]\!] & = & f \mapsto [\![M]\!](f) + [\![N]\!](f) \end{array}$$

For example,

$$[\![A(x,y)]\!] = f \mapsto f(x) + f(y) \ .$$

If we define the relation R_{A} as the equality relation on semantics then we have that $\mathcal{R}_{\mathrm{A}}$ is sound and complete with respect to R_{A}.

In the remaining sections of this chapter we assume that the semantics used is the graph semantics of terms. We will axiomatize several equivalence relations on those semantics, starting with isomorphism.

5.2 Graph semantics

In this section we will give a sound and complete axiomatization of isomorphism on the graphs of terms.

Definition 5.2.1 The proof system $\mathcal{R}_{\mathrm{graph}}$ is the extension of equational logic with the axioms in Table. 5.2

In the remainder of this section we will prove that $\mathcal{R}_{\mathrm{graph}}$ is sound and complete with respect to isomorphism. Because of the large number of axioms in $\mathcal{R}_{\mathrm{graph}}$ these proofs will be quite extensive.

A pictorial "proof" of the soundness of $\mathcal{R}_{\mathrm{graph}}$ is given in Fig. 5.1. In this figure we have constructed the graph semantics of every left and right-hand side of every axiom. In these graphs we have indicated subgraphs that are the semantics of a letrec with dashed boxes and we have used dotted arrows to indicate the presence of zero or more real edges. For the last three axioms we have omitted the simplification step of the construction, because after simplification there would not be any difference. Let us now give the formal proof:

Theorem 5.2.2 The proof system $\mathcal{R}_{\mathrm{graph}}$ is sound with respect to isomorphism.

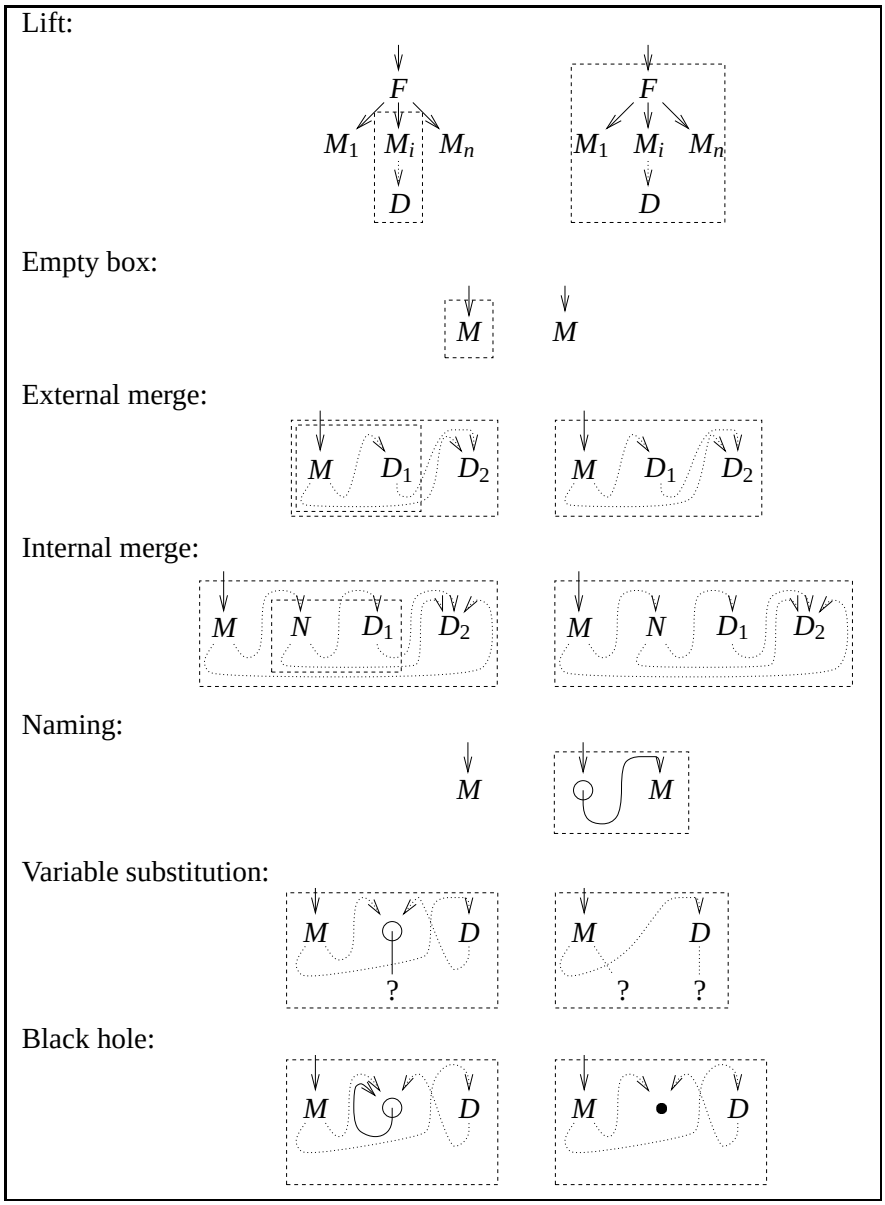


Figure 5.1: Pictorial description of $\mathcal{R}_{\text{graph}}$

Table 5.2 the axioms of $\mathcal{R}_{\text{graph}}$

Lift:

$$F(M_1,\cdots,\langle M_i \mid D\rangle,\cdots,M_n) = \langle F(M_1,\cdots,M_i,\cdots,M_n) \mid D\rangle$$

Empty box garbage collection:

$$\langle M \mid\rangle = M$$

Merge:

$$\langle\langle M \mid D_1\rangle \mid D_2\rangle = \langle M \mid D_1, D_2\rangle$$

$$\langle M \mid x = \langle N \mid D_1\rangle, D_2\rangle = \langle M \mid x = N, D_1, D_2\rangle$$

Naming:

$$M = \langle x \mid x = M\rangle \qquad x \text{ fresh}$$

Variable substitution:

$$\langle M \mid x = y, D\rangle = \langle M[x := y] \mid D[x := y]\rangle \qquad x \not\equiv y$$

Black hole:

$$\langle M \mid x = x, D\rangle = \langle M \mid x = \bullet, D\rangle$$

Proof. Because isomorphism is an equivalence relation we have that the reflexivity, symmetry and transitivity rules are sound. Because the definition of graph semantics is based on structural induction we have that the compatibility rule is sound. Thus, we only have to consider the axioms in Table 5.2:

Lift:

$$\begin{aligned}
&[\![F(M_1,\cdots,\langle M_i \mid x_1 = N_1,\cdots,x_k = N_k\rangle,\cdots,M_n)]\!]\\
&\quad = F([\![M_1]\!],\cdots,[\![\langle M_i \mid x_1 = N_1,\cdots,x_k = N_k\rangle]\!],\cdots,[\![M_n]\!])\\
&\quad = F([\![M_1]\!],\cdots,\mathrm{Sim}(\langle [\![M_i]\!] \mid x_1 = [\![N_1]\!],\cdots,x_k = [\![N_k]\!]\rangle),\cdots,[\![M_n]\!])\\
&\quad = \mathrm{Sim}(F([\![M_1]\!],\cdots,\langle [\![M_i]\!] \mid x_1 = [\![N_1]\!],\cdots,x_k = [\![N_k]\!]\rangle,\cdots,[\![M_n]\!]))\\
&\quad = \mathrm{Sim}(\langle F([\![M_1]\!],\cdots,[\![M_i]\!],\cdots,[\![M_n]\!]) \mid x_1 = [\![N_1]\!],\cdots,x_k = [\![N_k]\!]\rangle)\\
&\quad = \mathrm{Sim}(\langle [\![F(M_1,\cdots,M_i,\cdots,M_n)]\!] \mid x_1 = [\![N_1]\!],\cdots,x_k = [\![N_k]\!]\rangle)\\
&\quad = [\![\langle F(M_1,\cdots,M_i,\cdots,M_n) \mid x_1 = N_1,\cdots,x_k = N_k\rangle)]\!]
\end{aligned}$$

Empty box: Trivial.

External merge:

$$\begin{aligned}
&[\![\langle\langle M_0 \mid x_1 = M_1,\cdots,x_k = M_k\rangle \mid y_1 = N_1,\cdots,y_n = N_n\rangle]\!]\\
&\quad = \mathrm{Sim}(\langle [\![\langle M_0 \mid x_1 = M_1,\cdots,x_k = M_k\rangle]\!] \mid \underbrace{y_1 = [\![N_1]\!],\cdots,y_n = [\![N_n]\!]}_{D}\rangle)\\
&\quad = \mathrm{Sim}(\langle \mathrm{Sim}(\langle [\![M_0]\!] \mid x_1 = [\![M_1]\!],\cdots,x_k = [\![M_k]\!]\rangle) \mid D\rangle)\\
&\quad = \mathrm{Sim}(\langle\langle [\![M_0]\!] \mid x_1 = [\![M_1]\!],\cdots,x_k = [\![M_k]\!]\rangle \mid D\rangle)\\
&\quad = \mathrm{Sim}(\langle [\![M_0]\!] \mid x_1 = [\![M_1]\!],\cdots,x_k = [\![M_k]\!], y_1 = [\![N_1]\!],\cdots,y_n = [\![N_n]\!]\rangle)\\
&\quad = [\![\langle M_0 \mid x_1 = M_1,\cdots,x_k = M_k, y_1 = N_1,\cdots,y_n = N_n\rangle]\!]
\end{aligned}$$

Internal merge:

$$\begin{aligned}
&[\![\langle M_0 \mid x_1 = \langle M_1 \mid y_1 = N_1, \cdots, y_n = N_n\rangle, \underbrace{x_2 = M_2, \cdots, x_k = M_k}_{D}\rangle]\!] \\
&= \mathrm{Sim}(\langle [\![M_0]\!] \mid x_1 = [\![\langle M_1 \mid y_1 = N_1, \cdots, y_n = N_n\rangle]\!], [\![D]\!]\rangle) \\
&= \mathrm{Sim}(\langle [\![M_0]\!] \mid x_1 = \mathrm{Sim}(\langle [\![M_1]\!] \mid y_1 = [\![N_1]\!], \cdots, y_n = [\![N_n]\!]\rangle), [\![D]\!]\rangle) \\
&= \mathrm{Sim}(\langle [\![M_0]\!] \mid x_1 = \langle [\![M_1]\!] \mid y_1 = [\![N_1]\!], \cdots, y_n = [\![N_n]\!]\rangle, [\![D]\!]\rangle) \\
&= \mathrm{Sim}(\langle [\![M_0]\!] \mid x_1 = [\![M_1]\!], y_1 = [\![N_1]\!], \cdots, y_n = [\![N_n]\!], [\![D]\!]\rangle) \\
&= [\![\langle M_0 \mid x_1 = M_1, y_1 = N_1, \cdots, y_n = N_n, x_2 = M_2, \cdots, x_k = M_k\rangle]\!]
\end{aligned}$$

Naming: Given that

$$[\![M]\!] = (V, L, A, S, r) \quad .$$

We have that if x does not occur in M then

$$[\![\langle x \mid x = M\rangle]\!] = \mathrm{Sim}(V \oplus \{v\}, L \oplus \{v \mapsto \mathsf{O}\}, A \oplus \{(v,1) \mapsto r, S, v) = [\![M]\!] \quad .$$

The latter equality is because we know that there are no indirection nodes in $[\![M]\!]$, which means that v is the only indirection node.

Variable substitution:

$$\begin{aligned}
&[\![\langle M_0 \mid x_0 = x_1, x_1 = M_1, \cdots, x_n = M_n\rangle]\!] \\
&= \mathrm{Sim}(\langle [\![M_0]\!] \mid x_0 = [\![x_1]\!], x_1 = [\![M_1]\!], \cdots, x_n = [\![M_n]\!]\rangle) \\
&= \mathrm{Sim}(\langle [\![M_0[x_0 := x_1]]\!] \mid x_1 = [\![M_1[x_0 := x_1]]\!], \cdots, x_n = [\![M_n[x_0 := x_1]]\!]\rangle) \\
&= [\![\langle M_0[x_0 := x_1] \mid x_1 = M_1[x_0 := x_1], \cdots, x_n = M_n[x_0 := x_1]\rangle]\!]
\end{aligned}$$

More precisely, we have that

$$\begin{aligned}
&\langle [\![M_0]\!] \mid x_0 = [\![x_1]\!], x_1 = [\![M_1]\!], \cdots, x_n = [\![M_n]\!]\rangle \xrightarrow[\mathrm{Sim}]{} \\
&\quad \langle [\![M_0[x_0 := x_1]]\!] \mid x_1 = [\![M_1[x_0 := x_1]]\!], \cdots, x_n = [\![M_n[x_0 := x_1]]\!]\rangle \quad .
\end{aligned}$$

Black hole: We have that

$$[\![\bullet]\!] = (\{v\}, \{v \mapsto \bullet\}, \emptyset, \emptyset, v) \quad .$$

$$\begin{aligned}
[\![\langle x \mid x = x\rangle]\!] &= \langle (\emptyset, \emptyset, \emptyset, \emptyset, x) \mid x = (\emptyset, \emptyset, \emptyset, \emptyset, x)\rangle \\
&= \mathrm{Sim}(\{v\}, \{v \mapsto \mathsf{O}\}, \emptyset, \emptyset, v) \\
&= (\{v\}, \{v \mapsto \bullet\}, \emptyset, \emptyset, v)
\end{aligned}$$

Using the axioms that we have already proven sound, we may then derive that

$$\begin{aligned}
[\![\langle M \mid x = x, D\rangle]\!] &= [\![\langle M[x := y] \mid y = x, x = x, D[y := x]\rangle]\!] \\
&= [\![\langle M[x := y] \mid y = \langle x \mid x = x\rangle, D[y := x]\rangle]\!] \\
&= [\![\langle M[x := y] \mid y = \bullet, D[y := x]\rangle]\!] \\
&= [\![\langle M \mid x = \bullet, D\rangle]\!]
\end{aligned}$$

□

In our completeness proof we will use a complete rewrite system $\mathcal{R}^{\rightarrow}_{\text{graph}}$, whose normal forms are flat terms and whose conversion relation is exactly the same as the provable equality of $\mathcal{R}_{\text{graph}}$. We will also use an isomorphism ψ from higher-order term graphs to flat terms, whose inverse is $[\![.]\!]$. Because ψ is an isomorphism it gives a unique canonical representing term for every graph. Every other representation of the same graph is rewritten to this unique representation by $\mathcal{R}^{\rightarrow}_{\text{graph}}$. This implies that we can prove every representation of a graph equal to the canonical representation, which in turn implies that we can prove any two representations equal.

We will now design $\mathcal{R}^{\rightarrow}_{\text{graph}}$. The goal is to rewrite every term to a flat term. We start by orienting the axioms in $\mathcal{R}_{\text{graph}}$ from left to right, except the empty box axiom, which we orient from right to left. We need to orient this axiom from right to left because $\langle x \mid \rangle$ is a flat term and x is not. However, by orienting the axiom in this way we introduce non-termination. For example we have that:

$$x \to \langle x \mid \rangle \to \langle \langle x \mid \rangle \mid \rangle \to \cdots \quad .$$

In this sequence the first step is necessary to rewrite x into an equivalent flat term, but the further steps form an unwanted diverging sequence. The reason for this unwanted expansion is that we introduce a new box around an existing box. Intuitively this is unnecessary because we only need one box. Thus, we restrict the box introduction rule to

$$M \to \langle M \mid \rangle \text{ , if } M \not\equiv \langle M' \mid D \rangle \quad . \tag{5.1}$$

This rules out the diverging sequence in the example, but we can still get a cycle:

$$\langle x \mid x = A \rangle \to \langle x \mid x = \langle A \mid \rangle \rangle \xrightarrow[\text{merge}]{} \langle x \mid x = A \rangle \quad .$$

The reason for this cycle is that we can introduce a box where we don't need it in such a way that we can clean it up again with a merge step. The solution is to further restrict empty box introduction to places where we need them. That is to restrict box introduction steps to the roots of the term and the abstractions in the term. To write down this restriction we use the following abbreviations for contexts:

$$\begin{aligned} C_{\text{abs}} &= [x]\Box \\ C_{\text{fun}} &= F(M_1, \cdots, \Box, \cdots M_n) \\ C_{\text{ext}} &= \langle \Box \mid D \rangle \\ C_{\text{int}} &= \langle M \mid x = \Box, D \rangle \end{aligned}$$

With these abbreviations we can write the restricted rule as follows:

$$M \xrightarrow[\Box \mid C C_{\text{abs}}]{} \langle M \mid \rangle \text{ , if } M \not\equiv \langle M' \mid D \rangle \quad . \tag{5.2}$$

Another source of non-termination is the naming rule. Like the box introduction rule this rule can be applied to the wrong term:

$$x \to \langle z \mid z = x \rangle \to \langle w \mid w = \langle z \mid z = x \rangle \rangle \to \cdots \quad .$$

From this example it is clear that we do not need to apply naming to a box. We also do not need naming for variables:

$$\langle x \mid \rangle \to \langle \langle z \mid z = x \rangle \mid \rangle \to \langle z \mid z = x \rangle \langle x \mid \rangle \to \cdots \quad .$$

We do need naming for function symbols because we need to rewrite A to $\langle x \mid x = A \rangle$. The same holds for abstractions. Like for box introduction we can also apply the naming rule in the wrong place:

$$A \to \langle x \mid x = A \rangle \to \langle x \mid x = \langle y \mid y = A \rangle \rangle \to \langle x \mid x = y, y = A \rangle \langle y \mid y = A \rangle \to \cdots$$

in the second step we are giving the symbol A the name y, but A already had a name : x. In turn we then have to remove the double name, which is done in the fourth step. Intuitively that means that we need to apply naming to function application and abstractions that do not already have a name. We can be certain these subterms do not have a name if they occur at the top of a term or as the argument of a function or abstraction. Formally we have to take a slightly more relaxed notion, because we also need to be able to apply naming to terms such as

$$\langle A \mid \rangle \quad .$$

That is, we have to allow a few boxes to be added in between the top and the function symbol. The suitable rewrite rules for naming thus become:

$$\begin{array}{ll} \lambda x.M & \xrightarrow[(\Box \mid C(C_{\text{abs}} \mid C_{\text{fun}}))C^*_{\text{ext}}]{} \langle y \mid y = \lambda x.M \rangle \\ F(M_1, \cdots, M_n) & \xrightarrow[(\Box \mid C(C_{\text{abs}} \mid C_{\text{fun}}))C^*_{\text{ext}}]{} \langle x \mid x = F(M_1, \cdots, M_n) \rangle \end{array}$$

Note that these rules overlap with 5.2 for function application and abstractions. We remove this overlap by further restricting the box introduction rule to

$$x \xrightarrow[\Box \mid C C_{\text{abs}}]{} \langle x \mid \rangle$$

Definition 5.2.3 The representational rewriting system $\mathcal{R}^{\to}_{\text{graph}}$ is given by the following rewriting rules:

$$\begin{array}{lll} F(\cdots \langle M_i \mid D \rangle \cdots) & \xrightarrow[\text{lift}]{} & \langle F(\cdots M_i \cdots) \mid D \rangle \\ x & \xrightarrow[\Box \mid C C_{\text{abs}}]{\Box \text{I}} & \langle x \mid \rangle \\ \langle \langle M \mid D_1 \rangle \mid D_2 \rangle & \xrightarrow[\text{merge}]{} & \langle M \mid D_1, D_2 \rangle \\ \langle M \mid x = \langle N \mid D_1 \rangle, D_2 \rangle & \xrightarrow[\text{merge}]{} & \langle M \mid x = N, D_1, D_2 \rangle \\ \langle M \mid x = x, D \rangle & \xrightarrow[\bullet]{} & \langle M \mid x = \bullet, D \rangle \\ \langle M \mid x = y, D \rangle & \xrightarrow[\text{O}]{} & \langle M[x := y] \mid D[x := y] \rangle \quad x \not\equiv y \\ \lambda x.M & \xrightarrow[\text{name}]{(\Box \mid C(C_{\text{abs}} \mid C_{\text{fun}}))C^*_{\text{ext}}} & \langle y \mid y = \lambda x.M \rangle \\ F(M_1, \cdots, M_n) & \xrightarrow[(\Box \mid C(C_{\text{abs}} \mid C_{\text{fun}}))C^*_{\text{ext}}]{} & \langle x \mid x = F(M_1, \cdots, M_n) \rangle \end{array}$$

Remark 5.2.4 To obtain our goals of termination and confluence we put some restrictions on the rewrite rules to define $\mathcal{R}^{\rightarrow}_{\text{graph}}$. That goal leaves some room for choices. For example, instead of the rules

$$\begin{array}{ll} \lambda x.M & \xrightarrow[(\Box|C(C_{\text{abs}}|C_{\text{fun}}))C^*_{\text{ext}}]{} \langle y \mid y = \lambda x.M\rangle \\ F(M_1,\cdots,M_n) & \xrightarrow[(\Box|C(C_{\text{abs}}|C_{\text{fun}}))C^*_{\text{ext}}]{} \langle x \mid x = F(M_1,\cdots,M_n)\rangle \end{array}$$

we could have opted for

$$\begin{array}{ll} \lambda x.M & \xrightarrow[(\Box|C(C_{\text{abs}}|C_{\text{fun}}))]{} \langle y \mid y = \lambda x.M\rangle \\ F(M_1,\cdots,M_n) & \xrightarrow[(\Box|C(C_{\Box}|C_F))]{} \langle x \mid x = F(M_1,\cdots,M_n)\rangle \end{array}$$

This alternative choice leads to a small problem: with these rules we cannot rewrite $\langle A \mid\rangle$ to a flat term. Also, we would have had a critical pair that does not converge:

$$\begin{array}{ccc} A & \longrightarrow & \langle A \mid\rangle \\ \downarrow & & \\ \langle x \mid x = A\rangle & & \end{array}$$

To fix these problem we would have had to add two more rules:

$$\begin{array}{ll} \langle \lambda x.M \mid D\rangle & \xrightarrow[(\Box|C(C_{\text{abs}}|C_{\text{fun}}))]{} \langle y \mid y = \lambda x.M, D\rangle \\ \langle F(M_1,\cdots,M_n) \mid D\rangle & \xrightarrow[(\Box|C(C_{\text{abs}}|C_{\text{fun}}))]{} \langle x \mid x = F(M_1,\cdots,M_n), D\rangle \end{array}$$

We have chosen for the slightly more complicated rules to keep the total number of rules smaller.

When we derived the rewrite system from the axioms, we introduced several restrictions and new rules. Nevertheless, the provable equality of the axiom system and the convertibility of the rewrite system are the same:

Proposition 5.2.5 Given two terms M and N. We have that

$$\mathcal{R}_{\text{graph}} \vdash M = N \iff M \xleftrightarrow{\mathcal{R}^{\rightarrow}_{\text{graph}}} N \ .$$

Proof.

"$\Leftarrow$" Because all our rewrite rules are restrictions of axioms, this is trivial.

"$\Rightarrow$" All axioms are rewrite rules except box introduction and naming, thus we only need to show that the following conversions exist for any context C and any term M:

$$C[M] \quad \leftrightarrow \quad C[\langle M \mid\rangle] \tag{5.3}$$

$$C[M] \quad \leftrightarrow \quad C[\langle x \mid x = M\rangle] \tag{5.4}$$

We will show 5.3 by induction on the context C:

$C = \Box \mid C' C_{\text{abs}}$ We now must distinguish cases for the term M:

$M = x$

$$C[M] \xrightarrow[\Box I']{} C[\langle M \mid \rangle]$$

$M = F(N_1, \cdots, N_n)$ or $M = [x]N$

$$\begin{array}{rll} C[M] & \xrightarrow[\text{name}]{} & C[\langle x \mid x = M \rangle] \\ & \xleftarrow[\text{merge}]{} & C[\langle \langle x \mid x = M \rangle \mid \rangle] \\ & \xleftarrow[\text{name}]{} & C[\langle M \mid \rangle] \end{array}$$

$M = \langle N \mid D \rangle$

$$C[\langle M \mid \rangle] \xrightarrow[\text{merge}]{} C[M]$$

$C = C' C_{\text{fun}}$

$$\begin{array}{rll} C[M] & \equiv & C'[F(\cdots M \cdots)] \\ & \xleftrightarrow[\text{IH}]{} & C'[\langle F(\cdots M \cdots) \mid \rangle] \\ & \xleftarrow[\text{lift}]{} & C'[F(\cdots \langle M \mid \rangle \cdots)] \end{array}$$

$C = C' C_{\text{ext}}$ or $C = C' C_{\text{int}}$

$$C[M] \xleftarrow[\text{merge}]{} C[\langle M \mid \rangle]$$

To prove 5.4 we will use the following claim:

$$C[\langle M \mid D \rangle] \leftrightarrow C[\langle x \mid x = M, D \rangle] \tag{5.5}$$

Using this claim we have

$$C[M] \xleftrightarrow[5.3]{} C[\langle M \mid \rangle] \xleftrightarrow[5.5]{} C\langle x \mid x = M \rangle] \ .$$

We can prove the claim 5.5 by induction on M:

$M \equiv y$

$$C[\langle y \mid D \rangle] \xleftarrow[\text{vs}]{} C[\langle x \mid x = y, D \rangle] \ .$$

$M \equiv \langle M' \mid D' \rangle$

$$\begin{array}{rlll} C[\langle \langle M' \mid D' \rangle \mid D \rangle] & \xrightarrow[\text{merge}]{} & C[\langle M' \mid D', D \rangle] & \\ & \leftrightarrow & C[\langle x \mid x = M', D', D \rangle] & \text{IH} \\ & \xleftarrow[\text{merge}]{} & C[\langle x \mid x = \langle M' \mid D' \rangle, D \rangle] & \end{array}$$

$M \equiv [x]M'$ or $M \equiv F(M_1, \cdots, M_n)$ To prove the conversion for this case we use induction on C.

$C = \Box \mid C'(C_{\text{abs}} \mid C_{\text{fun}]})$

$$\begin{array}{rll} C[\langle M \mid D \rangle] & \xrightarrow[\text{name}]{} & C[\langle \langle x \mid x = M \rangle \mid D \rangle] \\ & \xrightarrow[\text{merge}]{} & C[\langle x \mid x = M, D \rangle] \end{array}$$

$C = C'\,C_{\text{ext}}]$

$$\begin{array}{lll} C'[\langle\langle M \mid D\rangle \mid D'\rangle] & \xrightarrow[\text{merge}]{} & C'[\langle M \mid D, D'\rangle] \\ & \leftrightarrow & C'[\langle x \mid x = M, D, D'\rangle] \qquad \text{IH} \\ & \xleftarrow[\text{merge}]{} & C'[\langle\langle x \mid x = M, D\rangle \mid D'\rangle] \end{array}$$

$C = C'\,C_{\text{int}}$

$$\begin{array}{ll} C'[\langle M' \mid y = \langle M \mid D\rangle, D'\rangle] & \\ \xrightarrow[\text{merge}]{} & C'[\langle M' \mid y = M, D, D'\rangle] \\ \xleftarrow[\text{vs}]{} & C'[\langle M' \mid y = x, x = M, D, D'\rangle] \\ \xleftarrow[\text{merge}]{} & C'[\langle M' \mid y = \langle x \mid x = M, D\rangle, D'\rangle] \end{array}$$

□

We will prove termination of $\mathcal{R}^{\rightarrow}_{\text{graph}}$, by defining a measure on terms and showing that every step decreases the measure. This measure will be recursively defined, with cases for variables, function applications, abstractions and letrecs. The case for variables will depend on a parameter T and the cases for function applications and abstractions will depend on a parameter S. Thus, the general form of the measure is $|.|^T_S$. We have to use these parameters because the cost of these cases depends on the context. That is, if a variable occurs directly below an abstraction then we have to apply the box introduction rule to it, the cost of which is $\{\{1\}\}$, if we have an equation such as $x = x$ then we have to introduce a black hole, the cost of which is $\{\{0\}\}$, and if a variable occurs as the argument of a function we do not have to do anything at all, the cost of which is $\emptyset$. For function applications and abstractions we might have to apply the naming rule (cost $\{\{1\}\}$) or not (cost $\emptyset$). The formal definition is:

Definition 5.2.6 Define

$$|M| = |M|^{\{\{1\}\}}_{\{\{1\}\}} \ ,$$

where

$$\begin{array}{lcl} |x|^T_S & = & T \\ |F(M_1,\cdots,M_n)|^T_S & = & S + inc(\Sigma^n_{i=1}|M_i|^{\emptyset}_{\{\{1\}\}}) \\ |[x]M|^T_S & = & S + inc(|M|^{\{\{1\}\}}_{\{\{1\}\}}) \\ |\langle M_0 \mid D\rangle|^T_S & = & \{\{0\}\} + |M_0|^{\{\{0\}\}}_S + |D| \\ |x_1 = M_1,\cdots,x_n = M_n| & = & \Sigma^n_{i=1}(\{\{0\}\} + |M_i|^{\{\{0\}\}}_{\emptyset}) \\ inc(\{\{m_1,\cdots,m_n\}\}) & = & \{\{m_1+1,\cdots,m_n+1\}\} \end{array}$$

In the termination proof we will have to prove many inequalities of the form $|C[M]| < |C[N]|$. The next lemma, tells us under which conditions $|M| < |N|$ implies that $|C[M]| < |C[N]|$:

Lemma 5.2.7 Given two cyclic terms M and N, we have that

$$\left(\forall S,T : |M|^T_S < |N|^T_S\right) \implies \left(\forall C,S,T : |C[M]|^T_S < |C[N]|^T_S\right) \ .$$

Proof. By structural induction on C:

$C \equiv \Box$ Trivial.

$C \equiv F(\cdots C' \cdots)$ For some multiset U and any term P we have that

$$|F(\cdots P \cdots)|_S^T = U + |P|_{\{\{1\}\}}^{\emptyset} \ .$$

By induction hypothesis we have that

$$|C'[M]|_{\{\{1\}\}}^{\emptyset} < |C'[N]|_{\{\{1\}\}}^{\emptyset} \ .$$

Hence:

$$|C[M]|_S^T = U + inc(|C'[M]|_{\{\{1\}\}}^{\emptyset}) < U + inc(|C'[N]|_{\{\{1\}\}}^{\emptyset}) = |C[N]|_S^T \ .$$

$C \equiv [x]C'$ For some U and by induction hypothesis we have:

$$|C[M]|_S^T = U + inc(|C'[M]|_{\{\{1\}\}}^{\{\{1\}\}}) < U + inc(|C'[N]|_{\{\{1\}\}}^{\{\{1\}\}}) = |C[N]|_S^T \ .$$

$C \equiv \langle C' \mid D \rangle$ For some U and by induction hypothesis we have:

$$|C[M]|_S^T = U + |C'[M]|_S^{\{\{0\}\}} < U + |C'[N]|_S^{\{\{0\}\}} = |C[N]|_S^T \ .$$

$C \equiv \langle M \mid x = C', D \rangle$ For some U and by induction hypothesis we have:

$$|C[M]|_S^T = U + |C'[M]|_{\emptyset}^{\{\{0\}\}} < U + |C'[N]|_{\emptyset}^{\{\{0\}\}} = |C[N]|_S^T \ .$$

□

The next lemma is the actual proof of termination of $\mathcal{R}_{\text{graph}}^{\rightarrow}$.

Lemma 5.2.8 $\mathcal{R}_{\text{graph}}^{\rightarrow}$ is terminating.

Proof. To show that $\mathcal{R}_{\text{graph}}^{\rightarrow}$ is terminating, we will show that every rule decreases the measure defined in Definition 5.2.6:

□I We have that

$$|x| = \{\{1\}\} > \{\{0,0\}\} = |\langle x \,|\rangle| \ . \tag{5.6}$$

For any S, T we also have that

$$|[x]y|_S^T = S + \{\{2\}\} > S + \{\{1,1\}\} = |[x]\langle y \,|\rangle|_S^T \ . \tag{5.7}$$

By Lemma 5.2.7 we conclude from 5.7 that

$$|C[[x]y]| > |C[[x]\langle y \,|\rangle]| \ . \tag{5.8}$$

From 5.6 and 5.8 we conclude that

$$M \xrightarrow[\Box I]{} N \implies |M| > |N| \ .$$

Naming Given

$$C[M] \xrightarrow[\text{name}]{} C[\langle x \mid x = M\rangle] \ ,$$

we know that

$$M \equiv F(N_1, \cdots, N_n) \text{ or } M \equiv [x]N \ .$$

From this we can infer that for any T and for some U we have that

$$|M|^T_{\{\{1\}\}} = \{\{1\}\} + U > \{\{0,0,0\}\} + U = |\langle x \mid x = M\rangle|^T_{\{\{1\}\}} \ .$$

We must now distinguish cases for C:

$C = \Box$ We now immediately have that

$$|C[M]| > |C[\langle x \mid x = M\rangle]| \ .$$

$C = C'(C_{\text{abs}} \mid C_{\text{fun}})C^n_{\text{ext}}$ By induction on n we can prove that

$$|C[M]| > |C[\langle x \mid x = M\rangle]| \ .$$

Lift

$$\begin{array}{lcl} |F(\langle M \mid D\rangle, N)|^T_S & = & S + inc(\{\{0\}\} + |M|^{\emptyset}_{\{\{1\}\}} + \{\{1\}\} + |D| + |N|^{\emptyset}_{\{\{1\}\}}) \\ |\langle F(M,N) \mid D\rangle|^T_S & = & \{\{0\}\} + S + inc(|M|^{\emptyset}_{\{\{1\}\}} + |N|^{\emptyset}_{\{\{1\}\}}) + |D| \end{array}$$

external merge

$$\begin{array}{lcl} |\langle\langle M \mid D_1\rangle \mid D_2\rangle|^T_S & = & \{\{0,0\}\} + |M|^{\{\{0\}\}}_S + |D_1| + |D_2| \\ |\langle M \mid D_1, D_2\rangle|^T_S & = & \{\{0\}\} + |M|^{\{\{0\}\}}_S + |D_1| + |D_2| \end{array}$$

internal merge

$$\begin{array}{lcl} |\langle M \mid x = \langle N \mid D_1\rangle, D_2|^T_S & = & \{\{0,0\}\} + |M|^{\{\{0\}\}}_S |N|^{\{\{0\}\}}_{\emptyset} + |D_1| + |D_2| \\ |\langle M \mid x = N, D_1, D_2|^T_S & = & \{\{0\}\} + |M|^{\{\{0\}\}}_S |N|^{\{\{0\}\}}_{\emptyset} + |D_1| + |D_2| \end{array}$$

○

$$\begin{array}{lcl} |\langle M \mid x = y, D\rangle|^T_S & = & \{\{0\}\} + |M|^{\{\{0\}\}}_S + \{\{0,0\}\} + |D| \\ |\langle M[x := y] \mid D[x := y]\rangle & = & \{\{0\}\} + |M|^{\{\{0\}\}}_S + |D| \end{array}$$

•

$$\begin{array}{lcl} |\langle M \mid x = x, D\rangle|^T_S & = & \{\{0\}\} + |M|^{\{\{0\}\}}_S + \{\{0,0\}\} + |D| \\ |\langle M \mid x = \bullet, D\rangle & = & \{\{0\}\} + |M|^{\{\{0\}\}}_S + \{\{0\}\} + |D| \end{array}$$

□

Proposition 5.2.9 $\mathcal{R}^{\rightarrow}_{\text{graph}}$ is confluent and terminating.

Proof. We know that $\mathcal{R}^{\rightarrow}_{\text{graph}}$ is terminating (Lemma 5.2.8). Therefore, confluence follows immediately from local confluence (Newman's lemma). The proof of local confluence is a long case analysis, which is left to the reader. □

We will now continue the completeness proof with the definition of the isomorphism ψ from higher-order term graphs to flat terms, whose inverse is $[\![.]\!]$. The mapping ψ is based on the scheme of translating every node to an equation and placing these equations in such a way that every equation gets placed in a subterm of an abstraction corresponding to node v if and only if the equation came from a node in the interior of the scope of v.

Definition 5.2.10 Given a graph $g \equiv (V, L, A, S, r)$. To simplify matters somewhat we assume that $V \subset \mathsf{Var}$, such that V is disjoint from the set of free variables. Let x_v be a variable for every abstraction node $v \in V$, such that all x_v are pairwise distinct from each other, from variables in V and from the free variables in the graph.

For every node v we define a term M_v by

$$M_v = \begin{cases} F(A(v,1), \cdots, A(v,n)) & L(v) = F \\ [x_v]\langle A(v) \mid D_{S^-(v)} \rangle & L(v) = \square \end{cases}$$

For every set of nodes S we define

$$D_S = \ x_1 = M_{x_1}, \cdots, x_n = M_{x_n} \ ,$$

where $\{x_1, \cdots, x_n\} = \{x \in S \mid x \notin S^-(y), y \in S\}$. Finally we define

$$\psi(g) = \langle r \mid D_V \rangle \ .$$

Lemma 5.2.11 Given a graph g and a flat term M. We have that

- $[\![\psi(g)]\!] = g$
- $\psi([\![M]\!]) = M$

Proof. Trivial. □

This means that for flat terms ψ is the inverse function of the graph semantics and that every graph has a representation. We will refer to $\psi(g)$ as the canonical representation of g.

Theorem 5.2.12 The proof system $\mathcal{R}_{\text{graph}}$ is complete with respect to isomorphism.

Proof. Given two terms M, N with the same graphs, we have that $\mathcal{R}^{\rightarrow}_{\text{graph}}(M)$ and $\mathcal{R}^{\rightarrow}_{\text{graph}}(N)$ also have the same graphs. Because the latter are flat terms we have that

$$\mathcal{R}^{\rightarrow}_{\text{graph}}(M) \equiv \psi([\![\mathcal{R}^{\rightarrow}_{\text{graph}}(M)]\!]) \equiv \psi([\![\mathcal{R}^{\rightarrow}_{\text{graph}}(N)]\!]) \equiv \mathcal{R}^{\rightarrow}_{\text{graph}}(N) \ .$$

By Prop. 5.2.5 we have that

$$\vdash_{\mathcal{R}_{\text{graph}}} M = \mathcal{R}^{\rightarrow}_{\text{graph}}(M) \text{and} \vdash_{\mathcal{R}_{\text{graph}}} N = \mathcal{R}^{\rightarrow}_{\text{graph}}(N) \ .$$

We can conclude that

$$\vdash_{\mathcal{R}_{\text{graph}}} M = N \ .$$

□

5.3 Garbage Collection

So far we have considered graphs where there could be nodes that are not reachable from the root of the graph. These inaccessible nodes are called garbage nodes. For many applications the garbage doesn't matter. That is, graphs that only differ as far as the garbage nodes are concerned are to be considered equivalent. Thus, in this section we will axiomatize the equivalence relation of terms whose graphs are equivalent up to garbage nodes. Also, for many applications it is important to consider garbage free graphs. That is, to consider graphs that do not have garbage nodes. Thus, we will also provide a confluent and terminating rewrite system that computes a canonical garbage free equivalent of a given cyclic term.

Definition 5.3.1 Given a graph $g \equiv (V, L, A, S, r)$. The garbage free version of g, denoted $\mathrm{gf}(g)$ is the graph $(W, L\upharpoonright_W, A\upharpoonright_W, S', r)$, where where W is the set of nodes accessible from r and $S' : W \to \mathcal{P}(W)$ is given by $S'(v) = S(v) \cap W$. A graph g is garbage free if $g \equiv \mathrm{gf}(g)$.

Given two graphs g_1, g_2. If $\mathrm{gf}(g_1) = \mathrm{gf}(g_2)$ then g_1 and g_2 are equivalent modulo garbage collection, denoted $g_1 \sim_{\mathrm{gc}} g_2$.

We want to find a proof system that is sound and complete for $\sim_{\mathrm{gc}}$. Because two terms with the same graph also have the same garbage free graph, it is obvious to try to find such a proof system by extending $\mathcal{R}_{\mathrm{graph}}$. Thus, we add the garbage collection (gc) axiom:

$$\langle M \mid D \rangle = M \ ,$$

where D does not bind a free variable of M. As the corresponding rewrite rule we have

$$\langle M_0 \mid x_1 = M_1, \cdots, x_n = M_n \rangle \to \langle M_0 \mid x_1 = M_1, \cdots, x_k = M_k \rangle \ ,$$

where $k < n$ and x_i does not occur free in M_j for $0 \leq j \leq k < i \leq n$. Thus, we get the proof system $\mathcal{R}_{\mathrm{gc}}$ and the rewrite system $\mathcal{R}_{\mathrm{gc}}^{\to}$:

Definition 5.3.2 The proof system $\mathcal{R}_{\mathrm{gc}}$ is the extension of $\mathcal{R}_{\mathrm{graph}}$ with the axiom

$$\langle M \mid D \rangle = M \ .$$

The rewrite system $\mathcal{R}_{\mathrm{gc}}^{\to}$ is the extension of $\mathcal{R}_{\mathrm{graph}}^{\to}$ with the rule

$$\langle M_0 \mid x_1 = M_1, \cdots, x_n = M_n \rangle \to \langle M_0 \mid x_1 = M_1, \cdots, x_k = M_k \rangle \ ,$$

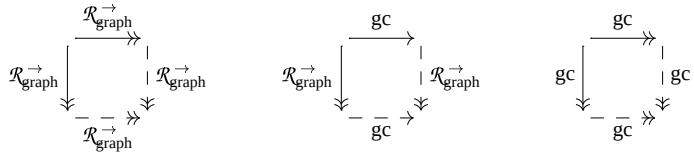
where $k < n$ and x_i does not occur free in M_j for $0 \leq j \leq k < i \leq n$.

Before we prove that the proof system is sound and complete, we will first prove some confluence and termination of the rewrite system.

Lemma 5.3.3 The rewrite system $\mathcal{R}_{\mathrm{gc}}^{\to}$ is confluent and terminating.

Figure 5.2: pictorial description of the garbage collection axiom

Proof. Confluence follows form the following three diagrams:



The first diagram is proven in Prop. 5.2.9. The proofs of the other two diagrams are simple case analyses, which are left to the reader.

To prove termination we will show that $\mathcal{R}^{\rightarrow}_{\text{gc}}$ decreases the measure defined in 5.2.6. For $\mathcal{R}^{\rightarrow}_{\text{graph}}$ this has been shown in the proof of Lemma 5.2.8, so we only need to show that an application of the garbage collection rule decreases the measure. By Lemma 5.2.7 we only need to prove that the measure of the left-hand side of the rule is larger than that of the right-hand side. If we compute these measures we get:

$$\begin{array}{lcl} |\langle M \mid D_1, D_2\rangle|^T_S & = & |M|^{\{\{0\}\}}_S + |D_1| + |D_2| \\ |\langle M \mid D_1\rangle|^T_S & = & |M|^{\{\{0\}\}}_S + |D_1| \end{array}$$

Because D_2 is never empty we have that the measure of the left-hand side is larger that that of the right-hand side. □

Theorem 5.3.4 The proof system $\mathcal{R}_{\text{gc}}$ is a sound and complete axiomatization of $\sim_{\text{gc}}$.

Proof. To prove soundness we need to prove soundness for $\mathcal{R}_{\text{graph}}$ and for the garbage collection axiom. The soundness of $\mathcal{R}_{\text{graph}}$ is obvious because $\mathcal{R}_{\text{graph}}$ is sound for graph isomorphism and because isomorphic graphs have isomorphic garbage free versions. The soundness of the garbage collection axiom follows from the facts that

$$[\![\langle M \mid D\rangle]\!] \sim_{\text{gc}} [\![M]\!] \ ,$$

if D does not bind a free variable in M and that for any context C and terms M, N we have that

$$[\![M]\!] \sim_{\text{gc}} [\![N]\!] \implies [\![C[M]]\!] \sim_{\text{gc}} [\![C[N]]\!] \ .$$

Soundness follows from the facts that a graph g is garbage free if and only if $\psi(g)$ is a $\mathcal{R}^{\rightarrow}_{\text{graph}}$ normal form and that

$$\mathcal{R}_{\text{gc}} \vdash M = N \iff M \xleftrightarrow[\mathcal{R}^{\rightarrow}_{\text{gc}}]{} N \ .$$

□

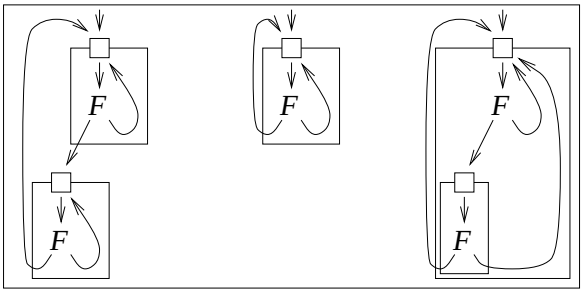


Figure 5.3: Bisimulation of higher-order term graphs

5.4 Bisimilarity

In this section we will define bisimulation on higher-order terms graphs and provide a proof system that is sound and complete with respect to bisimulation.

In the introduction we have already introduced bisimulation for first-order term graphs. We can define this notion of bisimulation as follows: Two graphs are bisimilar if they are both the same variable or if they both have nodes as roots and there exists a *bisimulation* between the two graphs that include the two roots. A bisimulation between two graphs is a relation on the nodes of those graphs, such that pairs of nodes in the relation have the same label and such that for every matching pair of arguments we have that those arguments are the same variable or a pair of nodes that is in the bisimulation.

To extend this definition to higher-order graphs we must deal with scopes and back-pointers. The obvious extension for back-pointers is that a matching pair of arguments can also be back-pointers to nodes in the bisimulation. Without further modification we would have that all three graphs in Fig. 5.3 are bisimilar. However, the last two graphs in the figure do not have the same unwinding, so we do not want them to be bisimilar. We can solve this problem by requiring that scopes are bisimilar too. More precisely, for any pair of abstraction nodes in the bisimulation and any other pair of nodes in the bisimulation we have that if a node of the second pair is in the interior of the scope of a node in the first pair then the other node in the second pair is in the interior of the scope of the other node in the first pair. With this requirement the middle and right graphs in Fig. 5.3 are no longer bisimilar.

In our formal definition we make one more extension, which we need in proofs: instead of considering only identical variables to be bisimilar we parameterize our notion of bisimulation with a relation that indicates the bisimilar variable.

Definition 5.4.1 Given higher-order graphs g_1, g_2 and a relation R_V on variables. The graphs g_1 and g_2 are bisimilar up to R_V, denoted $g_1 \underline{\leftrightarrow}_{R_V} g_2$, if there exists a relation $R \subset V_1 \times V_2$ such that $r_1 \,\dot{R}\, r_2$ and such that for all $(v_1, v_2) \in R$ we have that

- $L_1(v_1) = L_2(v_2)$
- $A_1(v_1) \,\ddot{R}\, A_2(v_2)$

- $\forall v_1' \in S_1^-(v_1), v_1' \; R \; v_2' : v_2' \in S_2^-(v_2)$
- $\forall v_2' \in S_2^-(v_2), v_1' \; R \; v_2' : v_1' \in S_1^-(v_1)$

where $a_1 \cdots a_n \; \ddot{R}_S \; b_1 \cdots b_n$ if $a_i \; \dot{R}_S \; b_i$ and where $\dot{R}$ is given by

- $x \; \dot{R} \; y$ if $x \; R_V \; y$
- $v_1 \; \dot{R} \; v_2$ if $v_1 \; R \; v_2$
- $\overline{v_1} \; \dot{R} \; \overline{v_2}$ if $v_1 \; R \; v_2$

The graphs g_1 and g_2 are bisimilar, denoted $g_1 \; \underline{\leftrightarrow} \; g_2$, if $g_1 \; \underline{\leftrightarrow}_{\equiv} \; g_2$.

Remark 5.4.2 In addition we could define weak bisimulation by changing

- $\forall v_1' \in S_1^-(v_1), v_1' \; R \; v_2' : v_2' \in S_2^-(v_2)$
- $\forall v_2' \in S_2^-(v_2), v_1' \; R \; v_2' : v_1' \in S_1^-(v_1)$

into

- $\forall v_1' \in S_1^-(v_1), (\exists i : A(v_1', i) = \overline{v_1}), v_1' \; R \; v_2' : v_2' \in S_2^-(v_2)$
- $\forall v_2' \in S_2^-(v_2), (\exists i : A(v_2', i) = \overline{v_2}), v_1' \; R \; v_2' : v_1' \in S_1^-(v_1)$

It is an open question if this yields a notion of bisimulation, such that terms are bisimilar if and only if their labeled unwindings are the same.

In order to give a complete axiomatization of bisimulation we will need a new axiom. The new axiom will be that two terms are equal if they are bisimilar. At first sight it seems that this makes everything trivial, but bisimulation on graphs and bisimulation on terms are not the same notion. Bisimulation on terms is defined below. The definition has to deal with a problem. In the definition for graphs we had a natural separation between recursion variables and nodes. In terms we only have variables. We will have to be very careful to decide which variables are allowed to be equated to which other variables. To do this we parameterize the definition of bisimulation with a relation S, which tells us which variables we are allowed to identify. With this in mind we have the following intuitive clauses:

- Two variables are bisimilar if we are allowed to identify them.
- Two function applications are bisimilar if they use the same function symbol and their arguments are bisimilar.
- Two abstraction are bisimilar if their arguments are bisimilar, considering two variables identical if:
    - they are the two abstraction variables

- they were previously considered identical and they are not the abstraction variables

- Two letrecs are bisimilar if there exists a relation on the recursion variables, such that terms bound by related recursion variables are bisimilar and such that the two external parts are bisimilar. Variables are considered identical if they are related recursion variables or if they are previously related variables that are not recursion variables.

Definition 5.4.3 Given terms M and N. We define $M \underline{\leftrightarrow}_S N$ by structural induction as:

$x \underline{\leftrightarrow}_S y$
if $x\, S\, y$

$F(M_1,\cdots,M_n) \underline{\leftrightarrow}_S F(N_1,\cdots,N_n)$
if $M_i \underline{\leftrightarrow}_S N_i$, for $1 \leq i \leq n$

$[x]M \underline{\leftrightarrow}_S [y]N$
if $M \underline{\leftrightarrow}_{\{(x',y')\in S \mid x'\not\equiv x, y'\not\equiv y\}\cup\{(x,y)\}} N$

$\langle M_0 \mid x_1 = M_1,\cdots,x_m = M_m\rangle \underline{\leftrightarrow}_S \langle N_0 \mid y_1 = N_1,\cdots,y_n = N_n\rangle$
if $\exists S' \subset \{x_1,\cdots,x_m\}\times\{y_1,\cdots,y_n\}$ such that

$$M_0 \underline{\leftrightarrow}_{S''} N_0 \text{ and } \forall (x_i,y_j)\in S' : M_i \underline{\leftrightarrow}_{S''} N_j \ ,$$

where $S'' = S' \cup \{(x,y)\in S \mid x \not\equiv x_i, y \not\equiv y_j\}$.

We have that $M \underline{\leftrightarrow} N$ if $M \underline{\leftrightarrow}_{\mathrm{id}_{\mathsf{Var}}} N$.

Note that in general we do not have that $M \underline{\leftrightarrow} N$ if $[\![M]\!] \underline{\leftrightarrow} [\![N]\!]$. For example, the graph semantics of $\langle x \mid x = F(x)\rangle$ and $\langle x \mid x = F(F(x))\rangle$ are bisimilar but the terms are not. However, the graph semantics of the latter term is equivalent to that of $\langle x \mid x = F(y), y = F(x)\rangle$. This term is flat and hence bisimilar to the first term. Also note that the garbage collection axiom is a special case of the bisimilarity axiom. That is, we have that

$$\langle M \mid D\rangle \underline{\leftrightarrow} M \ ,$$

if D does not bind a free variable in M. Thus, we get the following axiom system:

Definition 5.4.4 The axiom system $\mathcal{R}_{\mathrm{bisim}}$ is the axiom system $\mathcal{R}_{\mathrm{graph}}$ extended with the axiom

$$M = N, \text{ if } M \underline{\leftrightarrow}_{\emptyset} N \ .$$

We will now prove that $\mathcal{R}_{\mathrm{bisim}}$ is a sound and complete axiomatization of bisimulation. To do this, we will first prove two lemmas.

The first lemma states that the bisimulation axiom is sound. That is, if two terms are bisimilar then their graph semantics are bisimilar.

Lemma 5.4.5 If $M \underline{\leftrightarrow}_S N$ then $[\![M]\!] \underline{\leftrightarrow}_S [\![N]\!]$.

Proof. By structural induction over M and N.

$x \underline{\leftrightarrow}_S y$
: It is easily checked that $[\![x]\!] \emptyset_S [\![y]\!]$.

$F(M_1, \cdots, M_n) \underline{\leftrightarrow}_S F(N_1, \cdots, N_n)$
: We have for $1 \leq i \leq n$ that $M_i \underline{\leftrightarrow}_S N_i$. By induction hypothesis we have that $[\![M_i]\!] R^i_S [\![N_i]\!]$. Let $g_1 = [\![F(M_1, \cdots, M_n)]\!]$ and $g_2 = [\![F(N_1, \cdots, N_n)]\!]$. If v_1 and v_2 are the roots of g_1 and g_2, respectively, then we can show that $g_1 \; R_S \; g_2$, for $R = \{(v_1, v_2)\} \cup R^1 \cup \cdots \cup R^n$.

$[x]M \underline{\leftrightarrow}_S [y]N$
: For $S' = \{(x', y') \in S \mid x' \not\equiv x, y' \not\equiv y\} \cup \{(x, y)\}$ we have that $M \underline{\leftrightarrow}_{S'} N$. By induction hypothesis we have that $[\![M]\!] \underline{\leftrightarrow}_{S'} [\![N]\!]$. Thus, there exists a relation R such that $[\![M]\!] \; R_{S'} \; [\![N]\!]$. Let $g_1 = [\![[x]M]\!]$ and $g_2 = [\![[y]N]\!]$. If v_1 and v_2 are the roots of g_1 and g_2, respectively, then we can show that $g_1 \; R'_S \; g_2$, for $R' = \{(v_1, v_2)\} \cup R$. Hence we have that $[\![[x]M]\!] \underline{\leftrightarrow}_S [\![N]\!]$.

$\langle M_0 \mid x_1 = M_1, \cdots, x_m = M_m \rangle \underline{\leftrightarrow}_S \langle N_0 \mid y_1 = N_1, \cdots, y_n = N_n \rangle$
: By definition we have that $\exists S' \subset \{x_1, \cdots, x_m\} \times \{y_1, \cdots, y_n\}$ such that for $S'' = S' \cup \{(x, y) \in S \mid x \not\equiv x_i, y \not\equiv y_j\}$ we have that $M_0 \underline{\leftrightarrow}_{S''} N_0$ and $\forall (x_i, y_j) \in S' : M_i \underline{\leftrightarrow}_{S''} N_j$. By induction hypothesis it then follows that $[\![M_0]\!] \underline{\leftrightarrow}_{S''} [\![N_0]\!], \forall (x_i, y_j) \in S' : [\![M_i]\!] \underline{\leftrightarrow}_{S''} [\![N_j]\!]$. By definition this means that there exist R^{ij} such that $[\![M_0]\!] R^{00}_{S''} [\![N_0]\!]$ and $\forall (x_i, y_j) \in S' : [\![M_i]\!] R^{ij}_{S''} [\![N_j]\!]$.

The nodes of $g_1 = [\![\langle M_0 \mid x_1 = M_1, \cdots, x_m = M_m \rangle]\!]$ are the nodes of the graphs $[\![M_0]\!], \cdots, [\![M_m]\!]$ plus some black holes $v_1, \cdots, v_k$. The nodes of $g_2 = [\![\langle N_0 \mid y_1 = N_1, \cdots, y_n = N_n \rangle]\!]$ are the nodes of the graphs $[\![N_0]\!], \cdots, [\![N_n]\!]$ and some black holes $w_1, \cdots, w_l$.

Let $B = \{v_1, \cdots, v_k\} \times \{w_1, \cdots, w_l\}$ and let

$$R = R^{00} \cup B \cup \cup_{(x_i, y_j) \in S'} R^{ij} \quad .$$

We claim that $g_1 R_S g_2$. Thus, we have that

$$[\![\langle M_0 \mid x_1 = M_1, \cdots, x_m = M_m \rangle]\!] \underline{\leftrightarrow}_S [\![\langle N_0 \mid y_1 = N_1, \cdots, y_n = N_n \rangle]\!] \quad .$$

□

Because of their special structure we have that two flat terms are bisimilar if and only if their graph semantics are bisimilar. One of the directions was already proven above. The other direction follows below.

Lemma 5.4.6 If $g_1 \underline{\leftrightarrow} g_2$ then $\psi(g_1) \underline{\leftrightarrow} \psi(g_2)$

Proof. Given a bisimulation R for g_1 and g_2, we must show that $\psi(g_1)\underline{\leftrightarrow}\psi(g_2)$. The most important thing to do is to define the S' given the task of showing that

$$\langle M_0 \mid x_1 = M_1, \cdots, x_m = M_m \rangle \underline{\leftrightarrow}_S \langle N_0 \mid y_1 = N_1, \cdots, y_n = N_n \rangle \ .$$

A simple induction proof shows that the following definition for S' works:

$$S' = \{(x,y) \in R \mid x \in \{x_1, \cdots, x_m\}, y \in \{y_1, \cdots, y_n\}\} \ .$$

□

Theorem 5.4.7 The proof system $\mathcal{R}_{\text{bisim}}$ is sound and complete with respect to bisimulation.

Proof.

Soundness Follows from Lemma 5.4.5, Thm. 5.2.2 and the fact that every graph is bisimilar to itself.

Completeness Given bisimilar graphs g_1, g_2, any representation M of g_1 and any representation N of g_2. By Thm. 5.2.12 we have that $\mathcal{R}_{\text{graph}} \vdash M = \psi g_1$ and $\mathcal{R}_{\text{graph}} \vdash N = \psi g_2$. By Lemma 5.4.6 we have that $\psi g_1 \underline{\leftrightarrow} \psi g_2$. Thus, we have that $\mathcal{R}_{\text{bisim}} \vdash M = N$.

□

For first-order term graphs we have that two graphs are bisimilar if and only if they have the same unwinding. That means that for first order terms with letrec $\mathcal{R}_{\text{bisim}}$ is not only a sound and complete axiomatization of bisimilarity, but also a sound and complete axiomatization of unwinding equivalence. For higher-order term graphs bisimilarity is equivalent with having the same scoped unwinding. The term that have the same (labeled) unwinding are not necessarily bisimilar. We will now start working towards a sound and complete axiomatization for the unwinding equivalence on higher-order term graphs.

5.5 Scope Equivalence

If the unwindings of two term graphs are the same, but their unwindings are not, then the scoped unwindings of those two graphs differ only by their scopes. Thus, it seems natural to start by trying to find an axiomatization of graphs that differ only by their scopes.

Definition 5.5.1 Given two graphs g_1, g_2. If $g_1 \sim (V, L, A, S_1, r)$ and $g_2 \sim (V, L, A, S_2, r)$ then g_1 and g_2 are equivalent modulo scopes, denoted $g_1 \sim_{\text{scope}} g_2$. If $\text{gf}(g_1) \sim_{\text{scope}} \text{gf}(g_2)$ then g_1 and g_2 are equivalent modulo garbage and scope, denoted $\sim_{\text{gs}}$.

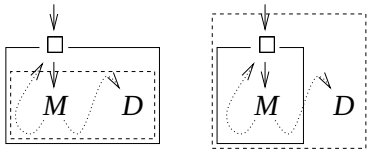


Figure 5.4: Pictorial description of the abstraction lift axiom

We will define an axiom system and prove that it is sound with respect to scope equivalence. We will also prove that the system is sound with respect to garbage free terms.

Definition 5.5.2 The proof systems $\mathcal{R}_{\text{scope}}$ and $\mathcal{R}_{\text{gs}}$ are the systems $\mathcal{R}_{\text{graph}}$ and $\mathcal{R}_{\text{gc}}$ extended with the abstraction lift axiom:

$$[x]\langle M \mid D\rangle = \langle [x]M \mid D\rangle, \text{ if } x \text{ does not occur free in } D.$$

Lemma 5.5.3 (soundness) Given two term M, N.

(i) If $\mathcal{R}_{\text{scope}} \vdash M = N$ then $[\![M]\!] \sim_{\text{scope}} [\![N]\!]$.

(ii) If $\mathcal{R}_{\text{gs}} \vdash M = N$ then $[\![M]\!] \sim_{\text{gs}} [\![N]\!]$.

Proof. It is clear from the pictorial description of the abstraction lift axiom in Fig. 5.4 that the lift axiom only has an effect on the scopes of a graph. Thus, the soundness of $\mathcal{R}_{\text{scope}}$ is obvious. The soundness of $\mathcal{R}_{\text{gs}}$ follows from the fact that

$$g_1 \sim_{\text{scope}} g_2 \implies g_1 \sim_{\text{gs}} g_2 \ .$$

□

As a rewrite rule the lambda lift axiom suffers from a critical pair that is similar to the critical pair from which the garbage collection axiom suffers:

$$\begin{array}{ccc} [x]\langle\langle y \mid y = F(x,z)\rangle \mid z = A\rangle & \longrightarrow & \langle [x]\langle y \mid y = F(x,z)\rangle \mid z = A\rangle \\ \downarrow & & \\ [x]\langle y \mid y = F(x,z), z = A\rangle & & \end{array}$$

We can resolve this critical pair by using the rewrite rule:

$$[x]\langle M \mid D_1, D_2\rangle \xrightarrow[\Box-\text{lift}]{} \langle [x]\langle M \mid D_1\rangle \mid D_2\rangle \ ,$$

if neither x nor the variables defined in D_1 occur free in D_2. This rule allows trivial infinite sequences. For example:

$$[x]\langle y \mid\rangle \to \langle [x]\langle y \mid\rangle \mid\rangle \to \langle\langle [x]\langle y \mid\rangle \mid\rangle \mid\rangle \to \cdots$$

To make the rule terminating we require that D_2 is non-empty. The results are the following two rewrite systems:

Definition 5.5.4 The rewrite systems $\mathcal{R}^{\rightarrow}_{\text{scope}}$ and $\mathcal{R}^{\rightarrow}_{\text{gs}}$ are the rewrite systems $\mathcal{R}^{\rightarrow}_{\text{graph}}$ and $\mathcal{R}^{\rightarrow}_{\text{gc}}$ extended with the rule

$$[x]\langle M \mid D_1, D_2\rangle \xrightarrow[\Box-\text{lift}]{} \langle [x]\langle M \mid D_1\rangle \mid D_2\rangle, \text{ if } D_2 \neq \emptyset \ .$$

Proposition 5.5.5 The rewrite systems $\mathcal{R}^{\rightarrow}_{\text{scope}}$ and $\mathcal{R}^{\rightarrow}_{\text{gs}}$ are terminating and confluent.

Proof. To prove termination we will show that both rewrite system decrease the measure defined in 5.2.6. For $\mathcal{R}^{\rightarrow}_{\text{gc}}$ this has been shown in the proof of Lemma 5.3.3, so we only need to show that an application of the abstraction lift rule decreases the measure. By Lemma 5.2.7 we only need to prove that the measure of the left-hand side of the rule is larger than that of the right-hand side. If we compute these measures we get:

$$\begin{aligned} |[x]\langle M \mid D_1, D_2\rangle|^T_S &= S + inc(\{\{0\}\} + |M|^{\{\{0\}\}}_{\{\{1\}\}} + |D_1| + |D_2|) \\ |\langle [x]\langle M \mid D_1\rangle \mid D_2\rangle|^T_S &= \{\{0\}\} + S + inc(\{\{0\}\} + |M|^{\{\{0\}\}}_{\{\{1\}\}} + |D_1|) + |D_2| \end{aligned}$$

Because D_2 is never empty we have that the measure of the left-hand side is larger that that of the right-hand side.

Because we have termination, confluence is implied by weak confluence. In turn weak confluence can be proven by a lengthy case analysis, which is left to the reader. □

Lemma 5.5.6 The proof system $\mathcal{R}_{\text{gs}}$ is complete for $\sim_{\text{gs}}$.

Proof. Given two terms M, N, such that $[\![M]\!] \sim_{\text{gs}} [\![N]\!]$, we have to show that $\mathcal{R}_{\text{gs}} \vdash M = N$.

Let M_1 and N_1 be the $\mathcal{R}^{\rightarrow}_{\text{gs}}$ normal forms of M and N, respectively. Because

$$P \xleftrightarrow[\mathcal{R}^{\rightarrow}_{\text{gs}}]{} Q \iff \mathcal{R}_{\text{gs}} \vdash P = Q$$

we then have that $\mathcal{R}_{\text{gs}} \vdash M = M_1$ and $\mathcal{R}_{\text{gs}} \vdash N = N_1$.

Furthermore, it is possible to derive that the graphs of M_1 and N_1 are isomorphic, which means that we have that $\mathcal{R}_{\text{graph}} \vdash M_1 = N_1$.

We may then conclude that

$$\mathcal{R}_{\text{gs}} M = N \ .$$

□

The proof system $\mathcal{R}_{\text{scope}}$ is not complete. Consider the graphs in Fig. 5.5. These graphs are represented by

$$\langle z \mid x_0 = [y_0]\langle x \mid x_1 = [y_1]\langle y \mid x_2 = y_0\, y_1\rangle\rangle\rangle$$

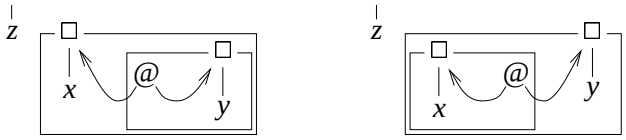


Figure 5.5: Example of scope equivalent graphs with garbage

and

$$\langle z \mid x_1 = [y_1]\langle y \mid x_0 = [y_0]\langle x \mid x_2 = y_0\, y_1\rangle\rangle\rangle \quad .$$

It is impossible to prove these terms equal using the lift axiom because that would require lifting the inner abstraction out of the scope of the outer abstraction, which is impossible because of the garbage application node which has to be inside the scopes of both abstractions. It is an open problem to find a sound and complete axiomatization of $\sim_{\text{scope}}$.

5.6 Unwinding

We have now reached the last axiomatization: that of terms representing graphs with the same unwinding. To axiomatize this equivalence, we will use $\mathcal{R}_{\text{graph}}$ extended with both bisimilarity and abstraction lift. (We do not have to add garbage collection, because garbage collection is a special case of bisimilarity.) We can formulate this extension as a union of proof systems.

Definition 5.6.1 Given two graphs g, h, we say that g and h have the same unwinding, denoted $g \sim_{\text{unw}} h$, if $g^l_u = g^l_u$. The proof system $\mathcal{R}_{\text{unw}}$ is defined by $\mathcal{R}_{\text{unw}} = \mathcal{R}_{\text{bisim}} \cup \mathcal{R}_{\text{scope}}$.

To prove the soundness and completeness of $\mathcal{R}_{\text{unw}}$ for $\sim_{\text{unw}}$ we need several lemmas. The first lemma states that term graphs are bisimilar if and only if they have the same unwinding.

Lemma 5.6.2 Given graphs g, h. We have that

$$g \underline{\leftrightarrow} h \iff g_u = h_u \quad .$$

The second lemma states that given two graphs g, h with the same (labeled) unwinding there are two other graphs g', h', such that g and g' have the same scoped unwinding, h and h' have the same scoped unwinding and g' and h' are equivalent modulo scopes.

Lemma 5.6.3 Given graphs g, h. If $g^l_u = h^l_u$ then there exist graphs g', h' such that $g^l_u = g_u$, $h^l_u = h_u$ and $g' \sim_{\text{scope}} h'$.

The last lemma states that graphs which are equivalent modulo scopes have the same labeled unwinding.

Table 5.3 An overview of the axioms.

Axiom			System
$F(\cdots,\langle M \mid D\rangle,\cdots)$	$=\langle F(\cdots,M_i,\cdots) \mid D\rangle$		} all
$\langle M \mid\rangle$	$= M$		
$\langle\langle M \mid D_1\rangle \mid D_2\rangle$	$=\langle M \mid D_1, D_2\rangle$		
$\langle M \mid x = \langle N \mid D_1\rangle, D_2\rangle$	$=\langle M \mid x = N, D_1, D_2\rangle$		
M	$=\langle x \mid x = M\rangle$	x fresh	
$\langle M \mid x = y, D\rangle$	$=\langle M[x := y] \mid D[x := y]\rangle$	$x \not\equiv y$	
$\langle M \mid x = x, D\rangle$	$=\langle M \mid x = \bullet, D\rangle$		
$\langle M \mid D\rangle$	$= M$		$\mathcal{R}_{\mathrm{gc}}, \mathcal{R}_{\mathrm{gs}}$
$[x]\langle M \mid D\rangle$	$=\langle [x]M \mid D\rangle$		$\mathcal{R}_{\mathrm{scope}}, \mathcal{R}_{\mathrm{gs}}, \mathcal{R}_{\mathrm{unw}}$
M	= N	$M \underline{\leftrightarrow} N$	$\mathcal{R}_{\mathrm{bisim}}, \mathcal{R}_{\mathrm{unw}}$

Lemma 5.6.4 Given graphs g, h. If $g \sim h$ then $g^l_u = h^l_u$.

Theorem 5.6.5 The proof system $\mathcal{R}_{\mathrm{unw}}$ is sound and complete for $\sim_{\mathrm{unw}}$.

Proof. Follows easily from previous lemmas. □

5.7 Summary

In Table 5.3 we have given an overview of the axioms of all the proof systems we have defined in this chapter. These proof systems were used to give axiomatizations of the equivalence relations in the hierarchy in Fig. 5.6. Except the proof system $\mathcal{R}_{\mathrm{scope}}$, which is only sound, all proof systems are sound and complete.

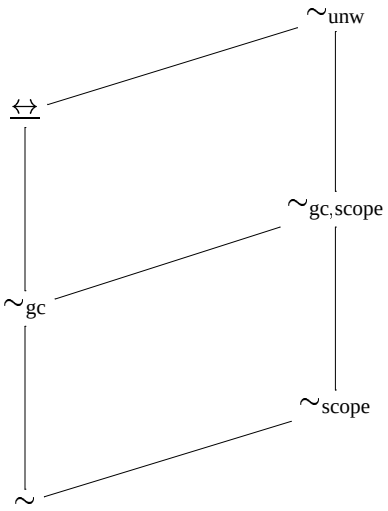


Figure 5.6: A hierarchy of equivalence relations

Chapter 6

Abstract Rewriting

In this chapter we will study the construction of Böhm trees and the basic principles of infinitary rewriting. This study will take place in the setting of Abstract Reduction Systems. In order to be able to develop some useful theory we must replace the sets of objects in ARSs with structures that model infinite objects. These structures express some useful properties of infinite objects. We have chosen to model infinite objects with algebraic complete partial orders, because in these structures the behavior of the infinite elements depends for a very large part on the behavior of the finite elements. For the theory developed in this thesis this is sufficient. Further development will require additional restrictions on the structures. For example, computation domains or concrete domains can be used (see [KP93]).

This chapter consists of four sections. In the first section, we define the notion of *skew confluence.* In the following sections, we study *Böhm trees*, *infinitary rewriting* and the relationship between them.

6.1 Skew Confluence

In this section we formally define the notion of *skew confluence*, which was introduced in Sect.1.4.2. We will also prove a few simple properties about skew confluence.

The formal definition of skew confluence reads:

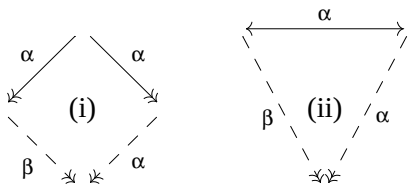


Figure 6.1: Skew Confluence and CR.

Definition 6.1.1 Given an ARS $\mathcal{A} \equiv (A, \xrightarrow[\alpha]{}, \xrightarrow[\beta]{})$. We have that $\xrightarrow[\alpha]{}$ is *skew confluent with respect to* $\xrightarrow[\beta]{}$ if

$$\forall a, b, c \in A, a \xrightarrow[\alpha]{}\!\!\!\!\!\rightarrow b, a \xrightarrow[\alpha]{}\!\!\!\!\!\rightarrow c : \exists d \in A : b \xrightarrow[\beta]{}\!\!\!\!\!\rightarrow d, c \xrightarrow[\alpha]{}\!\!\!\!\!\rightarrow d \ .$$

In Fig. 6.1 we have drawn two diagrams. Diagram (i) expresses that α is skew confluent with respect to β. Diagram (ii) expresses that α is *skew Church-Rosser* with respect to β. Skew CR is formally defined as:

Definition 6.1.2 We have that $\xrightarrow[\alpha]{}$ is *skew CR with respect to* $\xrightarrow[\beta]{}$ if

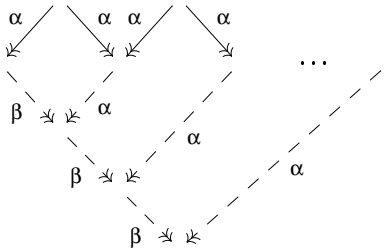
$$\forall a, b \in A, a \xleftarrow[\alpha]{}\!\!\!\!\!\rightarrow b : \exists c \in A : a \xrightarrow[\beta]{}\!\!\!\!\!\rightarrow c, b \xrightarrow[\alpha]{}\!\!\!\!\!\rightarrow c \ .$$

For any ARS we have that confluence and the CR property are equivalent. For any ARS we also have that skew confluence and the skew CR property are equivalent:

Proposition 6.1.3 Given an ARS $(A, \xrightarrow[\alpha]{}, \xrightarrow[\beta]{})$, we have that $\xrightarrow[\alpha]{}$ is skew confluent with respect to $\xrightarrow[\beta]{}$ if and only if $\xrightarrow[\alpha]{}$ is skew CR with respect to $\xrightarrow[\beta]{}$.

Proof.

"$\Rightarrow$" We can derive the skew CR from skew confluence by tiling as follows:



"$\Leftarrow$" Skew confluence is just a special case of skew CR.

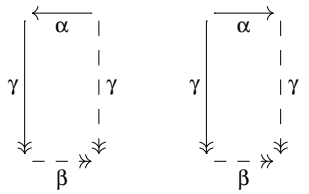
□

Commutativity is a generalization of confluence in the sense that a reduction relation $\rightarrow$ is confluent if and only if is commutes with itself. Similarly, we have that skew confluence is a generalization of confluence:

Proposition 6.1.4 Given an ARS $(A, \rightarrow)$. The ARS is confluent if and only if $\rightarrow$ is skew confluent with respect to $\rightarrow$.

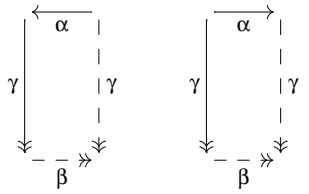
Proof. Elementary. □

In order to apply the notion of skew confluence successfully, we must have ways of proving skew confluence. We will now give a lemma that will help us later in this thesis. This lemma uses a third reduction relation. ($\xrightarrow[\gamma]{}$). The idea is that although $\xrightarrow[\alpha]{}$ is not confluent, we might be able to find a reduction relation $\xrightarrow[\gamma]{} \subset \xrightarrow[\alpha]{}\!\!\!\!\!\rightarrow$ that has better properties. For example, for $\xrightarrow[\gamma]{}$ two diagrams might hold:

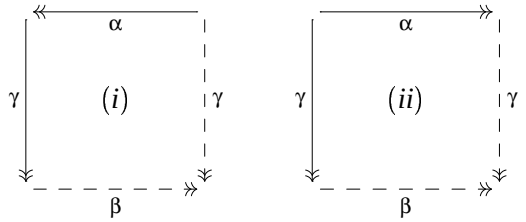


This would be good, because we can derive confluence up to from these diagrams:

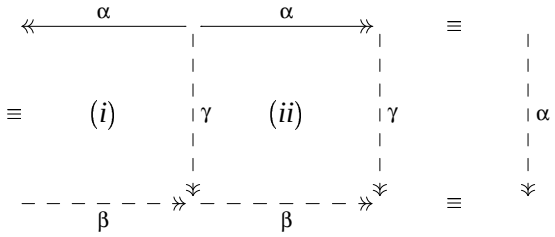
Lemma 6.1.5 Given an ARS $(A, \xrightarrow[i]{})_{i\in\{\alpha,\beta,\gamma\}}$, we have that $\xrightarrow[\alpha]{}$ is skew confluent with respect to $\xrightarrow[\beta]{}$ if $\xrightarrow[\gamma]{} \subset \xrightarrow[\alpha]{}\!\!\!\!\!\rightarrow$ and the following two diagrams hold:



Proof. From the given diagrams we can conclude that we also have the following two diagrams:

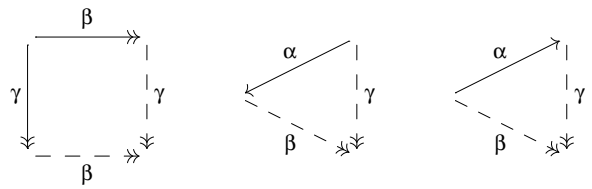


From these two diagrams we can prove the lemma with the following diagram:



□

To prove the two diagrams in the premise of the previous lemma, we can prove that the following three diagrams hold:



This completes the discussion of skew confluence in its own right. We will now focus on its main application.

6.2 Böhm Semantics

In this section we develop the theory of *infinite normal forms*, which is a generalization of the theory of Böhm trees to abstract reduction systems. This generalization will be based on the Böhm tree definition of Lévy, which we gave in section 1.4.1 of the introduction.

Seen from an abstract point of view the Böhm tree construction of Lévy works as follows. We have an ARS $(A, \to)$, a partial order $(B, \leq)$ and a monotonic function $\omega : A \to B$ from the elements of the ARS to the elements of the partial order. The infinite normal form of an element a is supposed to be the set of all information that can be found in reducts of that element. If we follow the definition of Lévy we would formalize that set as:

$$\downarrow \{\omega(b) \mid a \to\!\!\!\!\to b\} \quad . \tag{6.1}$$

This formalization works fine for computing the Böhm tree of a finite lambda calculus term, but for infinite terms there are two problems.

The first problem is due to the fact that we expect the Böhm tree of both finite and infinite lambda terms to be a possibly infinite lambda term. For finite lambda terms the set in 6.1 is an ideal over finite lambda terms which can be seen as a possibly infinite lambda term. For infinite lambda terms we get a set of infinite lambda terms. Even if this set is an ideal it cannot be seen as a possibly infinite lambda term.

The second problem is that for infinite terms we do not quite have the uniqueness we want. For example, let us consider the following reductions:

$$\begin{array}{l} (\lambda x.yx(yx(yx\cdots)))(Iz) \longrightarrow (\lambda x.yx(yx(yx\cdots)))z \longrightarrow yz(yz(yz\cdots)) \\ \qquad\qquad\qquad\quad \downarrow \\ y(Iz)(y(Iz)(y(Iz)\cdots)) \longrightarrow yz(y(Iz)(y(Iz)\cdots)) \longrightarrow\!\!\!\!\!\!\longrightarrow \cdots \end{array}$$

According to definition 6.1 the set of derivable information of $yz(yz(yz\cdots))$ is

$$\{\Omega,\ y\Omega\Omega,\ yz\Omega,\ y\Omega(y\Omega\Omega), \cdots\ ,yz(yz(yz\cdots))\}$$

and the set of derivable information of $y\,(I\,z)\,(y\,(I\,z)\,(y\,(I\,z)\cdots))$ is

$$\{\Omega,\ y\Omega\Omega,\ yz\Omega,\ y\Omega\,(y\Omega\Omega),\cdots\}\ .$$

These sets have the same least upper bound, but they are not equal.

We can solve these problems by only allowing finite elements in the set of derivable information:

$$\downarrow_{\mathcal{F}}\ \{\omega(b)\mid a{\twoheadrightarrow}b\}\ . \tag{6.2}$$

In the case of finite lambda terms this definition is equivalent to the old one and in the case of infinite lambda terms we get a set, which we can see as a possibly infinite lambda term. The problem of the two infinite terms with almost the same set of derivable information is also solved, because in an algebraic complete partial order we have that:

$$\operatorname{lub} S_1 = \operatorname{lub} S_2 \iff \downarrow_{\mathcal{F}} S_1 = \downarrow_{\mathcal{F}} S_2\ ,$$

for every two sets S_1, S_2, whose least upper bounds exist. We have now discussed all aspects of the construction of infinite normal forms. We summarize in the following definitions:

Definition 6.2.1 A structure $\mathcal{A} \equiv ((A,\to),\omega,(B,\leq))$ is an *ARS with information content* (ARSI) if $(A,\to)$ is an ARS, $(B,\leq)$ is a complete partial order and $\omega : A \to B$ is monotonic with respect to $\to$. We say that $\mathcal{A}$ is an abstract reduction system with finite information content if for every $a \in A$ we have that $\omega(a)$ is finite.

Given an ARSI $((A,\to),\omega,(B,\leq))$ we refer to $\omega(a)$ as the information content of a or as the direct approximation of a. The function ω induces a quasi order $\leq_\omega$ on A, defined by $a \leq_\omega a'$ if $\omega(a) \leq \omega(a')$. The ARSI that corresponds to the Böhm tree definition of Lévy is the structure

$$((\Lambda, \underset{\beta}{\to}), \omega_{\mathrm{BT}}, I(\omega_{\mathrm{BT}}(\Lambda), \leq_\Omega)\ . \tag{6.3}$$

This ARSI has finite information content. As a result of this, Lévy's definition of Böhm Tree coincides with our more general definition of infinite normal form:

Definition 6.2.2 Given an ARSI $((A,\to),\omega,(B,\leq))$. The infinite normal form $\mathrm{Inf}(a)$ of an element $a \in A$ is defined by

$$\mathrm{Inf}(a) = \downarrow_{\mathcal{F}} \{\omega(a') \mid a{\twoheadrightarrow}a'\}\ .$$

The ARSI has unique infinite normal forms (UN^∞) if

$$a \leftrightarrow a' \implies \mathrm{Inf}(a) = \mathrm{Inf}(a')\ .$$

We say that a notion of information content is *trivial* if the information content of every term is also the infinite normal form of that term.

Before we continue to study the conditions that imply uniqueness of infinite normal forms we will introduce some notation and a different characterization of the infinite normal form that is more convenient in proofs. The notation we need is a rewrite relation $\xrightarrow[\omega]{\mathcal{F}}$ that allows us to rewrite every element to every finite approximation of the information content of that element:

Definition 6.2.3 Given an ARSI $((A,\to),\omega,(B,\leq))$, we define the rewrite relation $\xrightarrow[\omega]{\mathcal{F}}$ as follows:

$$\forall b \in {\downarrow_{\mathcal{F}}}\,(\omega(a)) : a \xrightarrow[\omega]{\mathcal{F}} b$$

We then have the following facts:

$$\mathrm{Inf}(a) = \{b \mid a{\to\!\!\to} \xrightarrow[\omega]{\mathcal{F}} b\} \ ; \tag{6.4}$$

$$a \xrightarrow[\omega]{\mathcal{F}} b, b' \leq b, b' \text{ finite} \implies a \xrightarrow[\omega]{\mathcal{F}} b' \ ; \tag{6.5}$$

$$a \to a' \implies \mathrm{Inf}(a) \supset \mathrm{Inf}(a') \ . \tag{6.6}$$

To prove that the ARSIs in Equation 6.3 have unique infinite normal forms we use a theorem that states that every confluent ARSI has unique infinite normal forms:

Theorem 6.2.4 Given an ARSI $\mathcal{A} \equiv ((A,\to),\omega,(B,\leq))$. If $(A,\to)$ is confluent then $\mathcal{A}$ has unique infinite normal forms.

Proof. It suffices to show that if $a \to a'$ then $\mathrm{Inf}(a) = \mathrm{Inf}(a')$. We will distinguish two cases.

"$\supset$" If $a'{\to\!\!\to} \xrightarrow[\omega]{\mathcal{F}} b$ then also $a{\to\!\!\to} \xrightarrow[\omega]{\mathcal{F}} b$. Hence

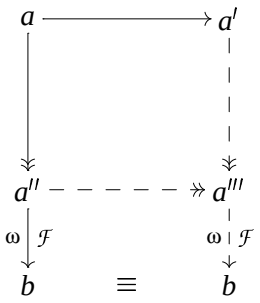
$$\{b \mid a'{\to\!\!\to} \xrightarrow[\omega]{\mathcal{F}} b\} \subset \{b \mid a{\to\!\!\to} \xrightarrow[\omega]{\mathcal{F}} b\} \ .$$

"$\subset$" Given that $a{\to\!\!\to}a'' \xrightarrow[\omega]{\mathcal{F}} b$, we have by confluence that there exists an a''' such that $a''{\to\!\!\to}a'''$ and $a'{\to\!\!\to}a'''$. By monotonicity we have that $\omega(a'') \leq \omega(a''')$. This implies that $a''' \xrightarrow[\omega]{\mathcal{F}} b$. In turn this implies that $a'{\to\!\!\to} \xrightarrow[\omega]{\mathcal{F}} b$ and hence

$$\{b \mid a{\to\!\!\to} \xrightarrow[\omega]{\mathcal{F}} b\} \subset \{b \mid a'{\to\!\!\to} \xrightarrow[\omega]{\mathcal{F}} b\} \ .$$

□

The second case in this proof can also be given in the form of a diagram:



The above theorem states that confluence is a sufficient condition for uniqueness. However, it is not a necessary condition.

Example 6.2.5 Consider the ARSI $(\mathbb{N},\rightarrow),\mathrm{id},(\overline{\mathbb{N}},\leq))$, where $n \rightarrow p \cdot n$ if p is a prime number larger than n. For example $1 \rightarrow 2 \rightarrow 6 \rightarrow 66$ and $1 \rightarrow 5 \rightarrow 55$. The ARS is not confluent because 66 will never reduce to an odd number and 55 will never reduce to an even number. However, the infinite normal form of every number is ∞, which means that we do have unique infinite normal forms.

Using the notion of skew confluence we can extend the result in Thm 6.2.4:

Theorem 6.2.6 Given an ARSI $\mathcal{A} \equiv ((A,\underset{\alpha}{\rightarrow}),\omega,(B,\leq))$ and another reduction relation $\underset{\gamma}{\rightarrow}$, we have that if $\underset{\gamma}{\rightarrow}$ is monotonic with respect to ω and $\underset{\alpha}{\rightarrow}$ is skew confluent with respect to $\underset{\gamma}{\rightarrow}$ then $\mathcal{A}$ has unique infinite normal forms.

Proof. Because of Eq. 6.6 it suffices to show that

$$a \underset{\alpha}{\rightarrow} a' \implies \mathrm{Inf}(a) \subset \mathrm{Inf}(a') \ .$$

Given $a \underset{\alpha}{\rightarrow} a'$ and $b \in \mathrm{Inf}(a)$, there exists an a'' such that $a \underset{\alpha}{\rightarrow\!\!\!\!\rightarrow} a'' \underset{\omega}{\overset{\mathcal{F}}{\rightarrow}} b$. Because of skew confluence we have that there exists an a''' such that $a' \underset{\alpha}{\rightarrow\!\!\!\!\rightarrow} a'''$ and $a'' \underset{\gamma}{\rightarrow\!\!\!\!\rightarrow} a'''$. Because $\underset{\gamma}{\rightarrow}$ is monotonic we can conclude that $\omega(a'') \leq \omega(a''')$. This and the fact that $a'' \underset{\omega}{\overset{\mathcal{F}}{\rightarrow}} b$ implies that $a''' \underset{\omega}{\overset{\mathcal{F}}{\rightarrow}} b$. □

The above theorem states that skew confluence is a sufficient condition for uniqueness of infinite norm forms. For ARSs with finite information content it is also necessary:

Theorem 6.2.7 Given an ARS with finite information content $\mathcal{A} \equiv ((A,\underset{\alpha}{\rightarrow}),\omega,(B,\leq))$, we have that $\mathcal{A}$ has unique infinite normal forms if and only if $\underset{\alpha}{\rightarrow}$ is skew confluent with respect to $\leq_\omega$.

Proof.

"⇒" Given $a \to\!\!\!\!\to a'$ and $a \to\!\!\!\!\to a''$ we must show the existence of an a''' such that $a'' \to\!\!\!\!\to a'''$ and $a' \leq_\omega a'''$. Because the information content is finite we have that $a' \xrightarrow[\omega]{\mathcal{F}} \omega(a')$. Because $\mathrm{Inf}(a') = \mathrm{Inf}(a'')$ we have that $a'' \to\!\!\!\!\to a''' \xrightarrow[\omega]{\mathcal{F}} \omega(a')$. For this a''' we also have that $a'' \leq_\omega a'''$.

"⇐" Corollary from Thm. 6.2.6.

□

In this characterization it is essential that we have finite information content. If we do not have this then you can have an ARSI with unique infinite normal forms, which is not skew confluent:

Example 6.2.8 Consider the ARSI $((\overline{\mathbb{N}} \times \overline{\mathbb{N}}, \to), \mathrm{id}, (\overline{\mathbb{N}} \times \overline{\mathbb{N}}, \leq))$, where

$$\begin{aligned}(0,0) &\to (\infty,0)\\ (0,0) &\to (0,\infty)\\ (\infty,n) &\to (\infty,n+1)\\ (n,\infty) &\to (n+1,\infty)\end{aligned}$$

and where $(m,n) \leq (p,q)$ if $m \leq p$ and $n \leq q$. The finite elements of the partial order are the elements of the set $\mathbb{N} \times \mathbb{N}$. The unique infinite normal form of any element is $\mathbb{N} \times \mathbb{N}$. However, $(\infty,0) \longleftrightarrow (0,\infty)$. None of the reducts of $(0,\infty)$ will be larger than $(\infty,0)$, so the ARSI is not skew confluent with respect to $\leq_\omega$.

In all of our examples the infinite normal form was an ideal and therefore had a least upper bound. For ARSs with finite information content it is known that skew confluence up to information content implies that the infinite normal form is a directed set, which implies that the least upper bound exists. The same holds if the information content is a c.p.o. where every two finite elements, which have an upper bound also have a finite upper bound. Terms with the $\leq_\Omega$ order fall into the latter category, because every ideal completion of a partial order falls into that category. For arbitrary c.p.o.'s it is an open question if the infinite normal form is a directed set or has a least upper bound.

6.3 Infinitary Rewriting

In this section we give a formal presentation of our abstract version of infinitary rewriting. More precisely, we will define an operator that is capable of extending rewrite relations from finite to infinite objects.

Kennaway et al. defined reduction on infinite terms, by extending the definition of substitution to infinite terms and applying the usual definition of rewrite step. Because in the setting of an ARS we do not have substitutions, we will have to use a different solution. The definition of Corradini [Cor93] is much more suitable. He defines how to contract an infinite set of redexes in an infinite term, by defining

how to contract that infinite set of redexes on every finite prefix of the infinite term. The result is then the least upper bound of the set of all reducts. This definition is complicated by the fact that it is required to rewrite every prefix of the left-hand side. Instead, we will just say that given a directed set, where each element rewrites to some other elements where the reducts form a second directed set, we have that the least upper bound of the first directed set rewrites to the least upper bound of the second directed set. We denote this extension with an operator $[\cdot\rangle$, which takes a rewrite relation and returns a rewrite relation. The transitive reflexive closure is denoted with $[\cdot\rangle\rangle$ and the conversion relation with $\langle\cdot\rangle$.

Definition 6.3.1 Given a complete partial order $(A,\leq)$, an abstract reduction system $(A',\to)$ with $A'\subseteq A$, we define the reduction relation $[\to\rangle$ by

$$\operatorname{lub} D_L [\to\rangle \operatorname{lub} D_R \ ,$$

where for some set I we have that $D_L=\{l_i\}_{i\in I}$ and $D_R=\{r_i\}_{i\in I}$ are directed sets, such that for all $i\in I$ we have that $l_i\to r_i$.

In diagrams we will use the arrows $\longrightarrow\!\!\boxed{\to}\!\!\longrightarrow$, $\longrightarrow\!\!\boxed{\to}\!\!\twoheadrightarrow$ and $\longleftarrow\!\!\boxed{\to}\!\!\longrightarrow$ to represent $[\to\rangle$,$[\to\rangle\rangle$ and $\langle\to\rangle$ respectively.

The lack of restrictions on the choice of rewrite relation and directed sets allows for a variety of rewrite relations on infinite terms. For example, consider infinite terms, ordered with the standard $\leq_\Omega$ and the following rewrite rules on finite terms:

$$\begin{array}{lcl} C & \to & A(B(C)) \\ A(X) & \to & X \\ B(X) & \to & X \end{array}$$

We then have that:

$$A(B(A(B(A(B(\cdots))))))\,[\to\rangle\, A(A(B(A(B(\cdots)))))\ ,$$

because

$$\begin{array}{lcl} A(B(\Omega)) & \to & A(\Omega) \\ A(B(A(B(\Omega)))) & \to & A(A(B(\Omega))) \\ A(B(A(B(A(B(\Omega))))))) & \to & A(A(B(A(B(\Omega))))) \\ & \vdots & \end{array}$$

We also have that:

$$A(B(A(B(A(B(\cdots))))))\,[\to\!\!\to\rangle\, A^\omega\ ,$$

because

$$\begin{array}{lcl} A(B(\Omega)) & \to\!\!\to & A(\Omega) \\ A(B(A(B(\Omega)))) & \to\!\!\to & A(A(\Omega)) \\ A(B(A(B(A(B(\Omega))))))) & \to\!\!\to & A(A(A(\Omega))) \\ & \vdots & \end{array}$$

Similarly, we have that

$$A(B(A(B(A(B(\cdots)))))) [\twoheadrightarrow\rangle B^\omega \ .$$

The same results can also be obtained by using infinite reduction sequences. For example:

$$A(B(A(B(A(B(\cdots)))))) \to A(A(B(A(B(\cdots))))) \to \cdots A^\omega \ . \tag{6.7}$$

Using infinite reduction sequences, the terms B^ω and A^Ω do not have a common reduct unless one adds a rewrite rule that allows them to be rewritten to Ω. In our approach we don't need to add these rules. That is, we have that

$$A^\omega[\twoheadrightarrow\rangle\Omega \text{ and } B^\omega[\twoheadrightarrow\rangle\Omega \ .$$

However, we cannot express the following sequence:

$$C \to A(B(C)) \to A(B(A(B(C)))) \to \cdots A(B(A(B(A(B(\cdots)))))) \ , \tag{6.8}$$

because even though we have that

$$\begin{array}{lll} C & \twoheadrightarrow & C \\ C & \twoheadrightarrow & A(B(C)) \\ C & \twoheadrightarrow & A(B(A(B(C)))) \\ C & \twoheadrightarrow & A(B(A(B(A(B(C)))))) \\ & \vdots & \end{array}$$

the right-hand sides do not form a directed set. The intuitive difference between 6.7 and 6.8 is that the former contracts only redexes that are present in the left-hand side and the latter contracts redexes that were created. In other words, the first sequence does an infinite number of redexes in parallel and the second does an infinite number in sequence. Parallel redexes seem a good application of this extension, but sequential redexes can better be handled by transfinite reduction sequences.

Given a labeled ARS $(A, \underset{l}{\to})_{l\in L}$, we often consider the reduction relation $\to=$ $\cup_{l\in L} \underset{l}{\to}$. This union operator and the infinitary extension do not commute, so we need to distinguish between the order of the union and the extension:

Definition 6.3.2 Given a labeled ARS $(A, \underset{l}{\to})_{l\in L}$, we have the following notation:

$$[\to\rangle = [\underset{l\in L}{\cup} \underset{l}{\to}\rangle \text{ and } [\underset{\cdot}{\to}\rangle = \underset{l\in L}{\cup} [\underset{l}{\to}\rangle \ .$$

With this notation we can define the contraction of a possible infinite set of redexes as follows: Let $(R, \underset{p}{\to})_{p\in \mathsf{Pos}}$ be a orthogonal TRS, labeled with positions. Given a set of positions P and terms M, N, we define $M \xrightarrow{P} N$, if for the set

$P' = P \cap \mathsf{Pos}(M)$ we have that N is the result of a complete development of the redexes at positions P'. The relation $[\underset{\cdot}{\rightarrow}\rangle$ now expresses the complete development of possibly infinite sets of redexes.

A very similar notion of parallel reduction has already been defined by Corradini in [Cor93]. The most important differences with the definition of Corradini are that we allow arbitrary directed sets for D_L instead of ideals and that we allow any reduction relation instead of a specifically chosen one.

The following lemma expresses an important property of the infinitary extension. It generalizes the fact that if given a CRS we have that $M[\rightarrow\rangle N$ then we have that every approximation of M can be extended to an approximation of M that rewrites to an approximation of N and every approximation of N extends to an approximation of N that is the result of rewriting an approximation of M.

Lemma 6.3.3 Given an algebraic CPO $(A, \leq)$, such that

$$\forall a, b \in \mathcal{F}(A) : \exists c \in A : a, b \leq c \implies \exists c \in \mathcal{F}(A) : a, b \leq c \ ,$$

and an ARS $(\mathcal{F}(A), \rightarrow)$, we have that

$$a[\rightarrow\rangle b \iff \left(\begin{array}{c} \forall a' \in{\downarrow_{\mathcal{F}}}(a) : \exists a'' \in{\downarrow_{\mathcal{F}}}(a), b'' \in{\downarrow_{\mathcal{F}}}(b) : a'' \rightarrow b'' \wedge a' \leq a'' \\ \wedge \\ \forall b' \in{\downarrow_{\mathcal{F}}}(b) : \exists a'' \in{\downarrow_{\mathcal{F}}}(a), b'' \in{\downarrow_{\mathcal{F}}}(b) : a'' \rightarrow b'' \wedge b' \leq b'' \end{array} \right) \ .$$

Proof. We distinguish two cases:

"$\Rightarrow$" Given $a[\rightarrow\rangle b$, we have that there exists a set I and sets $D_L = \{l_i\}_{i \in I}$ and $D_R = \{r_i\}_{i \in I}$, such that D_L and D_R are directed,for $i \in I$ we have that $l_i \rightarrow r_i$, $\mathrm{lub}\, D_L = a$ and $\mathrm{lub}\, D_R = b$. We now have two cases:

- Given $a' \in{\downarrow_{\mathcal{F}}}(a)$. Because a' is finite and $a' \leq \mathrm{lub}\, D_L$ there exists $a'' \in D_L$, such that $a' \leq a''$. For some i we have that $a'' = l_i$. Because $l_i \rightarrow r_i$, we have that a'' is finite. If we define $b'' = r_i$ then we have that

$$a'' \rightarrow b'' \wedge a' \leq a'' \ .$$

- Given $b' \in{\downarrow_{\mathcal{F}}}(b)$. Because b' is finite and $b' \leq \mathrm{lub}\, D_R$ there exists $b'' \in D_R$, such that $b' \leq b''$. For some i we have that $b'' = r_i$. Because $l_i \rightarrow r_i$, we have that b'' is finite. If we define $a'' = r_i$ then we have that

$$a'' \rightarrow b'' \wedge b' \leq b'' \ .$$

"$\Leftarrow$" Assume that

$$\begin{array}{c} \forall a' \in{\downarrow_{\mathcal{F}}}(a) : \exists a'' \in{\downarrow_{\mathcal{F}}}(a), b'' \in{\downarrow_{\mathcal{F}}}(b) : a'' \rightarrow b'' \wedge a' \leq a'' \\ \wedge \\ \forall b' \in{\downarrow_{\mathcal{F}}}(b) : \exists a'' \in{\downarrow_{\mathcal{F}}}(a), b'' \in{\downarrow_{\mathcal{F}}}(b) : a'' \rightarrow b'' \wedge b' \leq b'' \end{array} \ .$$

Define $I = \{\underline{l} \mid l \in{\downarrow_{\mathcal{F}}}\ (a), \exists r \in{\downarrow_{\mathcal{F}}}\ (b) : l \to r\} \uplus \{\overline{r} \mid r \in{\downarrow_{\mathcal{F}}}\ (b), \exists l \in{\downarrow_{\mathcal{F}}}\ (a) : l \to r\}$. Given $\underline{l} \in I$, there exist an $r \in{\downarrow_{\mathcal{F}}}\ (b)$ such that $l \to r$ and we define $l_{\underline{l}} = l$ and $r_{\underline{l}} = r$. Given $\overline{r} \in I$, there exist an $l \in{\downarrow_{\mathcal{F}}}\ (a)$ such that $l \to r$ and we define $l_{\overline{l}} = l$ and $r_{\overline{l}} = r$. Define $D_L = \{l_i \mid i \in I\}$ and $D_R = \{r_i \mid i \in I\}$.

To prove that $a[\to\rangle b$, we have to show that D_L and D_R are directed sets and that $\operatorname{lub} D_L = a$ and $\operatorname{lub} D_R = b$. We prove these claims below:

D_L is a directed set Given $a_1, a_2 \in D_L$, we have that $a_1, a_2 \in{\downarrow_{\mathcal{F}}}\ (a)$. This implies that $a_1, a_2 \leq a$, so there exists $a' \in{\downarrow_{\mathcal{F}}}\ (a)$ such that $a_1, a_2 \leq a'$. We also have that $\exists a'' \in{\downarrow_{\mathcal{F}}}\ (a), b'' \in{\downarrow_{\mathcal{F}}}\ (b) : a' \leq a'' \wedge a'' \to b''$. This implies that $a'' \in D_L$. We also have that $a_1, a_2 \leq a''$, so we have that D_L is a directed set.

D_R is a directed set Similar argument.

$\operatorname{lub} D_L = a$ By definition, we have that $D_L \subset{\downarrow_{\mathcal{F}}}\ (a)$. Thus, we have that $\operatorname{lub} D_L \leq \operatorname{lub}{\downarrow_{\mathcal{F}}}\ (a) = a$. Given $a' \in{\downarrow_{\mathcal{F}}}\ (a)$, we have that $\exists a'' \in{\downarrow_{\mathcal{F}}}\ (a), b'' \in{\downarrow_{\mathcal{F}}}\ (b) : a' \leq a'' \wedge a'' \to b''$. By definition, we have that $l_{\underline{a''}} = a''$. Thus, we have that $a'' \in D_L$. This implies that

$$a' \leq a'' \leq \operatorname{lub} D_L \ .$$

We can conclude that

$$\operatorname{lub}{\downarrow_{\mathcal{F}}}\ (a) \leq \operatorname{lub} D_L$$

$\operatorname{lub} D_R = b$ Similar argument.

□

In this thesis, we will be mainly interested in $[\cdot\rangle$ in conjunction with Böhm semantics. But as the definition itself is new, other questions may be asked.

For example, do we have the following compression property:

$$a[\to\to\rangle b[\to\to\rangle c \stackrel{?}{\Longrightarrow} a[\to\to\rangle c \ .$$

We will leave this as an open question, but not without remarking that at least some $\beta\eta$ sequences, which are not compressible in the infinitary lambda calculi of Kennaway et al [KKSdV97], can be compressed. For example, the reduction sequence

$$\lambda x.((\lambda y.z)\,x)^{\omega}\, x[\underset{\beta}{\to}\rangle \lambda x.(z\,((\lambda y.z)\,x)^{\omega})\, x[\underset{\beta}{\to\to}\rangle \lambda x.z^{\omega}\, x[\underset{\eta}{\to}\rangle z^{\omega} \ .$$

Because we have that

$$\lambda x.(((\lambda y.z)\,x)^{n}\,\Omega)\, x \underset{\beta}{\to\to} \lambda x.(z^{n}\,\Omega)\, x \underset{\eta}{\to} (z^{n}\,\Omega) \ ,$$

we have that

$$\lambda x.((\lambda y.z)\,x)^{\omega}\,x[\underset{\beta\eta}{\longrightarrow\!\!\!\!\!\rightarrow}\rangle z^{\omega} \quad .$$

Another question is, if given an orthogonal TRS $\mathcal{R}$, we have that $[\underset{\mathcal{R}}{\longrightarrow\!\!\!\!\!\rightarrow}\rangle$ is confluent? Again, we will leave this as an open question. We will continue with a study of Böhm Semantics and Infinitary Rewriting.

6.4 Böhm Semantics and Infinitary Rewriting

In the previous section we have shown how we can extend a rewrite relation on the finite elements of an algebraic cpo to a rewrite relation on the entire cpo. In this section we will show how to extend a notion of Böhm semantics from the finite elements to the whole cpo.

The extension of Böhm semantics can be done in two ways. First, we can directly define an infinite normal form based on the infinite normal form for finite elements. Second, we can define a notion of information content for arbitrary elements based on the information content of finite elements.

In the connecting text between the formal definitions and results, we will be talking about an algebraic cpo $(A,\leq)$ and an ARSI $((\mathcal{F}(A),\rightarrow),\omega,(B,\leq_B))$ with unique infinite normal forms defined on the finite elements of A. Note that in this setting we have to deal with three partial orders: $\leq_A$,$\leq_\omega$ and $\leq_B$. To avoid confusion we will refer to elements of A as objects and to elements of B as bits of information.

6.4.1 The direct extension

Defining the direct extension $\mathrm{Inf}^{\infty}_{\omega}$ of Inf_{ω} is simple. We expect the direct extension to be a set of finite bits of information just as the usual infinite normal form. Thus, we can define the infinite normal form of any element as the union of the infinite normal forms of all finite approximations of the element we want to compute. We also want this extension to be a true extension. That is, for all finite elements we want the extension to coincide with the original infinite normal form. Therefore, we will require that Inf_{ω} is monotonic with respect to $\leq_A$. That is, we require that:

$$\forall a,a' \in \mathcal{F}(A):\ a \leq_A a' \implies \mathrm{Inf}_{\omega}(a) \subset \mathrm{Inf}_{\omega}(a') \quad .$$

We summarize formally:

Definition 6.4.1 Given a cpo $(A,\leq_A)$ and an ARS with information content $((\mathcal{F}(A),\rightarrow),\omega,(B,\leq_B))$ with unique infinite normal forms, such that Inf_{ω} is monotonic with respect to $\leq_A$. The Böhm semantics of any element $a \in A$ is given by

$$\mathrm{Inf}^{\infty}_{\omega}(a) = \cup\{\mathrm{Inf}_{\omega}(a') \mid a' \in{\downarrow_{\mathcal{F}}}(a)\} \quad .$$

We must now check that the properties we expect do indeed hold. We will start by proving that $\mathrm{Inf}_\omega^\infty$ is an extension of $\mathrm{Inf}_\omega(a)$:

Proposition 6.4.2 Given a cpo $(A, \leq_A)$ and an ARS with information content $((\mathcal{F}(A), \to), \omega, (B, \leq_B))$ with unique infinite normal forms, such that Inf_ω is monotonic with respect to $\leq_A$, we have that:

$$\forall a \in \mathcal{F}(A) : \mathrm{Inf}_\omega(a) = \mathrm{Inf}_\omega^\infty(a) \ .$$

Proof. Let $a \in \mathcal{F}(A)$ be given. The fact that a is finite implies that $a \in \downarrow_{\mathcal{F}} (a)$. Therefore, we have that

$$\mathrm{Inf}_\omega(a) \subset \mathrm{Inf}_\omega^\infty(a) \ .$$

Because of the monotonicity of Inf_ω we have that

$$\forall a' \in \downarrow_{\mathcal{F}} (a) : \mathrm{Inf}_\omega(a') \subset \mathrm{Inf}_\omega(a).$$

Therefore, we have that:

$$\mathrm{Inf}_\omega^\infty(a) \subset \mathrm{Inf}_\omega(a) \ .$$

□

Apart from the fact that $\mathrm{Inf}_\omega^\infty$ is a real extension, we also expect that it is unique. That is, we expect that two convertible terms have the same extended infinite normal form. To prove that, we need the following lemma:

Lemma 6.4.3 Given a cpo $(A, \leq_A)$, a directed set $D \subset \mathcal{F}(A)$ and an ARS with information content $((\mathcal{F}(A), \to), \omega, (B, \leq_B))$ with unique infinite normal forms, such that Inf_ω is monotonic with respect to $\leq_A$, we have that:

$$\mathrm{Inf}_\omega^\infty(\mathrm{lub}\, D) = \cup\{\mathrm{Inf}_\omega(d) \mid d \in D\} \ .$$

Proof. We will distinguish two cases:

"$\supset$" Trivial.

"$\subset$" It suffices to show that for every $a \in \downarrow_{\mathcal{F}} (\mathrm{lub}\, D)$ there exists a $d \in D$ such that

$$\mathrm{Inf}_\omega(a) \subset \mathrm{Inf}_\omega(d) \ . \tag{6.9}$$

Because a is finite and $a \leq_A \mathrm{lub}\, D$, there exists a d such that $a \leq_A d$. By monotonicity of Inf_ω we have that 6.9 holds for this d.

□

With this lemma the uniqueness of the direct extension can be proven:

Theorem 6.4.4 Given a cpo $(A, \leq_A)$ and an ARS with information content $((\mathcal{F}(A), \to), \omega, (B, \leq_B))$ with unique infinite normal forms, such that Inf_ω is monotonic with respect to $\leq_A$, we have that:

$$a \langle \to\!\!\to \rangle b \implies \mathrm{Inf}_\omega^\infty(a) = \mathrm{Inf}_\omega^\infty(b) \ .$$

Proof. To prove the result it suffices to prove that

$$a[\twoheadrightarrow\rangle b \implies \mathrm{Inf}^{\infty}_{\omega}(a) = \mathrm{Inf}^{\infty}_{\omega}(b) \ .$$

If $a[\twoheadrightarrow\rangle b$ then for some index set I there exist directed sets $D_L = \{l_i\}_{i\in I}$ and $D_R = \{r_i\}_{i\in I}$, such that $\mathrm{lub}\, D_L = a$, $\mathrm{lub}\, D_R = b$ and for $i \in I$ we have that $l_i \to r_i$. We then have that

$$\begin{array}{rcll}
\mathrm{Inf}^{\infty}_{\omega}(a) & = & \mathrm{Inf}^{\infty}_{\omega}(\mathrm{lub}\, D_L) & \\
& = & \cup\{\mathrm{Inf}_{\omega}(d) \mid d \in D_L\} & \text{Lemma 6.4.3} \\
& = & \cup\{\mathrm{Inf}_{\omega}(l_i) \mid i \in I\} & \\
& = & \cup\{\mathrm{Inf}_{\omega}(r_i) \mid i \in I\} & \text{uniqueness of } \mathrm{Inf}_{\omega} \\
& = & \cup\{\mathrm{Inf}_{\omega}(d) \mid d \in D_R\} & \\
& = & \mathrm{Inf}^{\infty}_{\omega}(\mathrm{lub}\, D_R) & \text{Lemma 6.4.3} \\
& = & \mathrm{Inf}^{\infty}_{\omega}(b) &
\end{array}$$

□

6.4.2 The extension of the information content

From now on we will assume that $(A, \leq_A)$ is an algebraic semi lattice. This strengthening of the assumption allows us to conclude that $\downarrow_{\mathcal{F}}(a)$ is a directed set for any $a \in A$. If we also assume that ω is monotonic with respect to $\leq_A$ then for any $a \in A$ we have that $\{\omega(a') \mid a' \in \downarrow_{\mathcal{F}}(a)\}$ is a directed set. This allows us to define a notion of information content on A as follows:

Definition 6.4.5 Given an algebraic semi lattice $(A, \leq_A)$ and an ARSI $\mathcal{A} \equiv ((\mathcal{F}(A), \to), \omega, (B, \leq_B))$, such that ω is monotonic with respect to $\leq_A$, we define the the infinitary extension of $\mathcal{A}$ as follows:

$$\mathcal{A}_{\infty} = ((A, [\twoheadrightarrow\rangle), \omega_{\infty}, (B, \leq_B)) \ ,$$

where

$$\omega_{\infty}(a_{\infty}) = \mathrm{lub}\{\omega(a) \mid a \in \mathcal{F}(a_{\infty})\} \ .$$

We would like to have that for any $a \in A$ the two infinite normal forms are the same. However, monotonicity of ω and Inf_{ω} is not a sufficient condition for the equivalence of the two infinite normal forms. It is not even a sufficient condition for the uniqueness of $\mathrm{Inf}_{\omega_{\infty}}$. This is shown in the following example:

Example 6.4.6 We define the set A as:

$$A = \{\bot\} \cup \{1,2,3\} \times \overline{\mathbb{N}} \cup \{4\} \times \mathbb{N} \ .$$

On this set we define an order $\leq_A$:

$$\forall a \in A : \bot \leq_A a \text{ and } \forall i \in \{1,2,3\}, \forall m,n \in \overline{\mathbb{N}} : (i,n) \leq_A (i,m), \text{ if } n \leq m \ .$$

On the finite elements of A we define the following reduction relation:

$$\forall n \in \mathbb{N} : (1,n) \to (2,n), (1,n) \to (3,n), (2,n) \to (4,n) \ .$$

We define a function $\omega : \mathcal{F}(A) \to \mathbb{N}$ as follows:

$$\omega(\bot) = 0 \text{ and } \omega((i,n)) = n \ .$$

We then have that $((A, \leq_A), \omega, (\mathbb{N}, \leq))$ is an ARSI, that ω is monotonic with respect to $\leq_A$ and that

$$\mathrm{Inf}_\omega((i,n)) = \{0, 1, \cdots, n\} \ .$$

From this we can conclude that Inf_ω is monotonic with respect to $\leq_A$. If we compute the infinite normal forms of (i,∞) we get

$$\mathrm{Inf}^\infty_\omega((i,\infty)) = \mathrm{Inf}_{\omega_\infty}((1,\infty)) = \mathrm{Inf}_{\omega_\infty}((3,\infty)) = \mathbb{N} \ .$$

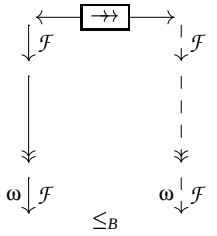
However, we also get

$$\mathrm{Inf}_{\omega_\infty}((2,\infty)) = \{0\} \ .$$

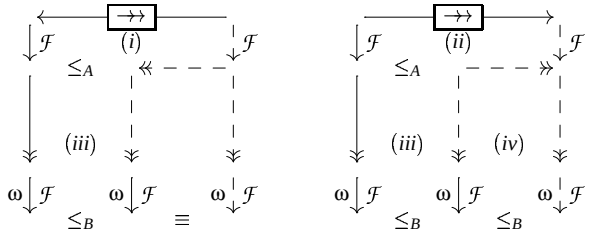
The problem in the example is that by rewriting finite elements we can get information that we cannot get by rewriting the least upper-bounds. So by rewriting infinite objects we may not get all information we can get by rewriting finite ones.

Given monotonicity of Inf_ω and ω, we have a positive result in the other direction. That is, every bit of information we can get by using the extended reduction relation can also be got by reducing finite approximations. The proof of this result requires the following lemma, which uses the notation $\xrightarrow{\mathcal{F}}$ for $\xrightarrow[\mathsf{id}]{\mathcal{F}}$:

Lemma 6.4.7 Given an algebraic semi lattice $(A, \leq_A)$ and an ARS with information content $\mathcal{A} \equiv ((\mathcal{F}(A), \to), \omega, (B, \leq_B))$ with unique infinite normal forms, such that ω and Inf_ω are monotonic, we have the following diagram:



Proof. The result follows easily from the following two diagrams:



(i) Given are a', a and b, such that $a \xrightarrow{\mathcal{F}} a'$ and $a\langle\twoheadrightarrow]b$. From the definition of $[\cdot\rangle$ we get that there are directed sets $\{a_i\}_{i\in I}$ and $\{b_i\}_{i\in I}$, such that $\text{lub}\{a_i\}_{i\in I} = a$, $\text{lub}\{b_i\}_{i\in I} = b$ and $b_i \twoheadrightarrow a_i$. Because a' is finite and $a' \leq_A a$, we have that there exists an index j such that $a' \leq_A a_j$. Because $\to$ is only defined on finite elements we have that $b \xrightarrow{\mathcal{F}} b_j$.

(ii) Similar to (i).

(iii) From monotonicity of Inf_ω.

(iv) From the uniqueness of Inf_ω and the fact that $\equiv \subset \leq_B$.

□

Corollary 6.4.8 Given an algebraic semi lattice $(A, \leq_A)$ and an ARSI $\mathcal{A} \equiv ((\mathcal{F}(A), \to), \omega, (B, \leq_B))$ with unique infinite normal forms, such that ω and Inf_ω are monotonic, we have that for any possibly infinite element a that

$$\text{Inf}_{\omega_\infty}(a) \subseteq \text{Inf}^\infty_\omega(a) \ .$$

Proof. We have to show that

$$a[\twoheadrightarrow\rangle c \implies \omega_\infty(c) \leq_B \text{Inf}^\infty_\omega(a) \ .$$

This follows from the previous lemma,

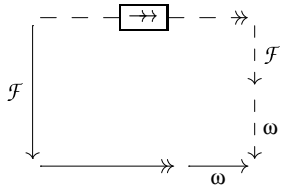
$$\text{Inf}^\infty_\omega(a) = \downarrow_{\mathcal{F}} \{b \mid a \xrightarrow[\mathcal{F}]{} \twoheadrightarrow \xrightarrow[\omega]{} b\}$$

and

$$\omega_\infty(c) = \downarrow_{\mathcal{F}} \{b \mid c \xrightarrow[\mathcal{F}]{} \xrightarrow[\omega]{} b\} \ .$$

□

Some analysis learns that $\text{Inf}^\infty_\omega(a) \subseteq \text{Inf}_{\omega_\infty}(a)$ is equivalent with the following diagram:

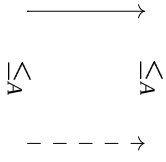


(6.10)

A very simple property that implies monotonicity of the infinite normal form and 6.10 is

$$a \leq_A a', a \to b \implies \exists b' : b \leq_A b', a' \to b' \ .$$

As a diagram this property looks like



All orthogonal term rewriting systems have this property. Not all orthogonal combinatory reduction systems have this property. For example, the lambda calculus with $\beta\eta$ reduction doesn't have this property because

$$\lambda x.x\,(y\,\Omega) \underset{\eta}{\rightarrow} y\,\Omega$$

but no $M \geq_\omega y\,\Omega$ such that

$$\lambda x.x\,(y\,x) \underset{\beta\eta}{\rightarrow\!\!\!\!\rightarrow} M$$

exists. The problem is that by expanding the subterm Ω to x we introduced an bound variable x into a subterm where the η-rule does not allow it. However, all orthogonal combinatory reduction systems where in the left-hand sides there are no restrictions on the occurrence of bound variables have the property. [1] Any CRS (and hence any TRS) with a non-left linear rule does not have this property.

[1] In CRS terminology: every CRS, such that in the left-hand side of every rule every metavariable has every possible bound variable as an argument has the property.

Chapter 7

Semantics of Graph Rewriting

One way of developing a graph rewriting system is to start with a term rewrite system or a higher-order rewrite system and extend this tree rewrite system to a graph rewrite system. When doing so it is convenient to use the letrec syntax for graphs, because this syntax is an extension of the syntax of term rewrite systems. An example of such a derived system is the following derivation of the lambda calculus:

$$\begin{array}{lcl} (\lambda x.M)\,N & \xrightarrow[\beta\circ]{} & \langle M \mid x = N\rangle \\ \langle C[x] \mid x = M, D\rangle & \xrightarrow[\text{es}]{} & \langle C[M] \mid x = M, D\rangle \\ \langle M \mid D\rangle\, N & \xrightarrow[\text{lift}]{} & \langle M\,N \mid D\rangle \end{array} \tag{7.1}$$

We will refer to a rewrite system developed in such a way as a cyclic extension. The rewrite rules of a cyclic extension can be divided into two subsets: the *work* rules that are directly derived from rules in the tree rewrite system and the *administrative* rules that are there only because they are needed to bring a term with letrec into a form that allows a rule of the first subset to be applied. For example, our simple cyclic extension of the lambda calculus has one work rule: $\xrightarrow[\beta\circ]{}$ and two administrative rules: $\xrightarrow[\text{es}]{}$ and $\xrightarrow[\text{lift}]{}$.

Another way of thinking of these graph rewrite systems is as the implementation of infinitary rewriting. That is, we think of terms with letrecs as representations of infinite terms and of graph rewrite steps as representations of complete developments of sets of redexes. This way of thinking makes it natural to require that rewriting a term with an administrative rule does not change the unwinding of that term.

By adding rules to the administrative part of a graph rewrite system we can make the graph rewrite system more powerful, but by doing so we can easily lose the confluence property.

7.1 Unwinding Calculi

In Chap. 3 we have implicitly defined the unwinding of a cyclic term by defining the graph of a term and the unwinding of a graph. Another way of obtaining the unwinding is to define a rewrite system and a notion of information content such that the infinite normal form is the unwinding. In this chapter we will study such rewrite systems with information content, which we will refer to as *unwinding calculi.* As long as the infinite normal forms are the unwindings, any notion of information content is allowed in an unwinding calculus. The result of this freedom is that any rewrite system that does not change the unwinding of a term can be seen as an unwinding calculus by using the unwinding itself as the information content.

As the formal definition of the unwinding of a term we will use the infinite normal form of the term with respect to a simple rewrite system. The rewrite system, that we will use to define the unwinding, is a translation of the μ-calculus.

For μ-terms, we can define the unwinding as the infinite normal form with respect to the μ-rule and the information content defined by:

$$\mu x.M \xrightarrow[\omega_\mu]{} \Omega \ .$$

Unfortunately, the straight-forward translation $\mu x.M \to \langle x \mid x = M \rangle$ does not work. If we translate the μ-rule based on this translation we get

$$\langle x \mid x = M \rangle \to M[x := \langle x \mid x = M \rangle] \ .$$

It is not obvious how we can extend this rule to allow for arbitrary terms in the external part and arbitrary sets of declarations. We solve this problem by using the translation

$$\mu x.M \to \langle M \mid x = M \rangle \ .$$

This translation does not preserve graphs. That is, the graph of $\mu x.M$ is usually not the same as the graph of $\langle M \mid x = M \rangle$. However, the graphs of those two terms do have the same unwinding. The resulting translation of the μ-rule is

$$\langle M \mid x = M \rangle \to M[x := \langle M \mid x = M \rangle] \ .$$

The rule for information content can also be translated:

$$\langle M \mid x = M \rangle \to \Omega \ .$$

Both translations can easily be extended to arbitrary external parts and sets of declarations. The result is the generalized μ-calculus, displayed in Table 7.1. Like the μ-calculus the generalized μ-calculus is an orthogonal CRS. Therefore, it is confluent and hence it has unique infinite normal forms. We use this rewrite system with information content for the formal definition of the unwinding of a term:

Table 7.1 The generalized μ-calculus

Axiom:

$$\langle M \mid \underbrace{x_1 = M_1, \cdots, x_n = M_n}_{D} \rangle \xrightarrow[\mu\circ]{} M[x_1 := \langle M_1 \mid D \rangle, \cdots, x_n := \langle M_n \mid D \rangle] \ .$$

Information content:

$$\langle M \mid D \rangle \xrightarrow[\omega_{\mu\circ}]{} \Omega \ .$$

Definition 7.1.1 Given a cyclic term M, we define the *unwinding* $\mathrm{Unw}(M)$ of M as:

$$\mathrm{Unw}(M) = \mathrm{Inf}_{\mu\circ}(M) \ .$$

Using this definition as a reference point we can define an unwinding calculus as a rewrite system with information content, whose unique infinite normal forms are equivalent to the unwinding. Formally:

Definition 7.1.2 A rewrite system with information content $\xrightarrow[\mathrm{unw}]{}$ is an unwinding calculus if it has unique infinite normal forms and for every cyclic term M we have $\mathrm{Unw}(M) \equiv \mathrm{Inf}_{\mathrm{unw}}(M)$.

Like rewrite systems with information content, unwinding calculi can have finite or infinite information content. The generalized μ-calculus is an example of an unwinding calculus with finite information content. Given any rewrite system $\rightarrow$ that preserves the unwinding, we can define an unwinding calculus with infinite information content by using Unw as the information content of the terms. In particular we can use the empty reduction relation, obtaining the *trivial unwinding calculus*:

Definition 7.1.3 The *trivial unwinding calculus* over a signature Σ is given by $((\mathsf{Terms}_\Sigma, \emptyset), \mathrm{Unw}, I(\mathsf{Terms}_\Sigma, \leq_\Omega))$.

We expect that the graph of the unwinding of a term is isomorphic to the unwinding of the graph of that term. in the remainder of this section we will sketch a proof of this fact. In the next section we continue with the relation between graph rewriting and infinitary rewriting.

A complete metric space on the set of finite graphs and infinite trees is formed by defining a metric as follows: given two graphs, we give every node a second label (the first label being the function symbol) that consists of the set of paths by which the nodes is accessible from the root. The distance between two different graphs is defined as 2^{-l}, where l is the length of the shortest path that exposes a difference between the two graphs. (A path that leads from the root to two nodes in the graph where the two labels do not match is a path the exposes a difference.)

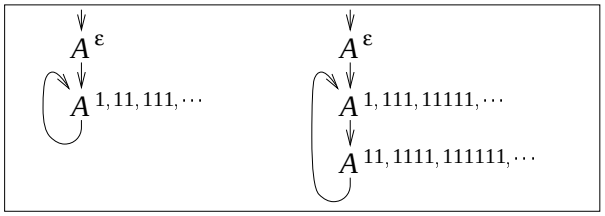


Figure 7.1: Labels for computing the distance between two graphs.

The distance between identical graphs is of course 0. For example, the distance between the two graphs in Fig. 7.1 is $\frac{1}{2}$.

Let $\xrightarrow[\mathrm{FS}(\mu\circ)]{}$ denote the result of doing a complete development of all $\mu\circ$ redexes. Given a term M_0 we consider the sequence

$$M_0 \xrightarrow[\mathrm{FS}(\mu\circ)]{} M_1 \xrightarrow[\mathrm{FS}(\mu\circ)]{} M_2 \xrightarrow[\mathrm{FS}(\mu\circ)]{} \cdots \quad .$$

We conjecture that this sequence has the following properties:

- The graphs of M_i have the same unwinding.
- The limit of the sequence

$$[\![M_0]\!], [\![M_1]\!], [\![M_2]\!], \cdots \tag{7.2}$$

  exists and is the same as the unwinding of M_0.
- The unwinding of the term M_0 is the limit of the sequence

$$\omega_{\mu\circ}(M_0), \omega_{\mu\circ}(M_1), \omega_{\mu\circ}(M_2), \cdots \quad . \tag{7.3}$$

- The limits of the sequences 7.2 and 7.3 are the same.

7.2 Properties of Cyclic Extensions

We described a cyclic extension of a rewrite system as a rewrite system on cyclic terms that was derived from the original rewrite system. We do not intend to give a precise formal definition of the notion of cyclic extension, but we will define two properties that cyclic extensions can have: *infinitary soundness* and *completeness.*

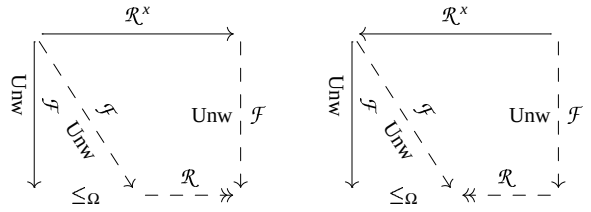
A cyclic extension is infinitarily sound with respect to a rewrite system on trees if we have that if a cyclic term reduces to another cyclic term using the cyclic extension then the unwinding of the first term reduces to the unwinding of the second term in the infinitary extension of the multi-step rewrite relation on the finite acyclic terms. Formally:

Definition 7.2.1 Given a CRS $\mathcal{R}$ and a cyclic extension $\mathcal{R}^x$ of $\mathcal{R}$. We say that $\mathcal{R}^x$ is *infinitarily sound* if

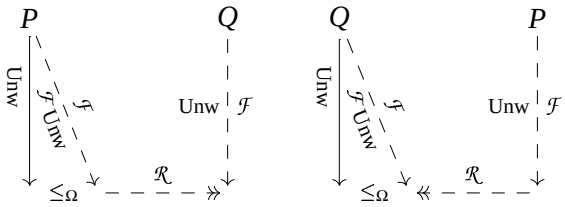
$$M \xrightarrow[\mathcal{R}^x]{} N \implies \mathrm{Unw}(M)\,[\xrightarrow[\mathcal{R}]{}\!\!\!\!\!\rightarrow\rangle\,\mathrm{Unw}(N) \quad .$$

Several proofs in this chapter will use diagrams. Therefore, we will formulate infinitary soundness in terms of diagrams in the next proposition:

Proposition 7.2.2 Given a CRS $\mathcal{R}$ and a cyclic extension $\mathcal{R}^x$ of $\mathcal{R}$, we have that $\mathcal{R}^x$ is infinitarily sound if and only if the following two diagrams hold:



Proof. By Lemma 6.3.3 we have that $P\,[\xrightarrow[\mathcal{R}]{}\!\!\!\!\!\rightarrow\rangle\,Q$ if and only if the following two diagrams hold:

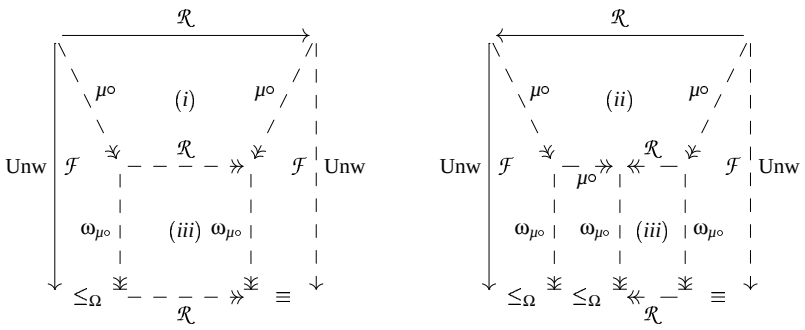


From this the result follows easily. □

The simplest possible cyclic extension of a CRS is that CRS itself. That is, the simplest extension is obtained by keeping the same rewrite rules and extending only the signature with letrec. This very simple extension is infinitarily sound:

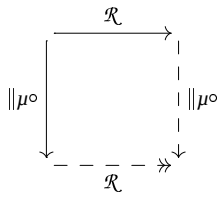
Lemma 7.2.3 Given an orthogonal CRS $\mathcal{R}$, we have that $\xrightarrow[\mathcal{R}]{}$ is an infinitarily sound cyclic extension of $\mathcal{R}$.

Proof. We have to show that the diagrams indicated in Prop. 7.2.2 hold, we will do this by decomposing them into smaller diagrams:



The numbered diagrams need further proof.

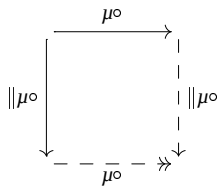
(i) Let us consider complete developments of $\mu\circ$-steps, denoted with $\|\,\mu\circ$. Because $\xrightarrow[\mathcal{R},\mu\circ]{}$ is an orthogonal CRS, we have that $\xrightarrow[\mathcal{R},\mu\circ]{}$ is confluent by complete developments. Thus, the following diagram holds:



(7.4)

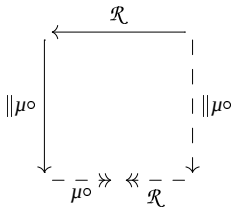
By tiling with this diagram we can derive diagram (i).

(ii) By the same argument used to prove 7.4, we also have:



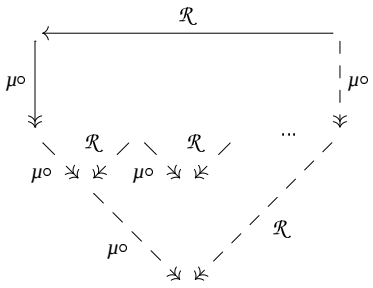
(7.5)

We also have the diagram:



(7.6)

By tiling we can then derive the diagram:

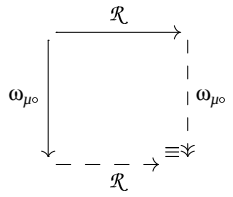


(iii) Given two cyclic terms M, N, we claim that

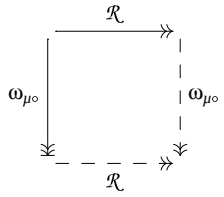
$$M \xrightarrow[\mathcal{R}]{}\!\!\!\!\rightarrow N \implies \omega_{\mu\circ}(M) \xrightarrow[\mathcal{R}]{}\!\!\!\!\rightarrow \omega_{\mu\circ}(N) \ .$$

This claim together with the fact that $\xrightarrow[\omega_{\mu\circ}]{}$ is confluent and terminating proves the diagrams marked with (iii).

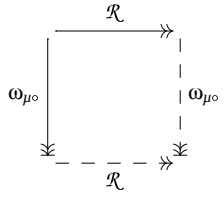
A simple analysis shows that the following diagram holds:



By tiling with this diagram we can prove that



Because contracting an $\mathcal{R}$-redex does not create an $\omega_{\mu\circ}$-redex we also have



Finally, the confluence and termination of $\xrightarrow[\omega_{\mu\circ}]{}$ allows us to conclude that the claim holds.

□

A rewrite rule from an unwinding calculus doesn't change the unwinding and therefore it is infinitarily sound with respect to every possible rewrite system on terms:

Lemma 7.2.4 Given a rewrite system on terms $\mathcal{R}$ and an unwinding calculus unw, we have that $\xrightarrow[\text{unw}]{}$ is an infinitarily sound cyclic extension of $\mathcal{R}$.

Proof. If $M \xrightarrow[\text{unw}]{} N$ then we have that $\text{Unw}(M) = \text{Unw}(N)$. This implies that

$$\text{Unw}(M)[\equiv\rangle\text{Unw}(N) \quad .$$

Because $\equiv \subset \xrightarrow[\mathcal{R}]{}$, we have that

$$\text{Unw}(M)[\xrightarrow[\mathcal{R}]{}\rangle\text{Unw}(N) \quad .$$

□

From Lemmas 7.2.3 and 7.2.4 we can immediately conclude that $\xrightarrow[\mathcal{R},\mu\circ]{}$ is an infinitarily sound cyclic extension of a given orthogonal CRS $\mathcal{R}$. By combining the two lemmas we can prove that every cyclic extension, which is a restriction of $\mathcal{R}$ modulo unwinding equivalence possibly extended with an unwinding calculus is infinitarily sound:

Theorem 7.2.5 Given a cyclic extension $\mathcal{R}^x$ of a CRS $\mathcal{R}$, we have that $\mathcal{R}^x$ is infinitarily sound if $\xrightarrow[\mathcal{R}^x]{} \subset (=_{\mathrm{Unw}} \circ \xrightarrow[\mathcal{R}]{} \circ =_{\mathrm{Unw}}) \cup =_{\mathrm{Unw}}$.

Proof. If $M \xrightarrow[\mathcal{R}^x]{} N$ then we have two cases:

- If $M =_{\mathrm{Unw}} N$ then by Lemma 7.2.4 $\mathrm{Unw}(M)[\xrightarrow[\mathcal{R}]{}\!\!\!\!\!\rightarrow\rangle \mathrm{Unw}(N)$.

- If $M =_{\mathrm{Unw}} M' \xrightarrow[\mathcal{R}]{} N' =_{\mathrm{Unw}} N$ then

$$\mathrm{Unw}(M) \equiv \underbrace{\mathrm{Unw}(M')[\xrightarrow[\mathcal{R}]{}\!\!\!\!\!\rightarrow\rangle \mathrm{Unw}(N')}_{\text{Lemma 7.2.3}} \equiv \mathrm{Unw}(N) \ .$$

□

We can apply this theorem to prove that the simple extension of the lambda calculus given at the start of this chapter is an infinitarily sound extension of the lambda calculus:

Example 7.2.6 The rewrite system in Equation 7.1 is a cyclic extension of the lambda calculus. We can show this by applying the previous theorem. Given a $\beta\circ$ rewrite step

$$C[(\lambda x.M)\,N] \xrightarrow[\beta\circ]{} C[\langle M \mid x = N\rangle] \ ,$$

we have that

$$C[(\lambda x.M)\,N] \xrightarrow[\beta]{} C[M[x := N]]$$

and that

$$C[\langle M \mid x = N\rangle] \xrightarrow[\mu\circ]{} C[M[x := \langle N \mid x = N\rangle]] \xrightarrow[\mu\circ]{}\!\!\!\!\!\rightarrow C[M[x := N]] \ .$$

From the $\mu\circ$-reduction we can conclude that

$$C[M[x := N]] =_{\mathrm{Unw}} C[\langle M \mid x = N\rangle] \ .$$

This means that we have that

$$C[(\lambda x.M)\,N] \xrightarrow[\beta]{} C[M[x := N]] =_{\mathrm{Unw}} C[\langle M \mid x = N\rangle]$$

and therefore that we have that

$$\xrightarrow[\beta^\circ]{} \subset =_{\mathrm{Unw}} \circ \xrightarrow[\beta]{} \circ =_{\mathrm{Unw}} \quad .$$

If we abbreviate $x_1 = M_1, \cdots, x_n = M_n$ with D then we have that

$$\begin{array}{ccc} \langle M \mid D\rangle N & \xrightarrow[\text{lift}]{} & \langle MN \mid D\rangle \\ \downarrow{\scriptstyle \mu^\circ} & & \downarrow{\scriptstyle \mu^\circ} \\ (M[x_i := \langle M_i \mid D\rangle])\, N & \equiv & (MN)[x_i := \langle M_i \mid D\rangle] \end{array}$$

and

$$\begin{array}{ccc} \langle C[x_1] \mid_D\rangle & \xrightarrow[\text{es}]{} & \langle C[M_1] \mid D\rangle \\ \downarrow{\scriptstyle \mu^\circ} & & \downarrow{\scriptstyle \mu^\circ} \\ C[x_1][x_i := \langle M_i \mid D\rangle] & \xrightarrow[\mu^\circ]{} & C[M_1][x_i := \langle M_i \mid D\rangle] \end{array}$$

From these diagrams we may conclude that

$$\xrightarrow[\text{lift,es}]{} \subset =_{\mathrm{Unw}} \quad .$$

If a cyclic extension of a CRS is infinitarily sound then we know that the cyclic extension does not equate more infinite terms than the infinitary extension of the same CRS. In general we do not have the inverse, that is if $\mathrm{Unw}(M)[\twoheadrightarrow\rangle \mathrm{Unw}(N)$ then we do not necessarily have that $M \xrightarrow[\mathcal{R}^x]{} N$. The reason is that to mimic the rewrite step on the unwinding you may need infinitely many steps on the cyclic term. For example, the unwinding of $\langle x \mid x = yx, y = I\rangle$ is $I(I(\cdots))$. We have that

$$I(I(\cdots))[\xrightarrow[\beta]{}\rangle \Omega \ ,$$

but in the cyclic extension $\xrightarrow[\beta,\mu^\circ]{}$ the term will never rewrite to term whose unwinding is Ω.

Fortunately, we do not have to look at reduction on infinite terms: reduction on infinite terms is derived from reduction on finite approximations of those terms, so we can simply consider reductions on finite approximations of the unwinding.

Given a cyclic term M, finite approximation P of the unwinding of M and a reduction $P \xrightarrow[\mathcal{R}]{} Q$ in a CRS $\mathcal{R}$, we may have for a cyclic extension $\mathcal{R}^x$ of $\mathcal{R}$ that $M \xrightarrow[\mathcal{R}^x]{} N$ in such a way that Q is a finite approximation of the unwinding of N. That is, the diagram

$$\begin{array}{ccc} & \overset{\mathcal{R}}{\longrightarrow} & \\ \mathrm{Unw}\uparrow\ \mathcal{F} & & \mathrm{Unw}\uparrow\ \mathcal{F} \\ & \underset{\mathcal{R}^x}{\dashrightarrow} & \end{array} \tag{7.7}$$

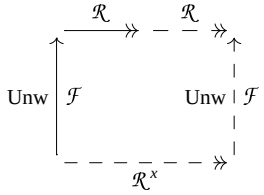
might hold. However, not in every cyclic extension do we have this diagram. For example, consider the following cyclic extension of the lambda calculus:

$$\begin{array}{ll}
(\lambda x.M)\,N & \to \langle M \mid x = N\rangle \\
\langle M \mid D\rangle N & \to \langle M\,N \mid D\rangle \\
\langle M \mid x = \langle N \mid D_1\rangle, D_2\rangle & \to \langle M \mid x = N, D_1, D_2\rangle \\
\langle C[x] \mid x = \lambda y.M, D\rangle & \to \langle C[\lambda y.M] \mid x = \lambda y.M, D\rangle \\
\langle M \mid x = \lambda y.N, z = C[x], D\rangle & \to \langle M \mid x = \lambda y.N, z = C[\lambda y.N], D\rangle \\
\langle M \mid x = \lambda y.C[x], D\rangle & \to \langle M \mid x = \lambda y.C[\lambda y.C[x]], D\rangle
\end{array}$$

The term $M \equiv \langle xyy \mid y = (\lambda u.u)\,z\rangle$ reduces to normal form as follows:

$$\langle xyy \mid y = (\lambda u.u)\,z\rangle \to \langle xyy \mid y = \langle u \mid u = z\rangle\rangle \to \langle xyy \mid y = u, u = z\rangle \quad .$$

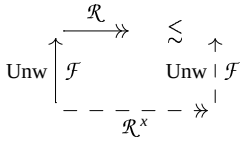
The unwinding of M is $x\,(I\,z)\,(I\,z)$. This unwinding is a finite approximation of itself and it reduces to $xz\,(I\,z)$. The only two reducts of M have xzz as its unwinding, so $xz\,(I\,z)$ cannot be the finite approximation of the unwinding of a reduct of M. We do have that $xz\,(I\,z) \underset{\beta}{\twoheadrightarrow} xz\,(I\,z)$, so we might hope that the diagram



(7.8)

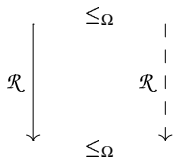
holds. Unfortunately it doesn't. For example, the term $x\,(I\,z)\,(\Omega z)$ is also a finite approximation of the unwinding of M and reduces to $xz\,(\Omega z)$. The latter term is not itself a finite approximation of the unwinding of a reduct of M nor does it reduce to one. We solve this problem by replacing the reduction $-\overset{\mathcal{R}}{-}\twoheadrightarrow$ with an unspecified relation$\lesssim$ and call the resulting property completeness up to $\lesssim$.

Definition 7.2.7 Given a CRS $\mathcal{R}$, a cyclic extension $\mathcal{R}^x$ of $\mathcal{R}$ and a relation $\lesssim$ on the terms of $\mathcal{R}$, we say that $\mathcal{R}^x$ is complete up to $\lesssim$ if the following diagram holds:



In many cases, we will be able to find a convenient relation for which we can prove completeness up to. For example, if we have a relation $\mathcal{R}$ that satisfies 7.7 then $\mathcal{R}$ is complete up to $\equiv$ and if $\mathcal{R}$ satisfies 7.8 then we have that $\mathcal{R}$ is complete up to $\underset{\mathcal{R}}{\twoheadrightarrow}$. A key property used to prove completeness up to is $\leq_\Omega$-monotonicity:

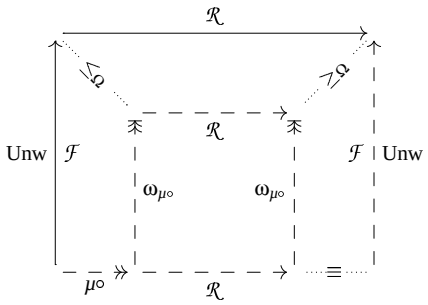
Definition 7.2.8 A CRS $\mathcal{R}$ is $\leq_\Omega$-monotonic if the following diagram holds:



The lambda calculus with β-reduction is $\leq_\Omega$-monotonic. With βη-reduction it is not. For $\leq_\Omega$-monotonic orthogonal CRSs we have the following result:

Proposition 7.2.9 Given an $\leq_\Omega$-monotonic orthogonal CRS $\mathcal{R}$, we have that the cyclic extension $\xrightarrow[\mathcal{R},\mu\circ]{}$ is complete up to $\equiv$.

Proof. We have the following diagram:



The result follows from this diagram by means of a simple tiling argument. □

7.3 Böhm Semantics of Cyclic Extensions

In the previous chapter we have shown how to extend Böhm semantics from a CRS to its infinitary extension. To be able to define this extension we need the CRS to have a notion of information content, such that its infinite normal forms are unique and both its information content and those normal forms are monotonic with respect to $\leq_\Omega$. Because we will need these conditions as standard, we will call a notion of information content regular if it has these properties:

Definition 7.3.1 We say that the information content of a CRS with information content $\mathcal{R}$ is *regular* if

- $\mathcal{R}$ has unique infinite normal forms;
- $\omega_\mathcal{R}$ is monotonic with respect to $\leq_\Omega$;
- $\mathrm{Inf}_\mathcal{R}$ is monotonic with respect to $\leq_\Omega$.

We will also need well-behaved cyclic extensions. That is, we will need cyclic extensions that are infinitarily sound and complete up to a useful relation. For this useful relation we choose $\leq_{\omega_{\mathcal{R}}}$ and we refer to a cyclic extension with these properties as a *regular cyclic extension*:

Definition 7.3.2 A cyclic extension $\mathcal{R}^x$ of a CRS $\mathcal{R}$ with information content is *regular* if $\mathcal{R}^x$ is infinitarily sound and complete up to $\leq_{\omega_{\mathcal{R}}}$.

Given a CRS with regular information content $\mathcal{R}$ and a cyclic extension $\mathcal{R}^x$ of $\mathcal{R}$ that contains an unwinding calculus unw, we can now define the information content of a cyclic term as the information contained in the part of the cyclic term that has become stable with respect to unwinding:

Definition 7.3.3 Given a complete and infinitarily sound cyclic extension $\mathcal{R}^x$ of a CRS with regular information content $\mathcal{R}$ containing an unwinding calculus unw, we define the function $\omega^{\circ}_{\mathcal{R}^x}$ by

$$\omega^{\circ}_{\mathcal{R}^x} = \omega^{\infty}_{\mathcal{R}} \circ \omega_{\text{unw}} \quad .$$

Note that if the information content of the unwinding calculus is finite we have that

$$\omega^{\circ}_{\mathcal{R}^x} = \omega_{\mathcal{R}} \circ \omega_{\text{unw}} \quad ,$$

because $\omega^{\infty}_{\mathcal{R}}$ is a proper extension of $\omega_{\mathcal{R}}$. The function $\omega^{\circ}_{\mathcal{R}^x}$ is not necessarily a notion of information content, as the following example shows:

Example 7.3.4 Let $\mathcal{R}$ be the term rewrite system with the single rule

$$A(X) \to B(X) \quad .$$

The information content of a term is defined as the normal form of the term:

$$\omega_{\mathcal{R}}(M) = \mathcal{R}(M).$$

This notion of information content is regular. Define

$$\xrightarrow[\mathcal{R}^x]{} = \xrightarrow[\mathcal{R},\mu\circ]{} \quad .$$

As the unwinding calculus of $\mathcal{R}^x$ we consider the rewrite rule $\mu\circ$ with information content defined by

$$\begin{aligned} \langle A(M) \mid x_1 = M_1, \cdots, x_n = M_n \rangle &\xrightarrow[\omega_{\text{unw}_{\mathcal{R}}}]{} A(M)[x_1 := \Omega, \cdots, x_n := \Omega] \\ \langle B(M) \mid x_1 = M_1, \cdots, x_n = M_n \rangle &\xrightarrow[\omega_{\text{unw}_{\mathcal{R}}}]{} \Omega \end{aligned}$$

Even with this modified notion of information content $\text{unw}_{\mathcal{R}}$ is an unwinding calculus. We now have that

$$\langle A(x) \mid\rangle \xrightarrow[\mathcal{R}^x]{} \langle B(x) \mid\rangle \quad .$$

However,

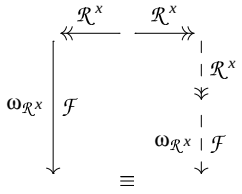
$$\begin{aligned}\omega^{\circ}_{\mathcal{R}}(\langle A(x)\,|\rangle) &= \mathcal{R}(A(x)) = B(x)\\ \omega^{\circ}_{\mathcal{R}}(\langle B(x)\,|\rangle) &= \mathcal{R}(\Omega) \quad = \Omega\end{aligned}$$

Thus, the function $\omega^{\circ}_{\mathcal{R}}$ is not monotonic with respect to $\xrightarrow[\mathcal{R}^x]{}$ and hence it is not a notion of information content. Note that $\mathcal{R}^x$ is a regular cyclic extension of $\mathcal{R}$.

If $\omega^{\circ}_{\mathcal{R}^x}$ is a notion of information content then it will generate unique infinite normal forms if in addition $\mathcal{R}^x$ is a complete and infinitarily sound cyclic extension:

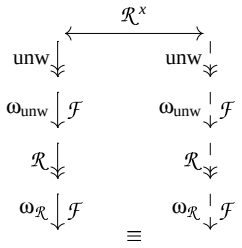
Theorem 7.3.5 Given a regular cyclic extension $\mathcal{R}^x$ of a CRS with regular information content $\mathcal{R}$ containing an unwinding calculus unw, such that $\omega_{\mathcal{R}^x}$ is a notion of information content, we have that $\mathcal{R}^x$ has unique infinite normal forms.

Proof. To prove that $\mathcal{R}^x$ has unique infinite normal forms, we have to prove that the following diagram holds:

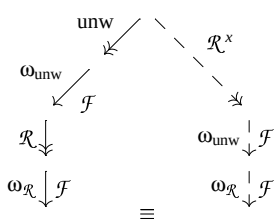


This diagram can be proven easily using the following claims:

$$M \xrightarrow[\omega_{\mathcal{R}^x}]{\mathcal{F}} a \iff M \xrightarrow[\omega_{\mathrm{unw}}]{\mathcal{F}} \xrightarrow[\omega_{\mathcal{R}}]{\mathcal{F}} a \tag{7.9}$$



(7.10)



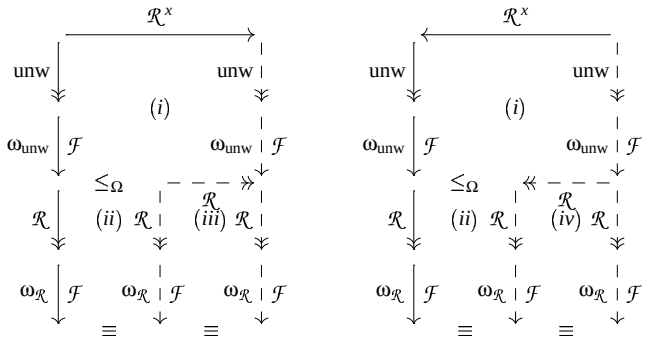
(7.11)

To complete the proof, we must prove the claims:

(7.9)

$$\begin{aligned} M \xrightarrow[\omega_{\mathcal{R}^x}]{\mathcal{F}} a &\iff a \in{\downarrow_{\mathcal{F}}} \left(\omega_{\mathcal{R}^x}(M)\right) = {\downarrow_{\mathcal{F}}} \left(\omega_{\mathcal{R}}^{\infty}(\omega_{\mathrm{unw}}(M))\right) \\ &\iff a \in{\downarrow_{\mathcal{F}}} \left(\mathrm{lub}\{\omega_{\mathcal{R}}(N) \mid N \in{\downarrow_{\mathcal{F}}} \left(\omega_{\mathrm{unw}}(M)\right)\}\right) \\ &\iff a \in{\downarrow_{\mathcal{F}}} \left\{\omega_{\mathcal{R}}(N) \mid N \in{\downarrow_{\mathcal{F}}} \left(\omega_{\mathrm{unw}}(M)\right)\right\} \\ &\iff \exists N \in{\downarrow_{\mathcal{F}}} \left(\omega_{\mathrm{unw}}(M)\right) : a \in{\downarrow_{\mathcal{F}}} \left(\omega_{\mathcal{R}}(N)\right) \\ &\iff M \xrightarrow[\omega_{\mathrm{unw}}]{\mathcal{F}} \xrightarrow[\omega_{\mathcal{R}}]{\mathcal{F}} a \end{aligned}$$

(7.10) This diagram can be obtained by tiling with the following two diagrams:



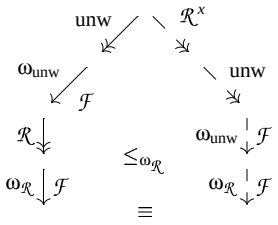
(i) From the infinitary soundness of $\mathcal{R}^x$ by applying Prop. 7.2.2.

(ii) From the monotonicity of $\mathrm{Inf}_{\mathcal{R}}$ with respect to $\leq_\Omega$.

(iii) From the uniqueness of the infinite normal form of $\mathcal{R}$.

(iv) Trivial.

(7.11) Because $\xrightarrow[\mathrm{unw}]{} \subset \xrightarrow[\mathcal{R}^x]{}$ the following diagram proves the claim:



The top part of this diagram follows from the completeness of $\mathcal{R}^x$ up to $\leq_{\omega_{\mathcal{R}}}$. The bottom part is elementary.

□

The information content of a cyclic term depends not only on the information content of the finite terms, but also on the chosen unwinding calculus. We will now show that the choice of unwinding calculus has no effect on the resulting infinite normal forms. We will do so by showing that no matter what the choice of unwinding calculus is, we always have that the infinite normal form of the cyclic term is equal to the infinite normal form of the unwinding of the cyclic term:

Theorem 7.3.6 Given a regular cyclic extension $\mathcal{R}^x$ of a CRS with regular information content $\mathcal{R}$ containing an unwinding calculus unw, such that $\omega_{\mathcal{R}^x}$ is a notion of information content, we have for every cyclic term M that:

$$\mathrm{Inf}_{\mathcal{R}^x}(M) = \mathrm{Inf}^{\infty}_{\mathcal{R}}(\mathrm{Unw}(M)) \ .$$

Proof. By some elementary reasoning we can derive that

$$\mathrm{Inf}^{\infty}_{\mathcal{R}}(\mathrm{Unw}(M)) = \{a \mid M \xrightarrow[\mathrm{Unw}]{\mathcal{F}} \xrightarrow[\mathcal{R}]{}\!\!\!\!\!\rightarrow \xrightarrow[\omega_{\mathcal{R}}]{\mathcal{F}} a\} \ .$$

The claims 7.9,7.10 and 7.11 are again valid. From these claims we can prove that

$$M \xrightarrow[\mathcal{R}^x]{}\!\!\!\!\!\rightarrow \xrightarrow[\omega_{\mathcal{R}^x}]{\mathcal{F}} a \iff M \xrightarrow[\mathrm{Unw}]{\mathcal{F}} \xrightarrow[\mathcal{R}]{}\!\!\!\!\!\rightarrow \xrightarrow[\omega_{\mathcal{R}}]{\mathcal{F}} a \ .$$

Hence, we have that

$$\mathrm{Inf}_{\mathcal{R}^x}(M) = \{a \mid M \xrightarrow[\mathrm{Unw}]{\mathcal{F}} \xrightarrow[\mathcal{R}]{}\!\!\!\!\!\rightarrow \xrightarrow[\omega_{\mathcal{R}}]{\mathcal{F}} a\} \ .$$

□

Because we have that $\mathrm{Inf}^{\infty}_{\mathcal{R}}$ is a proper extension of $\mathrm{Inf}_{\mathcal{R}}$, an important corollary from this theorem is that $\mathrm{Inf}_{\mathcal{R}^x}$ is a proper extension of $\mathrm{Inf}_{\mathcal{R}}$:

Corollary 7.3.7 Given a complete and infinitarily sound cyclic extension $\mathcal{R}^x$ of a CRS with regular information content $\mathcal{R}$ containing an unwinding calculus unw, such that $\omega_{\mathcal{R}^x}$ is a notion of information content, we have for every plain term M that:

$$\mathrm{Inf}_{\mathcal{R}^x}(M) = \mathrm{Inf}_{\mathcal{R}}(M) \ .$$

We have now shown how to extend uniqueness results from Böhm semantics on plain terms to cyclic terms. We will continue with a study of a property, which we could not introduce on abstract rewriting systems: *syntactic continuity.*

7.4 Syntactic continuity of Böhm Semantics

Böhm tree equivalence is a compatible relation. That is, if two terms M and N have the same Böhm tree then for any context C we have that $C[M]$ and $C[N]$ have the same Böhm tree. To prove this for the lambda calculus, Lévy ([Lév78]) showed that the Böhm tree had a property called syntactic continuity. This property implies that Böhm tree equivalence is compatible.

Syntactic continuity is a property which can be defined on rewrite systems, where the infinite normal forms are possibly infinite terms. The property states that given any context C and any term M, the infinite normal form of $C[M]$ depends only on C and the infinite normal form of M:

Definition 7.4.1 A rewrite system with information content $\mathcal{R}$ is syntactically continuous if

$$\mathrm{Inf}_{\mathcal{R}}(C[M]) = \cup\{\mathrm{Inf}_{\mathcal{R}}(C[P] \mid P \in \mathrm{Inf}_{\mathcal{R}}(M)\} \ .$$

The goal in this section is to find conditions on CRSs and their cyclic extensions that allow us to prove that the cyclic extensions are syntactically continuous. An obvious condition is the requirement that the CRS is syntactically continuous, but there is another requirement: *substitutive continuity*:

Definition 7.4.2 A rewrite system with information content $\mathcal{R}$ is substitutive continuous if

$$\mathrm{Inf}_{\mathcal{R}}(M\sigma) = \cup\{\mathrm{Inf}_{\mathcal{R}}(P\sigma \mid P \in \mathrm{Inf}_{\mathcal{R}}(M)\} \ .$$

In the lambda calculus with the β-rule we can derive this property from syntactic continuity if the free variables in the infinite normal form are a subset of those in the term. Given a term M and a substitution σ, let $\vec{x}$ be a vector of variables such that $FV(M) = \{\vec{x}\}$. We then have

$$\begin{aligned}\mathrm{Inf}(M\sigma) &= \mathrm{Inf}(((\lambda\vec{x}.M)\,\vec{x})\sigma) \\ &= \cup\{\mathrm{Inf}(((\lambda\vec{x}.P)\,\vec{x})\sigma) \mid P \in \mathrm{Inf}(M)\} \\ &= \cup\{\mathrm{Inf}(P\sigma) \mid P \in \mathrm{Inf}(M)\} \ .\end{aligned}$$

It is possible for a CRS with information content to have unique infinite normal forms and be syntactically continuous, without also being substitutive continuous. This is shown in the following example:

Example 7.4.3 Given the CRS

$$\begin{aligned} B(C) &\to C \\ B(A) &\to B(A) \\ F\,x.Z(x) &\to Z(A)\end{aligned}$$

with information content defined by

$$\begin{aligned} B(Z) &\xrightarrow[\omega]{} \Omega \\ F\,x.Z(x) &\xrightarrow[\omega]{} Z(\Omega)\end{aligned}$$

This CRS is syntactically continuous, but it is not substitutive continuous:

$$\begin{aligned} C &= \mathrm{Inf}(B(C)) = \mathrm{Inf}(B(x)[x := C]) \\ &\neq \cup\{\mathrm{Inf}(P[x := C]) \mid P \in \mathrm{Inf}(B(x))\} \\ &= \cup\{\mathrm{Inf}(P[x := C]) \mid P \in \Omega\} \\ &= \Omega \ .\end{aligned}$$

The example showed that for CRSs syntactic continuity does not imply substitutive continuity. However, for cyclic extensions syntactic continuity does imply substitutive continuity:

Proposition 7.4.4 Given a cyclic extension $\mathcal{R}^x$ of a CRS $\mathcal{R}$, we have that $\mathcal{R}^x$ is substitutive continuous if $\mathcal{R}^x$ is syntactically continuous.

Proof. Given a term M and a substitution σ, let $\vec{x}$ be a vector of variables such that $FV(M) = \{\vec{x}\}$. We then have

$$\begin{aligned}\mathrm{Inf}(M\sigma) &= \mathrm{Inf}(\langle M \mid x_1 = x_1\sigma, \cdots, x_n = x_n\sigma\rangle)\\ &= \cup\{\mathrm{Inf}(\langle P \mid x_1 = x_1\sigma, \cdots, x_n = x_n\sigma\rangle) \mid P \in \mathrm{Inf}(M)\}\\ &= \cup\{\mathrm{Inf}(P\sigma) \mid P \in \mathrm{Inf}(M)\} \ .\end{aligned}$$

□

Because for plain terms we have that $\mathrm{Inf}_{\mathcal{R}}(M) = \mathrm{Inf}_{\mathcal{R}^x}(M)$, we can conclude from the proposition and the example that we will need to require substitutive continuity.

We are now going to develop the theory that we need in order to prove our main result. The first thing we need is an operation on multiple hole contexts, that fills all the holes with the same term:

Definition 7.4.5 Given a multiple hole context C and a term M, we define

$$C\{M\} = C[M, \cdots, M] \ .$$

With this notation we can prove a simple extension of syntactic continuity from single hole contexts to multiple hole contexts:

Lemma 7.4.6 Given a CRS with syntactically continuous regular information content, we have that:

$$\mathrm{Inf}_{\mathcal{R}}(C\{M\}) = \cup\{\mathrm{Inf}_{\mathcal{R}}(C\{a\}) \mid a \in \mathrm{Inf}_{\mathcal{R}}(M)\} \ .$$

Proof.

$$\begin{aligned}&\mathrm{Inf}_{\mathcal{R}}(C\{M\})\\ &= \mathrm{Inf}_{\mathcal{R}}(C[M, \cdots, M])\\ &= \cup\{\mathrm{Inf}_{\mathcal{R}}(C[a_1, M, \cdots, M]) \mid a_1 \in \mathrm{Inf}_{\mathcal{R}}(M)\} && (1)\\ &\cdots\\ &= \cup\{\cdots \cup \{\mathrm{Inf}_{\mathcal{R}}(C[a_1, \cdots, a_n]) \mid a_n \in \mathrm{Inf}_{\mathcal{R}}(M)\} \cdots \mid a_1 \in \mathrm{Inf}_{\mathcal{R}}(M)\} && (1)\\ &= \cup\{\mathrm{Inf}_{\mathcal{R}}(C[a_1, \cdots, a_n]) \mid a_i \in \mathrm{Inf}_{\mathcal{R}}(M)\}\\ &= \cup\{\mathrm{Inf}_{\mathcal{R}}(C[a, \cdots, a]) \mid a \in \mathrm{Inf}_{\mathcal{R}}(M)\} && (2)\\ &= \cup\{\mathrm{Inf}_{\mathcal{R}}(C\{a\}) \mid a \in \mathrm{Inf}_{\mathcal{R}}(M)\} \ .\end{aligned}$$

(1) Application of continuity in a single hole context.

(2) It is obvious that we have "$\supset$". Because $\mathrm{Inf}_{\mathcal{R}}(M)$ is an ideal, we have that for every given combination of $a_i \in \mathrm{Inf}_{\mathcal{R}}(M)$ there exists an $a \in \mathrm{Inf}_{\mathcal{R}}(M)$, such that $a_i \leq_\omega a$. From monotonicity of $\mathrm{Inf}_{\mathcal{R}}$ with respect to $\leq_\Omega$ we get "$\subset$".

□

As a first step towards proving syntactic continuity of cyclic extensions we will now consider the extension of CRSs with let.

Definition 7.4.7 Given a CRS with information content $\mathcal{R}$, we define the extension of $\mathcal{R}$ with lets as follows:

$$\begin{aligned}\mathcal{R}_{\mathsf{let}} &= \mathcal{R} \cup \{\mathsf{let}\ x = M\ \mathsf{in}\ N \xrightarrow[\text{let}]{} N[x := M]\}\\ \omega_{\mathcal{R}_{\mathsf{let}}}(M) &= \omega_{\mathcal{R}}(\text{let}(M))\end{aligned}$$

An important characteristic of the let-extension is that it is syntactically continuous if the CRS is syntactically and substitutive continuous:

Lemma 7.4.8 Given an orthogonal CRS $\mathcal{R}$ with information content that is syntactically and substitutive continuous, we have that $\mathcal{R}_{\mathsf{let}}$ is syntactically and substitutive continuous.

Proof. Once we have proven syntactic continuity we can prove substitutive continuity in the same way as we did for Prop 7.4.4. Given a context C and a term M, both with lets, let $\vec{x}$ be a vector of variables such that $\{\vec{x}\} = FV(M)$. For any term Q, such that $FV(Q) \subset FV(M)$ we then have that

$$C[\mathsf{let}\ \vec{x} = \vec{x}\ \mathsf{in}\ Q] \xrightarrow[\text{let}]{}\!\!\!\!\rightarrow C'\{\mathsf{let}\ \vec{x} = \vec{P}\ \mathsf{in}\ Q\} \ ,$$

where C' contains no lets, by rewriting every let in C. We also have that

$$C'\{\mathsf{let}\ \vec{x} = \vec{P}\ \mathsf{in}\ Q\} \xrightarrow[\text{let}]{}\!\!\!\!\rightarrow C'\{\mathsf{let}\ \vec{x} = \vec{P}\ \mathsf{in}\ \text{let}(Q)\}$$

and that

$$C'\{\mathsf{let}\ \vec{x} = \vec{P}\ \mathsf{in}\ \text{let}(Q)\} \xrightarrow[\text{let}]{}\!\!\!\!\rightarrow C'\{\text{let}(Q)\underbrace{[x_1 := \text{let}(P_1), \cdots, x_n := \text{let}(P_n)]}_{\sigma}\} \ .$$

Because $C[\mathsf{let}\ \vec{x} = \vec{x}\ \mathsf{in}\ Q] \xrightarrow[\text{let}]{}\!\!\!\!\rightarrow C[Q]$, we can conclude that

$$\text{let}(C[\mathsf{let}\ \vec{x} = \vec{x}\ \mathsf{in}\ Q]) = C'\{\text{let}(Q)\sigma\} \ .$$

This implies that

$$\text{Inf}_{\mathcal{R}_{\text{let}}}(C[Q]) = \text{Inf}_{\mathcal{R}}(\text{let}(C[Q])) = \text{Inf}_{\mathcal{R}}(C'\{\text{let}(Q)\sigma\}) \ .$$

Using this identity we have that:

$$\begin{aligned}\text{Inf}_{\mathcal{R}_{\text{let}}}(C[M]) &= \text{Inf}_{\mathcal{R}}(C'\{\text{let}(M)\sigma\})\\ &= \cup\{\text{Inf}_{\mathcal{R}}(C'\{Q\}) \mid Q \in \text{Inf}_{\mathcal{R}}(\text{let}(M)\sigma)\}\\ &= \cup\{\text{Inf}_{\mathcal{R}}(C'\{Q\}) \mid Q \in \cup\{\text{Inf}_{\mathcal{R}}(P\sigma) \mid P \in \text{Inf}_{\mathcal{R}}(\text{let}(M)\}\\ &= \cup\{\text{Inf}_{\mathcal{R}}(C'\{Q\}) \mid Q \in \text{Inf}_{\mathcal{R}}(P\sigma), P \in \text{Inf}_{\mathcal{R}_{\text{let}}}(M)\}\\ &= \cup\{\cup\{\text{Inf}_{\mathcal{R}}(C'\{Q\}) \mid Q \in \text{Inf}_{\mathcal{R}}(P\sigma)\} \mid P \in \text{Inf}_{\mathcal{R}_{\text{let}}}(M)\}\\ &= \cup\{\text{Inf}_{\mathcal{R}}(C'\{P\sigma\}) \mid P \in \text{Inf}_{\mathcal{R}_{\text{let}}}(M)\}\\ &= \cup\{\text{Inf}_{\mathcal{R}_{\text{let}}}(C[P]) \mid P \in \text{Inf}_{\mathcal{R}_{\text{let}}}(M)\}\end{aligned}$$

□

Definition 7.4.9 Given an orthogonal CRS $\mathcal{R}$ with regular information content and a regular cyclic extension $\mathcal{R}^x$ of $\mathcal{R}$, we have

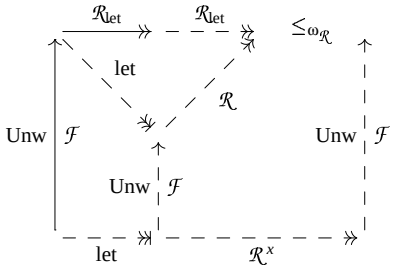
$$\mathcal{R}^x_{\text{let}} = \mathcal{R}^x \cup \xrightarrow[\text{let}]{} \quad .$$

Lemma 7.4.10 Given an orthogonal CRS $\mathcal{R}$ with regular information content and a regular cyclic extension $\mathcal{R}^x$ of $\mathcal{R}$, we have that $\mathcal{R}^x_{\text{let}}$ is a regular cyclic extension of $\mathcal{R}_{\text{let}}$ and that

$$\text{Inf}_{\mathcal{R}^x_{\text{let}}}(M) = \text{Inf}_{\mathcal{R}^x}(\text{let}(M)) \quad .$$

Proof. First, we have that $\mathcal{R}^x_{\text{let}}$ is infinitarily sound because it is given that $\mathcal{R}^x$ is infinitarily sound and because $\xrightarrow[\text{let}]{}$ is infinitarily sound because by Lemma 7.2.3.

Second we have that



Finally, we have that

$$\text{Inf}_{\mathcal{R}^x_{\text{let}}}(M) = \text{Inf}_{\mathcal{R}^x_{\text{let}}}(\text{let}(M)) = \text{Inf}_{\mathcal{R}^x}(\text{let}(M)) \quad .$$

□

We will now consider some special cases of $\text{Inf}_{\mathcal{R}^x}(C[M])$. The cases we want to consider correspond to the cases where free variables in M are not bound by letrecs in C. In these cases, we have the special property that

$$\text{Unw}(C[M]) =_{\downarrow} \{C'\{M'\} \mid C' \in \text{Unw}(C), M' \in \text{Unw}(M)\} \quad .$$

For contexts and terms with this property we can prove syntactic continuity:

Lemma 7.4.11 Given a regular cyclic extension $\mathcal{R}^x$ of a CRS with regular information content $\mathcal{R}$ containing an unwinding calculus unw, such that $\omega_{\mathcal{R}^x}$ is a notion of information content, we have that if $\mathcal{R}$ is syntactically continuous then for every context C and every term M, we have that

$$\begin{gathered}\text{Unw}(C[M]) =_{\downarrow} \{C'\{M'\} \mid C' \in \text{Unw}(C), M' \in \text{Unw}(M)\} \\ \Longrightarrow \\ \text{Inf}_{\mathcal{R}^x}(C[M]) = \{\text{Inf}_{\mathcal{R}^x}(C[P]) \mid P \in \text{Inf}_{\mathcal{R}^x}(M)\} \quad .\end{gathered}$$

Proof.

$$
\begin{array}{ll}
\mathrm{Inf}_{\mathcal{R}^x}(C[M]) & \\
\quad = \mathrm{Inf}^{\infty}_{\mathcal{R}}(\mathrm{Unw}(C[M])) & 7.3.6 \\
\quad = \cup\{\mathrm{Inf}_{\mathcal{R}}(P) \mid P \in \mathrm{Unw}(C[M])\} & \\
\quad = \cup\{\mathrm{Inf}_{\mathcal{R}}(P) \mid P \in{\downarrow} \{C'\{Q\} \mid C' \in \mathrm{Unw}(C), Q \in \mathrm{Unw}(M)\}\} & \\
\quad = \cup\{\mathrm{Inf}_{\mathcal{R}}(P) \mid P \in \{C'\{Q\} \mid C' \in \mathrm{Unw}(C), Q \in \mathrm{Unw}(M)\}\} & \\
\quad = \cup\{\mathrm{Inf}_{\mathcal{R}}(C'\{Q\}) \mid C' \in \mathrm{Unw}(C), Q \in \mathrm{Unw}(M)\} & \\
\quad = \cup\{\cup\{\mathrm{Inf}_{\mathcal{R}}(C'\{a\}) \mid a \in \mathrm{Inf}_{\mathcal{R}}(Q)\} \mid C' \in \mathrm{Unw}(C), Q \in \mathrm{Unw}(M)\} & 7.4.6 \\
\quad = \cup\{\mathrm{Inf}_{\mathcal{R}}(C'\{a\}) \mid C' \in \mathrm{Unw}(C), a \in \cup\{\mathrm{Inf}_{\mathcal{R}}(Q) \mid Q \in \mathrm{Unw}(M)\}\} & \\
\quad = \cup\{\mathrm{Inf}_{\mathcal{R}}(C'\{a\}) \mid C' \in \mathrm{Unw}(C), a \in \mathrm{Inf}^{\infty}_{\mathcal{R}}(M)\} & \\
\quad = \cup\{\mathrm{Inf}_{\mathcal{R}}(C'\{a\}) \mid C' \in \mathrm{Unw}(C), a \in \mathrm{Inf}_{\mathcal{R}^x}(M)\} & 7.3.6 \\
\quad = \cup\{\mathrm{Inf}_{\mathcal{R}}(a') \mid a' \in \{C'\{a\} \mid C' \in \mathrm{Unw}(C)\}, a \in \mathrm{Inf}_{\mathcal{R}^x}(M)\} & \\
\quad = \cup\{\mathrm{Inf}_{\mathcal{R}}(a') \mid a' \in{\downarrow} \{C'\{a\} \mid C' \in \mathrm{Unw}(C)\}, a \in \mathrm{Inf}_{\mathcal{R}^x}(M)\} & \\
\quad = \cup\{\mathrm{Inf}_{\mathcal{R}}(a') \mid a' \in \mathrm{Unw}(C[a]), a \in \mathrm{Inf}_{\mathcal{R}^x}(M)\} & \\
\quad = \cup\{\cup\{\mathrm{Inf}_{\mathcal{R}}(a') \mid a' \in \mathrm{Unw}(C[a])\} \mid a \in \mathrm{Inf}_{\mathcal{R}^x}(M)\} & \\
\quad = \cup\{\mathrm{Inf}^{\infty}_{\mathcal{R}}(\mathrm{Unw}(C[a])) \mid a \in \mathrm{Inf}_{\mathcal{R}^x}(M)\} & \\
\quad = \cup\{\mathrm{Inf}_{\mathcal{R}^x}(\mathrm{Unw}(C[a])) \mid a \in \mathrm{Inf}_{\mathcal{R}^x}(M)\} & 7.3.6
\end{array}
$$

□

We now have all the lemmas we need to prove syntactic continuity of cyclic extensions.

Theorem 7.4.12 Given a regular cyclic extension $\mathcal{R}^x$ of an orthogonal CRS with regular information content $\mathcal{R}$ containing an unwinding calculus unw, such that $\omega_{\mathcal{R}^x}$ is a notion of information content, we have that if $\mathcal{R}$ is syntactically and substitutive continuous then $\mathcal{R}^x$ is syntactically continuous.

Proof. Given a context C and a term M, let $\vec{x}$ be a vector of variables such that $\{\vec{x}\} = FV(M)$. We then have that

$$
\begin{array}{ll}
\mathrm{Inf}_{\mathcal{R}^x}(C[M]) & \\
\quad = \mathrm{Inf}_{\mathcal{R}^x_{\mathsf{let}}}(C[\mathsf{let}\ \vec{x} = \vec{x}\ \mathsf{in}\ M]) & \\
\quad = \cup\{\mathrm{Inf}_{\mathcal{R}^x_{\mathsf{let}}}(C[\mathsf{let}\ \vec{x} = \vec{x}\ \mathsf{in}\ Q]) \mid Q \in \mathrm{Inf}_{\mathcal{R}^x_{\mathsf{let}}}(M)\} & \text{Lemma 7.4.11} \\
\quad = \cup\{\mathrm{Inf}_{\mathcal{R}^x_{\mathsf{let}}}(C[Q]) \mid Q \in \mathrm{Inf}_{\mathcal{R}^x_{\mathsf{let}}}(M)\} & \\
\quad = \cup\{\mathrm{Inf}_{\mathcal{R}^x}(C[Q]) \mid Q \in \mathrm{Inf}_{\mathcal{R}^x}(M)\} &
\end{array}
$$

Note that in this proof we use the let as a function symbol and not as the syntactic abbreviation of a letrec. □

7.5 Conclusion

We have given an extension theory that works fine for $\leq_\Omega$-monotonic CRSs, we do not yet know much about CRSs that aren't. In many cases we depend on $\leq_\Omega$-monotonicity in key parts of proofs. Thus, it will be hard to either modify the

theory in a way that allows these non-monotonic systems to work. It may even be necessary to develop a completely new framework to deal with them.

Another possible direction of future work is extending acyclic calculi rather than CRSs. This could be useful because given a cyclic calculus, its acyclic fragment may have better properties than the entire calculus. For example, the cyclic lambda calculus is non-confluent, but its acyclic fragment is in essence confluent by complete developments.

In the next chapter we consider a few applications of the extension theory.

Chapter 8

Applications

In this chapter we will consider applications of the theory of semantics of graph rewriting in the previous chapter. We will consider orthogonal CRSs in general and cyclic lambda calculi in particular, but first we will give some more examples of unwinding calculi.

8.1 Unwinding Calculi

The only examples, we have seen of unwinding calculi were $\mu\circ$ and some calculi with Unw as information content. For example, the two rule rewrite system

$$\begin{array}{ll}\langle C[x] \mid x = M, D\rangle \xrightarrow[\text{es}]{} \langle C[M] \mid x = M, D\rangle & ; \\ \langle \lambda x.M \mid D\rangle N \qquad \xrightarrow[\text{lift}]{} \langle (\lambda x.M)\, N \mid D\rangle & .\end{array}$$

We want to have a notion of information content for this rewrite system that is finite. The notion $\omega_{\mu\circ}$ does not work, because there is a fundamental difference between this simple two rule system and $\mu\circ$. Both systems move letrec around in terms. The two rule system *lifts* letrecs towards the top of the term. In contrast $\mu\circ$ *distributes* the letrec away from the top of the term. These two systems and many others build up a tree-like prefix in a term. A system that distributes the letrecs exposes this tree like prefix very clearly. A system that lifts letrecs keeps the prefix more or less hidden. For example, we have that

$$\begin{array}{ll}\langle x \mid x = F(x)\rangle & \xrightarrow[\mu\circ]{} \langle F(x) \mid x = F(x)\rangle \\ & \xrightarrow[\mu\circ]{} F(\langle F(x) \mid x = F(x)\rangle) \\ & \xrightarrow[\mu\circ]{} F(F(\langle F(x) \mid x = F(x)\rangle)) \\ & \xrightarrow[\mu\circ]{} \cdots\end{array}$$

and that

$$\begin{array}{ll}\langle x \mid x = F(x)\rangle & \xrightarrow[\text{es}]{} \langle F(x) \mid x = F(x)\rangle \\ & \xrightarrow[\text{es}]{} \langle F(F(x)) \mid x = F(x)\rangle \\ & \xrightarrow[\text{es}]{} \langle F(F(F(x))) \mid x = F(x)\rangle \\ & \xrightarrow[\text{es}]{} \cdots \quad .\end{array}$$

Table 8.1 The rewrite rules of the lift calculus (lc)

$\langle C[x] \mid x = M, D\rangle$	$\xrightarrow[\text{es}]{}$	$\langle C[M] \mid x = M, D\rangle$	
$\langle M \mid x = C[y], y = N, D\rangle$	$\xrightarrow[\text{is}]{}$	$\langle M \mid x = C[N], y = N, D\rangle$	
$\langle M \mid x = C[x], D\rangle$	$\xrightarrow[\text{cs}]{}$	$\langle M \mid x = C[C[x]], D\rangle$	
$F(\cdots, \langle M \mid D\rangle, \cdots)$	$\xrightarrow[\text{F-lift}]{}$	$\langle F(\cdots, M, \cdots) \mid D\rangle$	
$[x]\langle M \mid D_1, D_2\rangle$	$\xrightarrow[\lambda\text{-lift}]{}$	$\langle [x]M \mid D_1\rangle \mid D_2\rangle$	
$\langle\langle M \mid D_1\rangle \mid D_2\rangle$	$\xrightarrow[\text{em}]{}$	$\langle M \mid D_1, D_2\rangle$	
$\langle M \mid x = \langle N \mid D_1\rangle, D_2\rangle$	$\xrightarrow[\text{im}]{}$	$\langle M \mid x = N, D_1, D_2\rangle$	
$\langle M \mid D_1, D_2\rangle$	$\xrightarrow[\text{gc}]{}$	$\langle M \mid D_1\rangle$	
$\langle M \mid\rangle$	$\xrightarrow[\text{gc}]{}$	M	
M	$\xrightarrow[\text{bisim}]{}$	N	, if $M \underline{\leftrightarrow} N$

In this case the same prefix is being built, but in the second sequence it stays hidden in the external part of the letrec. We expose the information in the external part by defining the information content as follows:

Definition 8.1.1 We define the function ω_{lift} by means of the rewrite system

$$\langle M \mid x_1 = M_1, \cdots, x_n = M_n\rangle \xrightarrow[\omega_{\text{lift}}]{} M[x_1 := \Omega, \cdots, x_n := \Omega] \ .$$

Later in this chapter, we will give several examples of rewrite systems with information content ω_{lift} are unwinding calculi. First, we will study a large superset of this calculus: the lift calculus. An important property of the lift calculus, given in Table 8.1, is that its convertibility relation identifies precisely those terms that have the same unwinding. The notion of information content that we use for the lift calculus is ω_{lift}. The proof of the fact that the lift calculus is an unwinding calculus is complicated by the fact it is not confluent. For example, the terms on the bottom of the following diagram do not have a reduct in common:

$$\begin{array}{ccc} \langle x \mid x = F(x)\rangle & \longrightarrow & \langle F(x) \mid x = F(x)\rangle \\ \downarrow & & \downarrow \\ \langle x \mid x = F(F(x))\rangle & & \langle F(x) \mid x = F(F(x))\rangle \end{array}$$

To show that the lift calculus is an unwinding calculus we will use yet another reduction relation and some lemmas. The reduction relation is a restriction of the external substitution rule in the lift calculus. We refer to this reduction relation as *standard reduction*: Standard reduction is a restriction of the application of external substitution to certain contexts. The redex that are allowed are those that do not occur in the definition of a variable:

Definition 8.1.2 Standard external substitution reduction is defined by

$$E_1[\langle E_2[x] \mid x = M, D\rangle] \underset{\text{es}}{\longmapsto} E_1[\langle E_2[M] \mid x = M, D\rangle] \ ,$$

where

$$E ::= \Box \mid F(\cdots, E, \cdots) \mid [x].E \mid \langle E \mid D\rangle \ .$$

Because we will have to distinguish between standard and non-standard reduction, we introduce the notation $(\circ\!\underset{\text{es}}{\longrightarrow})$ for non-standard reduction:

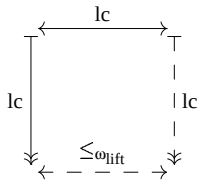
$$\circ\!\underset{\text{es}}{\longrightarrow} = \underset{\text{es}}{\longrightarrow} \setminus \underset{\text{es}}{\longmapsto} \ .$$

A few examples of standard and non-standard steps are:

$$\begin{array}{rcl}
\langle F(x,x) \mid x = \langle y \mid y = z\rangle\rangle & \underset{\text{es}}{\longmapsto} & \langle F(x, \langle y \mid y = z\rangle) \mid x = \langle y \mid y = z\rangle\rangle \\
\langle F(x,x) \mid x = \langle y \mid y = z\rangle\rangle & \underset{\text{es}}{\longmapsto} & \langle F(\langle y \mid y = z\rangle, x) \mid x = \langle y \mid y = z\rangle\rangle \\
\langle x \mid x = \langle y \mid y = z\rangle\rangle & \circ\!\underset{\text{es}}{\longrightarrow} & \langle x \mid x = \langle z \mid y = z\rangle\rangle
\end{array}$$

The most complicated step in showing that the lift calculus is an unwinding calculus is the following lemma:

Lemma 8.1.3 We have the following diagram:



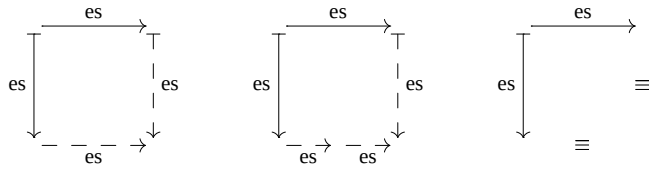
Proof. We will prove the diagram by using a tiling argument, but first we eliminate internal and cyclic substitution steps by replacing them with alternative conversions:

$$\begin{array}{rcl}
\langle M \mid x = C[y], y = N, D\rangle & \underset{\text{bisim}}{\longrightarrow} & \langle M \mid x = C[y'], y' = N, y = N, D\rangle \\
& \underset{\text{im}}{\longleftarrow} & \langle M \mid x = \langle C[y'] \mid y' = N\rangle, y = N, D\rangle \\
& \underset{\text{es}}{\longrightarrow} & \langle M \mid x = \langle C[N] \mid y' = N\rangle, y = N, D\rangle \\
& \underset{\text{gc}}{\longrightarrow} & \langle M \mid x = \langle C[N] \mid\rangle, y = N, D\rangle \\
& \underset{\text{gc}}{\longrightarrow} & \langle M \mid x = C[N], y = N, D\rangle
\end{array}$$

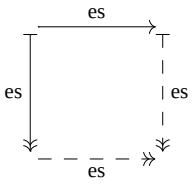
$$\begin{array}{rcl}
\langle M \mid x = C[x], D\rangle & \underset{\text{bisim}}{\longrightarrow} & \langle M \mid x = C[y], y = C[x], D\rangle \\
& \underset{\text{im}}{\longleftarrow} & \langle M \mid \langle x = C[y] \mid y = C[x]\rangle, D\rangle \\
& \underset{\text{es}}{\longrightarrow} & \langle M \mid \langle x = C[C[x]] \mid y = C[x]\rangle, D\rangle \\
& \underset{\text{im}}{\longrightarrow} & \langle M \mid x = C[C[x]], y = C[x], D\rangle \\
& \underset{\text{gc}}{\longrightarrow} & \langle M \mid x = C[C[x]], D\rangle
\end{array}$$

We will now distinguish three cases for the top step: external substitution steps from left to right external substitution steps from right to left and other steps, denoted $^{\neg}$s, in both horizontal directions:

$\xrightarrow[es]{}$ External substitution is confluent by complete developments of descendants. Duplication in elementary diagrams is limited: an arbitrary step can have one or two descendants and a standard step can only have one descendant. Therefore, these are the only elementary diagrams:

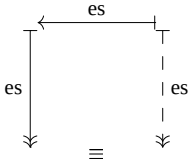


From these diagrams we can conclude that



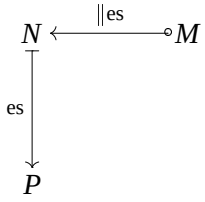
(8.1)

$\xleftarrow[es]{}$ If the given step is a standard step then we have



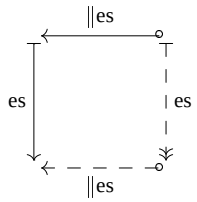
(8.2)

To deal with the case of a non-standard step we will use complete developments of nonstandard steps denoted with $\circ\!\xrightarrow[\|es]{}$. Consider the diagram

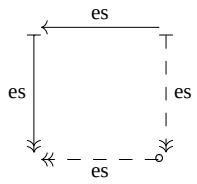


A non-standard step cannot create a standard step, so the redex contracted in N is a descendant of a redex in M. Moreover, a non-standard step cannot duplicate a standard redex. Thus, the redex contracted in N is the only descendant of the redex in M. Hence, the given reduction sequence is a complete development. We can also do a complete development of the same set of redexes, by first contracting all standard redexes and then developing the

set of the remaining non-standard redexes:

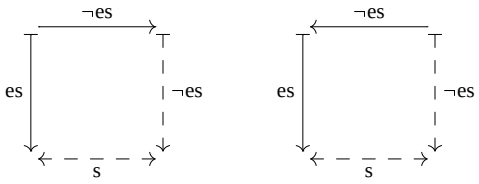


From this diagram and 8.2 we can derive:

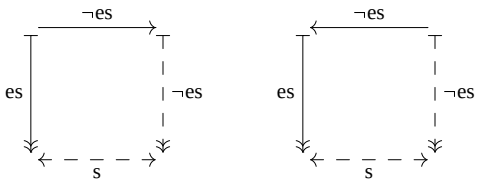


(8.3)

$\xrightarrow[\neg s]{}, \xleftarrow[\neg s]{}$ A lengthy case analysis will show that we have the following two diagrams:



From these diagram we may conclude that



(8.4)

Because $\xrightarrow[es]{}$ does not decrease the information content and $\circ\!\!\xrightarrow[es]{}$ and $\xrightarrow[\neg s]{}$ do not change the information content, the result follows from 8.1, 8.3 and 8.4. □

In the following proposition we prove that the infinite normal forms computed with respect to $\longmapsto\!\!\!\!\xrightarrow[es]{}$ are preserved by $\xrightarrow[lc]{}$ and that the infinite normal form computed with respect to the $\longmapsto\!\!\!\!\xrightarrow[es]{}$ and $\xrightarrow[lc]{}$ are the same.

Proposition 8.1.4 We have that

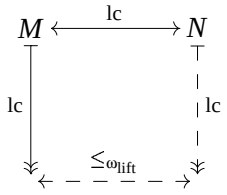
(i) $M \xleftrightarrow[lc]{} N \implies \mathrm{Inf}_{\longmapsto\!\!\!\!\xrightarrow[es]{}}(M) = \mathrm{Inf}_{\longmapsto\!\!\!\!\xrightarrow[es]{}}(N)$;

(ii) $\mathrm{Inf}_{\longmapsto\!\!\!\!\xrightarrow[es]{}}(M) = \mathrm{Inf}_{\xrightarrow[lc]{}}(M)$.

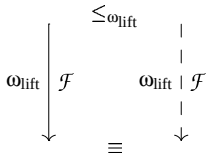
Proof. Due to symmetry it is sufficient to show that given $M \xleftarrow[\text{lc}]{} N$, we have that

$$\text{Inf}_{\xrightarrow[\text{es}]{}}(M) \subset \text{Inf}_{\xrightarrow[\text{es}]{}}(N) \ .$$

From Lemma 8.1.3 we get:



We also have

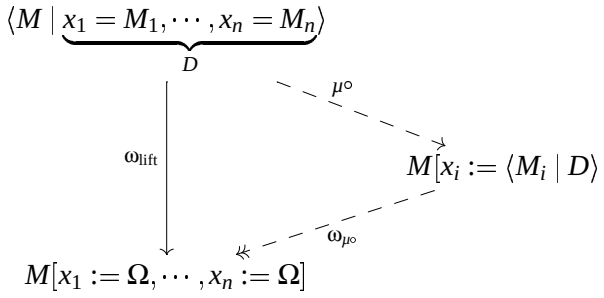


The composition of these two diagrams proves the result. □

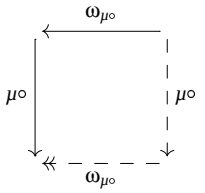
Because $\xrightarrow[\text{es}]{}$ is confluent, its infinite normal forms are unique. From this and the previous proposition it follows that the infinite normal forms of $\xrightarrow[\text{lc}]{}$ are unique. We are now almost ready to compare the infinite normal forms of $\mu\circ$ and the lift calculus. The only thing we need is a comparison between the notions of information content $\omega_{\mu\circ}$ and ω_{lift} used in those calculi:

Proposition 8.1.5 Given a term M, there exists a term N, such that $M \xrightarrow[\mu\circ]{} N$ and $\omega_{\text{lift}}(M) = \omega_{\mu\circ}(N)$.

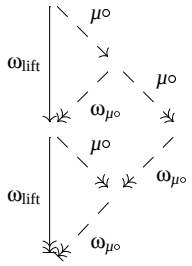
Proof. We have that



We also have that



Hence we can prove the result by induction on the number of ω_{lift} steps that are necessary to rewrite M to $\omega_{\text{lift}}(M)$:



□

This completes the preliminaries for the main theorem of this section:

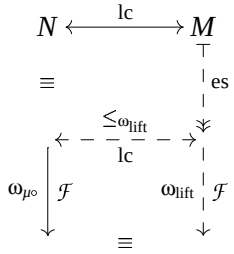
Theorem 8.1.6 The lift calculus is an unwinding calculus.

Proof. We have to show that

$$\text{Inf}_{\mu\circ}(M) = \text{Inf}_{\text{lc}}(M) \ .$$

We must distinguish two cases:

"$\subset$" Given $M \xrightarrow[\mu\circ]{}\!\!\!\!\rightarrow N \xrightarrow[\omega_{\mu\circ}]{\mathcal{F}} a$, we can prove that $M \leftarrow\!\!\!\!\rightarrow N$. We then have the following diagram:



The top halve follows from Lemma 8.1.3 the bottom half follows from the fact that for every term P we have that $\omega_{\mu\circ}(P) \leq_\Omega \omega_{\text{lift}}(P)$.

"$\supset$" By Prop. 8.1.4 it suffices to show that

$$\text{Inf}_{\xrightarrow[\text{es}]{}}(M) \subset \text{Inf}_{\mu\circ}(M) \ .$$

Given $M \xrightarrow[\text{es}]{}\!\!\!\!\rightarrow N \xrightarrow[\omega_{\text{lift}}]{\mathcal{F}} a$, we can prove that $M \xleftarrow[\mu\circ]{}\!\!\!\!\rightarrow N$. By Prop. 8.1.5 we have that $N \xrightarrow[\mu\circ]{}\!\!\!\!\rightarrow \xrightarrow[\omega_{\mu\circ}]{\mathcal{F}} a$. From the uniqueness of $\text{Inf}_{\mu\circ}$ it follows that $M \xrightarrow[\mu\circ]{}\!\!\!\!\rightarrow \xrightarrow[\omega_{\mu\circ}]{\mathcal{F}} a$.

□

As a corollary we have that many other rewrite system with ω_{lift} as information content are unwinding calculi:

Table 8.2 The value unwinding calculus ($\mathrm{unw_V}$)

Rewrite system:

$$\begin{array}{lll} \langle C[x] \mid x = V, D\rangle & \xrightarrow[\mathrm{es_V}]{} & \langle C[V] \mid x = V, D\rangle \\ \langle M \mid y = C[x], x = V, D\rangle & \xrightarrow[\mathrm{is_V}]{} & \langle M \mid y = C[V], x = V, D\rangle \\ \langle M \mid x = \langle N \mid D_1\rangle, D_2\rangle & \xrightarrow[\mathrm{im}]{} & \langle M \mid x = N, D_1, D_2\rangle \end{array}$$

where

$$V ::= x \mid \lambda x.M \ .$$

Information content:

$$\omega_V(M) = \mathrm{lub}\{a \mid M \xrightarrow[\mathrm{es_@}]{}\!\!\!\!\!\rightarrow \xrightarrow[\omega_{\mathrm{lift}}]{\mathcal{F}} a\} \ ,$$

where

$$\langle C[x] \mid x = MN, D\rangle \xrightarrow[\mathrm{es_@}]{} \langle C[MN] \mid x = MN, D\rangle \ .$$

Corollary 8.1.7 Every rewrite system $\xrightarrow[\mathrm{unw}]{}$ with information content ω_{lift}, such that $\xrightarrow[\mathrm{es}]{} \subset \xrightarrow[\mathrm{unw}]{} \subset \xrightarrow[\mathrm{lc}]{}$, is an unwinding calculus.

Proof. It is easy to show that

$$\mathrm{Inf}_{\xrightarrow[\mathrm{es}]{}}(M) \subset \mathrm{Inf}_{\xrightarrow[\mathrm{unw}]{}}(M) \subset \mathrm{Inf}_{\xrightarrow[\mathrm{lc}]{}}(M) \ .$$

The rest follows from Prop. 8.1.4. □

As a result of this corollary, we have that $\xrightarrow[\mathrm{es}]{}$ and $\xrightarrow[\mathrm{es,lift}]{}$ are unwinding calculi. We will now continue with an unwinding calculus for cyclic lambda terms.

Some existing lambda calculi with let or letrec restrict substitution to variables and lambda abstractions. The idea is that by only copying these so called *values* there will be no unnecessary duplication of redexes. In Table 8.2 we have given the value unwinding calculus. This calculus is a rewrite system with information content that obeys the restriction on substitution. We will now show that it is an unwinding calculus.

Theorem 8.1.8 The value unwinding calculus is an unwinding calculus.

Proof. We have that

$$M \xrightarrow[\omega_V]{\mathcal{F}} a \iff M \xrightarrow[\mathrm{es_@}]{}\!\!\!\!\!\rightarrow \xrightarrow[\omega_{\mathrm{lift}}]{\mathcal{F}} a \ .$$

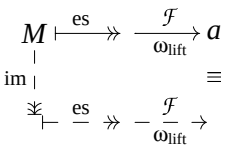
Hence it is easy to show that

$$\mathrm{Inf}_{\omega_V}(M) \subset \mathrm{Inf}_{\mathrm{lc}}(M) \ .$$

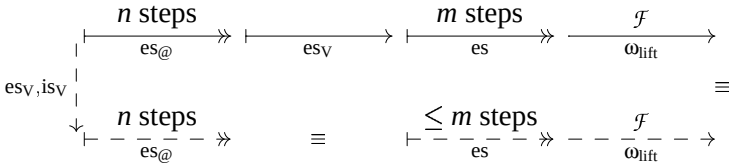
We will now show that

$$\mathrm{Inf}_{\underset{\mathrm{es}}{\longrightarrow}}(M) \subset \mathrm{Inf}_{\omega_V}(M) \ .$$

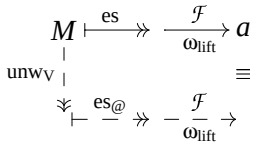
Given $M \underset{\mathrm{es}}{\longmapsto\!\!\!\!\!\rightarrow} \xrightarrow[\omega_{\mathrm{lift}}]{\mathcal{F}} a$, we have that



We also have that



Note that the standard sequence on the bottom is shorter than that at the top, by starting with the previous diagram and tiling with the second diagram as long as possible we get



□

8.2 Standard information content for CRSs

In this section we define a notion of information content that is applicable to all orthogonal CRSs.

Definition 8.2.1 Given an orthogonal CRS $\mathcal{R}$, we define the rewrite relation $\omega_{\mathcal{R}}$ by:

$$M \xrightarrow[\omega_{\mathcal{R}}]{} \Omega, \text{ if } \Omega <_\Omega M \leq_\Omega N \text{ and } N \text{ is a redex.}$$

Proposition 8.2.2 Given an orthogonal CRS $\mathcal{R}$, the rewrite relation $\omega_{\mathcal{R}}$ is confluent and terminating.

Proof.

- Termination follows from the fact that $\xrightarrow[\mathcal{R}]{}$ decreases the number of function symbols other than Ω in a term.

- Given termination, confluence follows from weak confluence. To prove weak confluence we must consider a term with two redexes. The two redex can be either disjoint or one nested below the other. In both cases we can easily find common reducts:

$$\begin{array}{ccc} C[M_1,M_2] & \longrightarrow & C[M_1,\Omega] \\ \downarrow & & \downarrow \\ C[\Omega,M_2] & \longrightarrow & C[\Omega,\Omega] \end{array} \qquad \begin{array}{ccc} C_1[C_2[M]] & \longrightarrow & C_1[C_2[\Omega]] \\ \downarrow & \swarrow & \\ C_1[\Omega] & & \end{array}$$

□

Proposition 8.2.3 Given an orthogonal CRS $\mathcal{R}$, we have that

$$M \leq_\Omega N \implies \omega_\mathcal{R}(M) \leq_\Omega \omega_\mathcal{R}(N) \ .$$

Proof. If $M \xrightarrow[\omega_\mathcal{R}]{} M'$ then $M' \leq_\Omega M$. Hence we have that

$$\omega_\mathcal{R}(M) \leq_\Omega M \ . \tag{8.5}$$

Given $\omega(M) \leq_\Omega N$ and $N \xrightarrow[\omega_\mathcal{R}]{} N'$, we have that $N \equiv C[P] \xrightarrow[\omega_\mathcal{R}]{} C[\Omega] \equiv N'$, because there exists a term $P' \geq_\Omega P$, which is a redex. If we do not have that $\omega_\mathcal{R}(M) \leq_\Omega N'$ then we must have that $\omega_\mathcal{R}(M) \equiv C'[Q]$, where the holes in C and C' occur in exactly the same position and $Q \not\equiv \Omega$. Because $\omega_\mathcal{R}(M) \leq_\Omega N$, we have that $Q \leq_\Omega P$. This implies that $Q \leq_\Omega P'$, which in turn implies that $C'[Q] \xrightarrow[\omega_\mathcal{R}]{} C'[\Omega]$. This is in contradiction with the fact that $\omega_\mathcal{R}(M)$ is in $\omega_\mathcal{R}$-normal form. Hence, we have that

$$\omega_\mathcal{R}(M) \leq_\Omega N, N \xrightarrow[\omega_\mathcal{R}]{} N' \implies \omega_\mathcal{R}(M) \leq_\Omega N' \ .$$

From this implication and the termination of $\omega_\mathcal{R}$, we may conclude that

$$\omega_\mathcal{R}(M) \leq_\Omega N \implies \omega_\mathcal{R}(M) \leq_\Omega \omega_\mathcal{R}(N) \ . \tag{8.6}$$

By 8.5 we have that $M \leq_\Omega N$ implies that $\omega_\mathcal{R}(M) \leq_\Omega N$. Hence, we may derive from 8.6 that

$$M \leq_\Omega N \implies \omega_\mathcal{R}(M) \leq_\Omega \omega_\mathcal{R}(N) \ .$$

□

Given $C[M] \xrightarrow[\mathcal{R}]{} C[N]$, we have that $\omega_\mathcal{R}(C[M]) = \omega_\mathcal{R}(C[\Omega])$. Because $C[\Omega] \leq_\Omega C[N]$, we have that $\omega_\mathcal{R}(C[\Omega]) \leq_\Omega \omega_\mathcal{R}(C[N])$. This implies that $\omega_\mathcal{R}(C[M]) \leq_\Omega \omega_\mathcal{R}(C[N])$. From this we can conclude that $\xrightarrow[\mathcal{R}]{}$ is monotonic with respect to $\Omega_\mathcal{R}$. Therefore, we can now define the standard information content of a CRS as follows:

Definition 8.2.4 Given an orthogonal CRS $\mathcal{R}$, we define the the CRS with standard information content $\mathcal{R}$ as $(\mathcal{R}, \Omega_\mathcal{R}, (\mathsf{Terms}^\infty_\mathcal{R}, \leq_\Omega))$.

The standard information content is useful in the theory of programming languages. For example, in programming languages the Lévy-Longo tree is used with the lambda calculus instead of the Böhm tree. The information content corresponding to the Lévy-Longo tree is given by

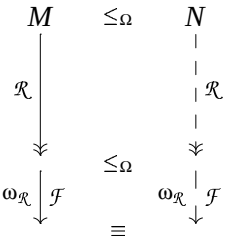
$$\begin{array}{ll}(\lambda x.M)\,N & \to \Omega \\ \Omega\,M & \to \Omega\end{array}$$

This is precisely the standard information content of the lambda calculus with β-reduction.

We will now prove some properties of CRSs with standard information content. From the previous chapter is has become clear that monotonicity of $\mathrm{Inf}_{\mathcal{R}}$ is a desirable property. It is an open question whether this property holds for every orthogonal CRS with standard information content. However, for $\leq_\Omega$-monotonic CRSs this property is easy to prove:

Proposition 8.2.5 Given an $\leq_\Omega$-monotonic orthogonal CRS $\mathcal{R}$ with standard information content $\mathcal{R}$, we have that $\mathrm{Inf}_{\mathcal{R}}$ is monotonic with respect to $\leq_\Omega$.

Proof. Given $M \leq_\Omega N$, we have the following diagram:



Hence, we have that

$$\mathrm{Inf}_{\mathcal{R}}(M) = \{a \mid M \xrightarrow[\mathcal{R}]{}\!\!\!\!\!\rightarrow\; \xrightarrow[\omega_{\mathcal{R}}]{\mathcal{F}} a\} \subset \{a \mid N \xrightarrow[\mathcal{R}]{}\!\!\!\!\!\rightarrow\; \xrightarrow[\omega_{\mathcal{R}}]{\mathcal{F}} a\} = \mathrm{Inf}_{\mathcal{R}}(N) \quad .$$

□

We will now consider syntactic and substitutive continuity of CRSs with standard information content. To prove syntactic continuity we will follow the proof strategy of Lévy ([Lév78]). This strategy requires introducing the notion of reducing a term without touching redexes descending from a certain subterm:

Definition 8.2.6 In an orthogonal CRS, we write

$$C[M] \xrightarrow[M]{}\!\!\!\!\!\rightarrow N \quad ,$$

if $C[M]$ reduces to N without reducing a descendant of a redex in M.

An arbitrary reduction from $C[M]$ to a term N may contract many redexes which are descendants from redexes in M. However, it is possible to reduce M to a term M', such that a common reduct N' of $C[M']$ and N can be found in such a way that the reduction form $C[M']$ to N' does not contract descendants from redexes in M':

Lemma 8.2.7 In any orthogonal CRS we have the diagram:

$$\begin{array}{ccc} C[M] & \twoheadrightarrow & N \\ \vdots & & \vdots \\ C[M'] & -\!\!\!\nearrow\!\!\!-\!\!\!\!\!\!\twoheadrightarrow_{M'} & N' \end{array}$$

Proof. We can view the given sequence $C[M] \twoheadrightarrow N$ as a sequence

$$C[M] \xrightarrow[S_1]{} \xrightarrow[S_2]{} \cdots \xrightarrow[S_n]{} N \ ,$$

where each step $\xrightarrow[S_i]{}$ stands for the complete development of the set of redexes S_i.

Given a sequence of complete developments

$$C[M] \xrightarrow[S_1]{} \xrightarrow[S_2]{} \cdots \xrightarrow[S_n]{} N \ ,$$

let S_i be the first set of redexes which contains a descendant from a redex in M. Let S be the set of redexes in M, whose descendants are in S_i. We then have:

$$\begin{array}{ccccccc} C[M] & \xrightarrow[S_1]{} & \cdots & \xrightarrow[S_i]{} & \cdots & \xrightarrow[S_n]{} & N \\ {\scriptstyle S}\downarrow & & & & & & \vdots \\ C[M'] & \xrightarrow[S'_1]{} & \cdots & -\!\!\!\rightarrow_{S'_i} & \cdots & -\!\!\!\rightarrow_{S_n} & N' \end{array}$$

In this diagram we have that $S'_1, \cdots, S'_i$ do not contain descendants form redexes in M', so in a finite number of iterations we will finish with the diagram we needed to prove. □

Theorem 8.2.8 An $\leq_\Omega$-monotonic orthogonal CRS with standard information content is syntactic continuous.

Proof. We have to show that

$$\mathrm{Inf}(C[M]) = \cup\{\mathrm{Inf}(C[P]) \mid P \in \mathrm{Inf}(M)\}$$

We will distinguish two cases

"$\supset$" Given $P \in \mathrm{Inf}(M)$, we have to show that $\mathrm{Inf}(C[P]) \subset \mathrm{Inf}(C[M])$.

Because $P \in \mathrm{Inf}(M)$, we have that $M \twoheadrightarrow N \xrightarrow[\omega]{\mathcal{F}} P$. This implies that $P \leq_\omega N$. From this it follows that $C[P] \leq_\Omega C[N]$ so by monotonicity and uniqueness of Inf we have:

$$\mathrm{Inf}(C[P]) \subset \mathrm{Inf}(C[N]) = \mathrm{Inf}(C[M]) \ .$$

"$\subset$" Given $C[M]\twoheadrightarrow N$, we will show that there exists a $P \in \mathrm{Inf}(M)$, such that $C[P]\twoheadrightarrow N'$ and $N \leq_\omega N'$.

By Lemma 8.2.7 we have that $C[M]\twoheadrightarrow C[M'] \overset{\nearrow}{\underset{M'}{\twoheadrightarrow}} N''$, with $N \twoheadrightarrow N''$. We have that

$$\begin{array}{ccc} C[Q] & \overset{\nearrow}{\underset{Q}{\twoheadrightarrow}} & R \\ {\scriptstyle\omega}\downarrow & & \vdots\,{\scriptstyle\omega} \\ C[Q'] & \overset{\nearrow}{\underset{Q'}{\twoheadrightarrow}} & R' \end{array}$$

If we define $P = \omega(M')$ then we have that

$$\begin{array}{ccc} C[M'] & \overset{\nearrow}{\underset{M'}{\twoheadrightarrow}} & N'' \\ {\scriptstyle\omega}\,\vdots & & \vdots\,{\scriptstyle\omega} \\ C[P] & \twoheadrightarrow & N' \end{array}$$

By monotonicity of ω it follows from $N \twoheadrightarrow N''$ that $N \leq_\omega N''$. Because $\omega(N'') = \omega(N')$ we have that

$$N \leq_\omega N' \quad .$$

□

Theorem 8.2.9 An $\leq_\Omega$-monotonic orthogonal CRS with standard information content $\mathcal{R}$ is substitutive continuous.

Proof. In this proof we use $\mathcal{R}_{\mathsf{let}}$, the extension of $\mathcal{R}$ with let. In the previous chapter we already defined a notion of information content for $\mathcal{R}_{\mathsf{let}}$, but in this proof we will use standard information content for $\mathcal{R}_{\mathsf{let}}$. With this extension we have that:

$$\begin{aligned} \mathrm{Inf}_{\mathcal{R}}(M\sigma) &= \mathrm{Inf}_{\mathcal{R}_{\mathsf{let}}}(M\sigma) \\ &= \mathrm{Inf}_{\mathcal{R}_{\mathsf{let}}}(\mathsf{let}\ \sigma\ \mathsf{in}\ M) \\ &= \cup\{\mathrm{Inf}_{\mathcal{R}_{\mathsf{let}}}(\mathsf{let}\ \sigma\ \mathsf{in}\ P) \mid P \in \mathrm{Inf}_{\mathcal{R}_{\mathsf{let}}}(M)\} \\ &= \cup\{\mathrm{Inf}_{\mathcal{R}_{\mathsf{let}}}(P\sigma) \mid P \in \mathrm{Inf}_{\mathcal{R}_{\mathsf{let}}}(M)\} \\ &= \cup\{\mathrm{Inf}_{\mathcal{R}}(P\sigma) \mid P \in \mathrm{Inf}_{\mathcal{R}}(M)\} \end{aligned}$$

□

8.3 Cyclic Lambda Calculi

The work on Böhm semantics for non-confluent systems started with the development cyclic lambda calculi in [AB97a, AB97b]. In this article and corresponding

Table 8.3 The call-by-name cyclic lambda calculus

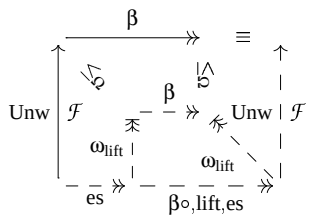
$(\lambda x.M)\ N$	$\xrightarrow[\beta\circ]{}$	$\langle M \mid x = N\rangle$	
$\langle C[x] \mid x = M, D\rangle$	$\xrightarrow[\text{es}]{}$	$\langle C[M] \mid x = M, D\rangle$	
$\langle M \mid x = C[y], y = N, D\rangle$	$\xrightarrow[\text{is}]{}$	$\langle M \mid x = C[N], y = N, D\rangle$	
$\langle\langle M \mid D_1\rangle \mid D_2\rangle$	$\xrightarrow[\text{em}]{}$	$\langle M \mid D_1, D_2\rangle$	
$\langle M \mid x = \langle N \mid D_1\rangle, D_2\rangle$	$\xrightarrow[\text{im}]{}$	$\langle M \mid x = N, D_1, D_2\rangle$	
$\langle M \mid D\rangle\ N$	$\xrightarrow[\text{lift}]{}$	$\langle M\ N \mid D\rangle$	
$M\ \langle N \mid D\rangle$	$\xrightarrow[\text{lift}]{}$	$\langle M\ N \mid D\rangle$	
$\lambda x.\langle M \mid D_1, D_2\rangle$	$\xrightarrow[\text{lift}]{}$	$\langle \lambda x.\langle M \mid D_1\rangle \mid D_2\rangle$	$(x = \bullet, D_1) \perp D_2 \neq \emptyset$
$\langle M \mid D_1, D_2\rangle$	$\xrightarrow[\text{gc}]{}$	$\langle M \mid D_1\rangle$	$D_2 \neq \emptyset, D_2 \perp \langle M \mid D_1\rangle$
$\langle M \mid\rangle$	$\xrightarrow[\text{gc}]{}$	M	
M^σ	$\xrightarrow[\text{cp}]{}$	M	$\sigma : \mathcal{V} \to \mathcal{V}$

technical report, three cyclic lambda calculi were developed: a call-by-name calculus, a sharing calculus and a call-by-value calculus. The first two calculi are cyclic extensions. The call-by-value cyclic lambda calculus is not a cyclic extension, because letrecs may be part of the infinite normal form. Therefore, we will only discuss the call-by-name and sharing calculi from the earlier work in this section. We will also briefly discuss the possibilities for a call-by-value calculus.

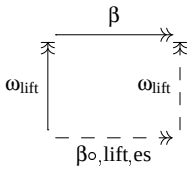
The call-by-name cyclic lambda calculus $\lambda\circ^{\to}_{\text{name}}$ is given in Table 8.3. We can derive an unwinding calculus from $\lambda\circ^{\to}_{\text{name}}$ by excluding the $\beta\circ$-rule and using Ω_{lift} as information content. We can then apply Thm 7.2.5 to show that $\lambda\circ^{\to}_{\text{name}}$ is infinitarily sound. To show that $\lambda\circ^{\to}_{\text{name}}$ is complete up to $\equiv$, it suffices to show that the subset consisting of $\beta\circ$, lift and es is complete up to $\equiv$:

Proposition 8.3.1 The rewrite system $\xrightarrow[\beta\circ,\text{lift},\text{es}]{}$ is a cyclic extension of the lambda calculus, which is complete up to $\equiv$.

Proof. We have the following diagram:



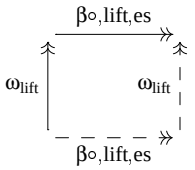
The key subdiagram is



To prove this diagram we replace every β step with a reduction sequence similar to

$$(\lambda x.M)\,N \xrightarrow[\beta\circ]{} \langle M \mid x = N\rangle \xrightarrow[es]{}\!\!\!\!\rightarrow \langle M[x := N] \mid x = N\rangle \xrightarrow[\omega_{\text{lift}}]{} M[x := N] \ .$$

The diagram is now proven if we can prove that



(8.7)

We will prove this diagram by tiling with elementary diagrams. The only possible elementary diagrams are

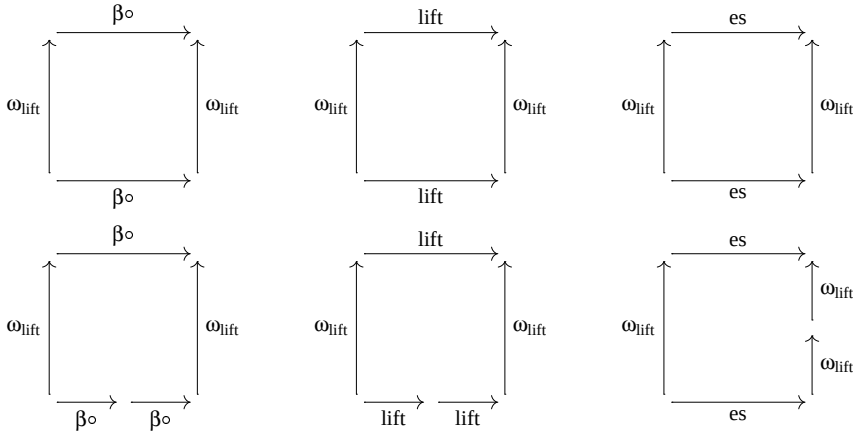


Diagram 8.7 follows by applying decreasing diagrams with the order

$$\text{es} > \omega_{\text{lift}} > \text{lift}, \beta\circ \ .$$

□

The cyclic sharing calculus $\lambda\circ_{\text{share}}$ is given in Table 8.4. It is easy to show that this calculus is infinitarily sound with respect to the lambda calculus. We will now show that it also complete up to:

Theorem 8.3.2 The rewrite system $\lambda\circ_{\text{share}}$ is complete up to $(\xrightarrow[\beta]{} \cup \leq_\Omega)^*$.

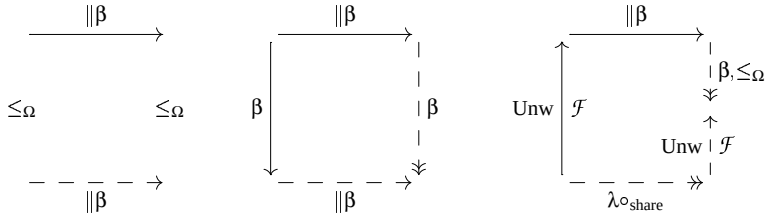
Table 8.4 The cyclic lambda calculus with sharing

$(\lambda x.M)N$	$\xrightarrow[\beta\circ]{}$	$\langle M \mid x = N\rangle$	
$\langle C[x] \mid x = V, D\rangle$	$\xrightarrow[\mathrm{es_V}]{}$	$\langle C[V] \mid x = V, D\rangle$	
$\langle M \mid x = C[x_1], x_1 = V, D\rangle$	$\xrightarrow[\mathrm{is_V}]{}$	$\langle M \mid x = C[V], x_1 = V, D\rangle$	
$\langle M \mid D\rangle N$	$\xrightarrow[\mathrm{lift}]{}$	$\langle MN \mid D\rangle$	
$M\langle N \mid D\rangle$	$\xrightarrow[\mathrm{lift}]{}$	$\langle MN \mid D\rangle$	
$\lambda x.\langle M \mid D, VD\rangle$	$\xrightarrow[\mathrm{lift}]{}$	$\langle \lambda x.\langle M \mid D\rangle \mid VD\rangle$	$x = \bullet, D \perp VD \neq \emptyset$
$\langle\langle M \mid D\rangle \mid D'\rangle$	$\xrightarrow[\mathrm{em}]{}$	$\langle M \mid D, D'\rangle$	
$\langle M \mid x = \langle N \mid D\rangle, D_1\rangle$	$\xrightarrow[\mathrm{im}]{}$	$\langle M \mid x = N, D, D_1\rangle$	
$\langle M \mid D, D'\rangle$	$\xrightarrow[\mathrm{gc}]{}$	$\langle M \mid D\rangle$	$\emptyset \neq D', D' \perp \langle M \mid D\rangle$
$\langle M \mid \rangle$	$\xrightarrow[\mathrm{gc}_\Box]{}$	M	
M	$\xrightarrow[\mathrm{cp_V}]{}$	N	$\exists\sigma : \mathcal{V} \to \mathcal{V}, N^\sigma \equiv M$ and $\forall x \not\equiv x', \sigma(x) \equiv \sigma(x') : \sigma(x)$ bound to a value in M
$C_{\mathrm{safe}}[M\ N]$	$\xrightarrow[\mathrm{name}]{}$	$C_{\mathrm{safe}}[\langle x \mid x = M\ N\rangle]$	x a new variable

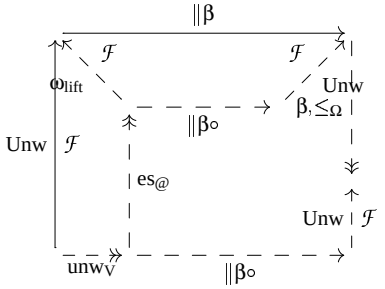
where

$$\begin{array}{rcl} C_{\mathrm{safe}} & ::= & C' \mid C[\lambda x.C'] \mid C[C'\ M] \mid C[M\ C'] \ . \\ C' & ::= & \Box \mid \langle C' \mid D\rangle \\ V & ::= & x \mid \lambda x.M \\ VD & ::= & x_1 = V_1, \cdots, x_n = V_n \end{array}$$

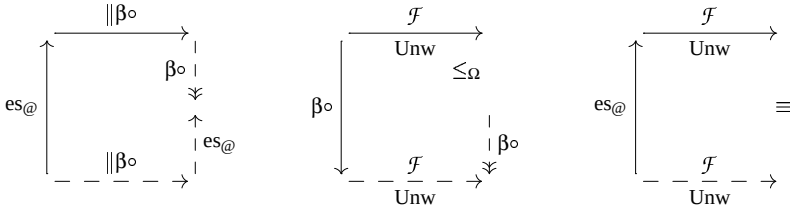
Proof. The result follows by induction on the length of the given reduction from the following three diagrams:



The first two diagrams are easily proven. The last diagram is proven as follows:



The top sub-diagram is easily proven, the left sub-diagram follows from the fact that $\mathrm{unw}_{\mathrm{V}}$ is an unwinding calculus and the last subdiagram follows from the following three diagrams:



□

Form all of this, we can conclude that the call-by-name cyclic lambda calculus and the cyclic sharing lambda calculus are cyclic extensions of the lambda calculus and hence that they have the same infinite normal forms.

To define a call-by-value cyclic lambda calculus as a cyclic extension, we must choose a suitable call-by-value lambda calculus. For example, we could use

$$\begin{array}{lcll} (\lambda x.M)\,N & \to & \mathsf{let}\ x = N\ \mathsf{in}\ M & \\ \mathsf{let}\ x = M\ \mathsf{in}\ C[x] & \to & \mathsf{let}\ x = M\ \mathsf{in}\ C[M] & \\ \mathsf{let}\ x = \lambda y.M\ \mathsf{in}\ N & \to & N & , \text{if } x \notin FV(N) \end{array}$$

Note that we do not include variables as values. We have to exclude because we want a call-by-value calculus that is substitutive continuous. For example, we should have that

$$\mathrm{Inf}_{\mathrm{value}}(\mathsf{let}\ x = (\lambda x.xx)\,(\lambda x.xx)\ \mathsf{in}\ \lambda x.x) = \Omega$$

and that

$$\mathrm{Inf}_{\mathrm{value}}(\lambda x.x) = \lambda x.x \ .$$

If we assume substitutive continuity then we have that

$$\begin{array}{l} \mathrm{Inf}_{\mathrm{value}}(\mathsf{let}\ x = (\lambda x.xx)\,(\lambda x.xx)\ \mathsf{in}\ \lambda x.x) \\ \quad = \mathrm{Inf}_{\mathrm{value}}((\mathsf{let}\ x = y\ \mathsf{in}\ z)\underbrace{[y := (\lambda x.xx)\,(\lambda x.xx), z := \lambda x.x]}_{\sigma}) \\ \quad = \cup\{\mathrm{Inf}_{\mathrm{value}}(a\sigma) \mid a \in \mathrm{Inf}_{\mathrm{value}}(\mathsf{let}\ x = y\ \mathsf{in}\ z)\} \ . \end{array}$$

We also have that

$$\cup\{\mathrm{Inf}_{\mathrm{value}}(a\sigma) \mid a \in \mathrm{Inf}_{\mathrm{value}}(z)\} = \mathrm{Inf}_{\mathrm{value}}(z\sigma) = \mathrm{Inf}_{\mathrm{value}}(\lambda x.x) \ .$$

Hence, we must conclude that

$$\mathrm{Inf}_{\mathrm{value}}(\mathsf{let}\ x = y\ \mathsf{in}\ z) \neq \mathrm{Inf}_{\mathrm{value}}(z) \ .$$

Thus, it is impossible to rewrite $\mathsf{let}\ x = y\ \mathsf{in}\ z$ to z while keeping the uniqueness of infinite normal forms property.

List of Symbols

Index

References

[AB97a] Z. M. Ariola and S. Blom. Cyclic lambda calculi. In Martín Abadi and Takayasu Ito, editors, *Theoretical Aspects of Computer Software*, volume 1281 of *Lecture Notes in Computer Science*, pages 77–106. Springer Verlag, September 1997.

[AB97b] Z. M. Ariola and S. Blom. Lambda calculi plus letrec. Technical Report IR-434, Department of Mathematics and Computer Science, Vrije Universiteit Amsterdam, October 1997.

[AK96] Z. M. Ariola and J. W. Klop. Equational term graph rewriting. *Fundamentae Informaticae*, 26(3,4):207–240, 1996. Extended version: CWI Report CS-R9552.

[AK97] Z. M. Ariola and J. W. Klop. Lambda calculus with explicit recursion. *Information and computation*, 139(2):154–233, December 1997.

[Bar84] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, Amsterdam, 1984.

[Blo95] S. Blom. A complete proofsystem for nested term graphs. In Gilles Dowek, Jan Heering, Karl Meinke, and Bernhard Möller, editors, *Higher-Order Algebra, Logic, and Term Rewriting*, volume 1074 of *Lecture Notes in Computer Science*, pages 74–89. Springer Verlag, 1995.

[Cor93] Andrea Corradini. Term rewriting in ct_σ. In *Proc. CAAP '93*, volume 668 of *Lecture Notes in Computer Science*, pages 468–484. Springer, Berlin, 1993.

[Cou83] B. Courcelle. Fundamental Properties of Infinite Trees. *Theorectical Computer Science*, 25:95–196, 1983.

[DJ90] N. Dershowitz and J. P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–320. Elsevier - The MIT Press, 1990.

[dMS95] André Luís de Medeiros Santos. *Compilation by Transformation in Non-Strict Functional Languages*. PhD thesis, University of Glasgow, July 1995.

[Hue80] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *JACM*, 27(4), 1980.

[KKSdV97] J.R. Kennaway, J.W. Klop, M.R. Sleep, and F.J. de Vries. Infinitary lambda calculus. *TCS*, 175:93–125, 1997.

[Klo80] Jan Willem Klop. *Combinatory Reduction Systems*. PhD thesis, CWI, 1980. Mathematical Centre Tracts 127.

[Klo92] J. W. Klop. Term rewriting systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume II, pages 1–116. Oxford University Press, 1992.

[KP93] G. Kahn and G. D. Plotkin. Concrete domains. *Theoretical Computer Science*, 121(1–2):187–277, 6 December 1993.

[KvOvR93] J. W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: Introduction and survey. *Theoretical Computer Science*, 121(1-2):279–308, 1993. A Collection of Contributions in Honour of Corrado Böhm on the Occasion of his 70th Birthday, guest eds. M. Dezani-Ciancaglini, S. Ronchi Della Rocca and M. Venturini-Zilli.

[LDLR99] Frédéric Lang, Daniel Dougherty, Pierre Lescanne, and Kristoffer Rose. Addressed term rewriting systems. Technical Report RR 1999-30, École Normal Supérieure de Lyon, June 1999.

[Lév78] J.-J. Lévy. *Réductions Correctes et Optimales dans le Lambda-Calcul*. PhD thesis, Universite Paris VII, October 1978.

[vO94] V. van Oostrom. *Confluence for Abstract and Higher-Order Rewriting*. PhD thesis, Vrije Universiteit, 1994.

Nederlandse samenvatting

Termgraafherschrijven

syntax en semantiek

De belangrijkste onderwerpen van dit proefschrift zijn een generalisering van de theorie van de Böhmboom en de toepassing van die theorie op niet-confluente termgraafherschrijfsystemen.

Niet-confluente termgraafherschrijfsystemen komen voort uit het modelleren van programma's, geschreven in (pure) functionele programmeertalen. Deze modellering kan op twee fundamenteel verschillende manieren gebeuren. De eerste methode modelleert de data als een eerste-orde termgraaf en de code als een graafherschrijfsysteem. De tweede methode modelleert de data en de code als één enkele tweede-orde termgraaf. Beide methodes zijn geschikt om de uitvoering van een programma te beschrijven. Voor het beschrijven van programmatransformaties is de tweede methode beter omdat het redeneren over een term eenvoudiger is dan het redeneren over een herschrijfsysteem.

Als eerste gaat het proefschrift in op de representatie van termgrafen met behulp van de *let*, *letrec* en μ syntax. Voor eerste-orde termgrafen wordt precies aangegeven welke grafen wel en welke niet met de verschillende syntactische constructies kunnen worden gerepresenteerd. Tevens wordt door middel van axiomatiseringen aangegeven welke termen equivalente grafen representeren. In het tweede gedeelte wordt de theorie van oneindige normaalvormen uitgewerkt. Eerst worden op het niveau van abstracte reductiesystemen de basisprincipes uitgewerkt. Vervolgens wordt een stelling bewezen waarmee een oneindige normaalvorm voor termen met *letrec* onder bepaalde voorwaarden kan worden afgeleid uit een oneindige normaalvorm voor gewone termen. Ten slotte wordt van een lambda calculus met *letrec* aangetoond dat deze aan de voorwaarden voldoet.

Acknowledgments

This thesis took a long time to be finished. I worked on it for 4 years as an AIO and for 2 years afterwards. During that time, many people supported me and/or were just good company. I'm grateful to all of them and I would like to mention some of them specifically.

First of all, I'd like to thank my promotor Jan Willem Klop and my copromotor and coauthor Zena Ariola for their guidance and support. They introduced me to the subject of term graph rewriting near the end of my master's degree in mathematics, and gave me the opportunity to do a PhD in computer science. Many of the problems studied in this thesis were suggested by them. I also thank Roel de Vrijer for his part in the supervision and for his interest in my work. Vincent van Oostrom read the draft of this thesis in an exceptionally thorough way. He gave many suggestions and found an annoying error in the draft. I thank him, Richard Kennaway, Detlef Plump and Roel de Vrijer for being members of my reading committee.

For the largest part of the 2 years between my position as a computer science AIO and finishing the draft of the thesis, I worked part time on a project in the mathematics department, supervised by Rien van Veldhuizen. He provided a unique environment, where I could freely schedule the work on the project and on my thesis.

Once most of the real work was done, I started working for the CWI. My bosses there (first Jan Friso Groote and later Wan Fokkink) were kind enough to let me deal with formalities and some corrections during working hours.

During my time as an AIO, I twice visited my copromotor in the USA for 6 months. To make these visits possible, she made some funds out of an NSF grant available. In addition, I twice received a grant from NWO. It was very kind of Andrzej Proskurowski to lend me his spare bike when I was in Eugene. I'm also very grateful to Jan Hillyer, who not only rented me a room, but more or less treated me as family and even invited my mother over for a month, the second time I was in Eugene.

Another visit was to Japan for a conference and a visit at ETL. I thank Aart Middeldorp and Fer-Jan de Vries for their guided tour of Japan. Support for this trip came from Shell and from ETL.

For coffee-breaks and lunches at the VU I often had good company in the form of Mirna Bognar, Jerry den Hartog, Michel Oey and (lunches only) Gerard Kok. In Eugene the most frequent companion in the quest for coffee was Miley

Semmelroth. For several years, I traveled from Amsterdam to Haarlem together with Remy Zandwijk. In Eugene, I was introduced to the game of bridge. Back at the VU, I regularly played in the staff-room and later we had a group of 5 people who played almost every day after lunch: Martijn Bot, Jerry den Hartog, Perry Groot, Evert Wattel and me.

Finally, I'd like to thank my mother for always being there when she was needed.

Titles in the IPA Dissertation Series

J.O. Blanco. *The State Operator in Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1996-1

A.M. Geerling. *Transformational Development of Data-Parallel Algorithms*. Faculty of Mathematics and Computer Science, KUN. 1996-2

P.M. Achten. *Interactive Functional Programs: Models, Methods, and Implementation*. Faculty of Mathematics and Computer Science, KUN. 1996-3

M.G.A. Verhoeven. *Parallel Local Search*. Faculty of Mathematics and Computing Science, TUE. 1996-4

M.H.G.K. Kesseler. *The Implementation of Functional Languages on Parallel Machines with Distrib. Memory*. Faculty of Mathematics and Computer Science, KUN. 1996-5

D. Alstein. *Distributed Algorithms for Hard Real-Time Systems*. Faculty of Mathematics and Computing Science, TUE. 1996-6

J.H. Hoepman. *Communication, Synchronization, and Fault-Tolerance*. Faculty of Mathematics and Computer Science, UvA. 1996-7

H. Doornbos. *Reductivity Arguments and Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1996-8

D. Turi. *Functorial Operational Semantics and its Denotational Dual*. Faculty of Mathematics and Computer Science, VUA. 1996-9

A.M.G. Peeters. *Single-Rail Handshake Circuits*. Faculty of Mathematics and Computing Science, TUE. 1996-10

N.W.A. Arends. *A Systems Engineering Specification Formalism*. Faculty of Mechanical Engineering, TUE. 1996-11

P. Severi de Santiago. *Normalisation in Lambda Calculus and its Relation to Type Inference*. Faculty of Mathematics and Computing Science, TUE. 1996-12

D.R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. Faculty of Mathematics and Computing Science, TUE. 1996-13

M.M. Bonsangue. *Topological Dualities in Semantics*. Faculty of Mathematics and Computer Science, VUA. 1996-14

B.L.E. de Fluiter. *Algorithms for Graphs of Small Treewidth*. Faculty of Mathematics and Computer Science, UU. 1997-01

W.T.M. Kars. *Process-algebraic Transformations in Context*. Faculty of Computer Science, UT. 1997-02

P.F. Hoogendijk. *A Generic Theory of Data Types*. Faculty of Mathematics and Computing Science, TUE. 1997-03

T.D.L. Laan. *The Evolution of Type Theory in Logic and Mathematics*. Faculty of Mathematics and Computing Science, TUE. 1997-04

C.J. Bloo. *Preservation of Termination for Explicit Substitution*. Faculty of Mathematics and Computing Science, TUE. 1997-05

J.J. Vereijken. *Discrete-Time Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1997-06

F.A.M. van den Beuken. *A Functional Approach to Syntax and Typing*. Faculty of Mathematics and Informatics, KUN. 1997-07

A.W. Heerink. *Ins and Outs in Refusal Testing*. Faculty of Computer Science, UT. 1998-01

G. Naumoski and W. Alberts. *A Discrete-Event Simulator for Systems Engineering*. Faculty of Mechanical Engineering, TUE. 1998-02

J. Verriet. *Scheduling with Communication for Multiprocessor Computation*. Faculty of Mathematics and Computer Science, UU. 1998-03

J.S.H. van Gageldonk. *An Asynchronous Low-Power 80C51 Microcontroller*. Faculty of Mathematics and Computing Science, TUE. 1998-04

A.A. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1998-05

E. Voermans. *Inductive Datatypes with Laws and Subtyping – A Relational Model*. Faculty of Mathematics and Computing Science, TUE. 1999-01

H. ter Doest. *Towards Probabilistic Unification-based Parsing*. Faculty of Computer Science, UT. 1999-02

J.P.L. Segers. *Algorithms for the Simulation of Surface Processes*. Faculty of Mathematics and Computing Science, TUE. 1999-03

C.H.M. van Kemenade. *Recombinative Evolutionary Search*. Faculty of Mathematics and Natural Sciences, Univ. Leiden. 1999-04

E.I. Barakova. *Learning Reliability: a Study on Indecisiveness in Sample Selection.* Faculty of Mathematics and Natural Sciences, RUG. 1999-05

M.P. Bodlaender. *Schedulere Optimization in Real-Time Distributed Databases*. Faculty of Mathematics and Computing Science, TUE. 1999-06

M.A. Reniers. *Message Sequence Chart: Syntax and Semantics*. Faculty of Mathematics and Computing Science, TUE. 1999-07

J.P. Warners. *Nonlinear approaches to satisfiability problems.* Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods.* Faculty of Computer Science, UT. 1999-09

P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems.* Faculty of Mechanical Engineering, TUE. 1999-11

J. Zwanenburg. *Object-Oriented Concepts and Proof Rules*. Faculty of Mathematics and Computing Science, TUE. 1999-12

R.S. Venema. *Aspects of an Integrated Neural Prediction System*. Faculty of Mathematics and Natural Sciences, RUG. 1999-13

J. Saraiva. *A Purely Functional Implementation of Attribute Grammars.* Faculty of Mathematics and Computer Science, UU. 1999-14

R. Schiefer. *Viper, A Visualisation Tool for Parallel Progam Construction.* Faculty of Mathematics and Computing Science, TUE. 1999-15

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic*. Faculty of Mathematics and Computer Science, UvA. 2000-01

T.E.J. Vos. *UNITY in Diversity. A stratified approach to the verification of distributed algorithms*. Faculty of Mathematics and Computer Science, UU. 2000-02

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes.* Faculty of Mathematics and Natural Sciences, RUG. 2000-03

W.O.D. Griffioen. *Studies in Computer Aided Verification of Protocols.* Faculty of Science, KUN. 2000-04

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor*. Faculty of Mathematics and Computing Science, TUE. 2000-05

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant*. Faculty of Mechanical Engineering, TUE. 2000-06

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs*. Faculty of Mathematics and Computing Science, TUE. 2000-07

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specification Language*. Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space Structure*. Faculty of Mathematics and Natural Sciences, UL. 2001-01

R. Ahn. *Agents, Objects and Events a computational approach to knowledge, observation and communication.* Faculty of Mathematics and Computing Science, TU/e. 2001-02

M. Huisman. *Reasoning about Java programs in higher order logic using PVS and Isabelle*. Faculty of Science, KUN. 2001-03

I.M.M.J. Reymen. *Improving Design Processes through Structured Reflection.* Faculty of Mathematics and Computing Science, TUE. 2001-04

S.C.C. Blom. *Term Graph Rewriting: syntax and semantics.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2001-05