A fully abstract semantics for concurrent graph reduction

ALAN JEFFREY

ABSTRACT. This paper presents a fully abstract semantics for a variant of the untyped λ-calculus with recursive declarations. We first present a summary of existing work on full abstraction for the untyped λ-calculus, concentrating on ABRAMSKY and ONG's work on the lazy λ-calculus. ABRAMSKY and ONG's work is based on leftmost outermost reduction without sharing. This is notably inefficient, and many implementations model sharing by reducing syntax graphs rather than syntax trees. Here we present a concurrent graph reduction algorithm for the λ-calculus with recursive declarations, in a style similar to BERRY and BOUDOL's Chemical Abstract Machine. We adapt ABRAMSKY and ONG's techniques, and present a program logic and denotational semantics for the λ-calculus with recursive declarations, and show that the three semantics are equivalent.

Contents

A fully abstract semantics for concurrent graph reduction

Alan Jeffrey

School of Cognitive and Computing Sciences
University of Sussex
Falmer
Brighton
BN1 9QH
UK

alanje@cogs.susx.ac.uk

Computer Science Report 12/93

Thanks to Lennart Augustsson, Matthew Hennessy, Mark Jones, John Launchbury, Edmund Robinson and Allen Stoughton for many useful comments.

This work has been funded by SERC project GR/H 16537.

1 Introduction

This paper is about the relationship between two fields of computer science: *full abstraction*, and *concurrent graph reduction*. Full abstraction is the study of relating denotational and operational semantics. Concurrent graph reduction is an efficient parallel implementation technique for non-strict functional programming languages.

In this paper we apply the techniques of ABRAMSKY (1989) and ONG (1988) to present a fully abstract denotational semantics for the concurrent graph reduction algorithm given in PEYTON JONES's textbook (1987).

In doing so, we use methods from full abstraction, compiler implementation, and concurrency theory.

1.1 Full abstraction

Full abstraction, originally defined by MILNER (1977), explores the relationship between an operational semantics of programming languages and its models. The operational view of a programming language is given by:

- A set of syntactic *terms* T, and a subset of terms called *programs*. The programs are then given an operational semantics.
- A set of *tests* together with an operational definition of when a term *passes* a test. This induces the *testing preorder* on terms $t \sqsubseteq_O u$ iff every test t passes is passed by u.

A *model* of such an operational view is:

- A partially ordered set $(\mathbf{D}, \leq)$.
- A function $[\![\cdot]\!] : T \to \mathbf{D}$. This induces the *denotational preorder* on terms $t \sqsubseteq_D u$ iff $[\![t]\!] \leq [\![u]\!]$.

We can then characterize such models:

- $\mathbf{D}$ is *correct* iff $t \sqsubseteq_O u$ implies $t \sqsubseteq_D u$.
- $\mathbf{D}$ is *complete* iff $t \sqsubseteq_D u$ implies $t \sqsubseteq_O u$.
- $\mathbf{D}$ is *fully abstract* iff it is correct and complete.

For example:

- In PLOTKIN's (1977) analysis of the typed functional language of Programming Computable Functions (PCF):
  - A term is a PCF term, and a program is a closed term. The operational semantics is given as a reductions $t \to u$ between programs.
  - A test is a closing context $C[\cdot]$ of type Bool or Int, together with a constant v. A term t passes $C[\cdot]$ iff $C[t]$ evaluates to v.

  This is then given a denotational semantics in terms of complete partial orders and continuous functions. PLOTKIN showed that this denotational semantics is correct but not complete, and showed that this denotational semantics is comlete for an extension of PCF with a 'parallel conditional' term pcond of type $\mathsf{Bool} \to \mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}$ with the semantics:

$$[\![\mathsf{pcond}\,tuv]\!]\sigma = \begin{cases} [\![u]\!]\sigma & \text{if } [\![t]\!]\sigma = 0 \text{ or } [\![u]\!]\sigma = [\![v]\!]\sigma \\ [\![v]\!]\sigma & \text{if } [\![t]\!]\sigma = 1 \text{ or } [\![u]\!]\sigma = [\![v]\!]\sigma \\ \bot & \text{otherwise} \end{cases}$$

  If such a term is added to PCF (and given an appropriate operational semantics) then the semantics is complete.

- In DE NICOLA's (1985) analysis of HOARE's (1985) Communicating Sequential Processes (CSP):
  - A term is a CSP process, and a program is a closed process. The operational semantics is given as a *labelled transition system* between programs $P \xrightarrow{a} Q$, in the style of MILNER (1989).
  - A test is a closed substitution ρ, a program T, and a special action ω. A process P passes (ρ, T, ω) iff every maximal computation of $P[\rho] \,||\, T$ passes through a state $P' \,||\, T'$ where T' can perform ω. This is HENNESSY's (1988) *must-testing* equivalence.

  This is then given a denotational semantics in a variant of BROOKES, HOARE and ROSCOE's (1984) *failures–divergences* model. DE NICOLA showed that this denotational semantics is fully abstract.

- In ABRAMSKY (1989) and ONG's (1988) analysis of the untyped λ-calculus:
  - A term is an untyped λ-calculus term, and a program is a closed term. The operational semantics is given as *leftmost-outermost* reduction between programs $M \to N$.
  - A test is a closing context $C[\cdot]$. A term M passes $C[\cdot]$ iff $C[M]$ evaluates to *weak head normal form*, that is a λ-term $\lambda w.\, N$.

  This is then given a denotational semantics in terms of complete partial orders and continuous functions. ABRAMSKY and ONG showed that this denotational semantics is correct but not complete, and that the completeness problem can again be reduced to definability, in that there is no untyped λ-calculus 'parallel convergence test' term P with the semantics:

$$[\![\mathsf{P}\,xyz]\!]\sigma = \begin{cases} \bot & \text{if } [\![x]\!]\sigma = [\![y]\!]\sigma = \bot \\ [\![z]\!]\sigma & \text{otherwise} \end{cases}$$

and that if such a term is added (and given an appropriate operational semantics) then the semantics is complete.

This paper is based on ABRAMSKY and ONG's work, which is surveyed in Chapter 2. The rest of this section will summarize that Chapter.

Given an infinite set V, ranged over by x, y and z, the untyped λ-calculus with P (Λ_{P}) is defined:

$$M ::= x \mid MM \mid \lambda x\,.\,M \mid \mathsf{P}\,MN$$

This can be given an operational semantics $M \to N$ with leftmost-outermost reduction, that is reduction is allowed on the left of an application, but not on the right, or inside a λ. We then define $M\Downarrow$ iff M reduces to *weak head normal form* (*whnf*), that is $M \to^* \lambda x\,.\,N$. Our notion of test is then a closing context $C[\cdot]$, and M passes $C[\cdot]$ iff $C[M]\Downarrow$. This induces the testing preorder:

$$M \sqsubseteq_O N \quad \text{iff} \quad C[M]\Downarrow \text{ implies } C[N]\Downarrow \text{ for any closing } C$$

This preorder is based on MORRIS's (1968) extensional preorder, but is based on leftmost-outermost reduction to whnf rather than full reduction to normal form, which was studied by BARENDREGT (1984).

The denotational semantics for Λ_{P} is given in the initial domain isomorphic to its own lifted continuous function space:

$$\mathbf{D} \simeq (\mathbf{D} \to \mathbf{D})_\perp$$

We can then give a semantics $[\![M]\!]\sigma$ in $\mathbf{D}$, where $\sigma : V \to \mathbf{D}$ is an *environment* assigning a meaning to any free variables in M. This induces the denotational preorder:

$$M \sqsubseteq_D N \quad \text{iff} \quad [\![M]\!] \leq [\![N]\!]$$

To link the operational and denotational semantics, we present a third semantics, which can be used as a 'stepping stone'. This is in the form of a *program logic*, with the language of propositions (or COPPO types (BARANDREGT *et al.*, 1983)) Φ defined:

$$\phi ::= \omega \mid \phi \wedge \phi \mid \phi \to \phi$$

We can give an operational characterization of when a closed term satisfies a proposition $\models M : \phi$, similar to the operational characterization of HENNESSY–MILNER (1980) logic:

- $\models M : \omega$ for any M.
- $\models M : \phi \wedge \psi$ iff $\models M : \phi$ and $\models N : \psi$.
- $\models M : \phi \to \psi$ iff $M\Downarrow$ and $\forall N\,.\,(\models N : \phi) \Rightarrow (\models MN : \psi)$.

For example, if Ω is a term which never reaches whnf then:

- Ω satisfies ω

- $\lambda x\,.\,\Omega$ satisfies $\omega \to \omega$.
- $\lambda x\,.\,x$ satisfies every $\phi \to \phi$.
- $\lambda xy\,.\,x$ satisfies every $\phi \to (\omega \to \phi)$.

This can be generalized to open terms by defining a *context* Γ to be a list of the form $x_1 : \phi_1, \ldots, x_n : \phi_n$ for distinct x_i. Then:

- $x_1 : \phi_1, \ldots, x_n : \phi_n \models M : \phi$ iff $\models M[M_1/x_1, \ldots, M_n/x_n] : \phi$ whenever $\models M_i : \phi_i$.

We can give two other characterizations of the logic:

- A denotational semantics $[\![\cdot]\!] : \Phi \to \mathbf{D}$.
- A proof system $\Gamma \vdash M : \phi$.

We can then show that the problem of full abstraction is one of showing that the three presentations of the logic agree, that is:

$$\Gamma \models M : \phi \quad \text{iff} \quad \Gamma \vdash M : \phi \quad \text{iff} \quad [\![\phi]\!] \leq [\![M]\!][\![\Gamma]\!]$$

To prove this, we show that the program logic characterizes the *compact* elements of $\mathbf{D}$, so a is compact iff $\exists \phi\,.\,[\![\phi]\!] = a$. From this we can show:

$$\Gamma \vdash M : \phi \quad \text{iff} \quad [\![\phi]\!] \leq [\![M]\!][\![\Gamma]\!]$$

We can then use some operational reasoning to show:

$$\Gamma \vdash M : \phi \quad \text{implies} \quad \Gamma \models M : \phi \quad \text{implies} \quad [\![\phi]\!] \leq [\![M]\!][\![\Gamma]\!]$$

The presentation in Chapter 2 follows ABRAMSKY and ONG quite closely, although the proofs are self-contained. The main differences are:

- We concentrate on the 'small step' operational semantics $M \to N$ rather than the 'big step' semantics $M\Downarrow$, since this agrees with our treatment of graph reduction in Chapter 3.
- We make no use of *applicative bisimulation*.
- The proofs are more concrete, and do not use all of the abstract machinery of ABRAMSKY's (1991) *domain theory in logical form*. The interested reader is highly encouraged to read that paper for less *ad hoc* proofs.

We shall follow the same outline in Chapter 3 when we prove full abstraction for concurrent graph reduction.

1.2 Concurrent graph reduction

Graph reduction is an efficient implementation technique for non-strict functional programming languages, such as AUGUSTSSON's (1984) Lazy ML, FAIRBURN's (1982) Ponder, JONES's (1992) Gofer, TURNER's (1985) Miranda, and Haskell (HUDAK *et al.*, 1992).

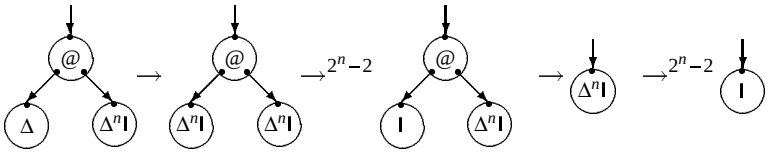
It was developed by WADSWORTH (1971) as an implementation of leftmost-outermost reduction. He observed that leftmost-outermost reduction can take exponential time to evaluate an expression, due to loss of *sharing* information. For example, if we define:

$$\mathsf{I} = \lambda x\,.\,x \qquad \Delta = \lambda x\,.\,xx \qquad M^0N = N \qquad M^{n+1}N = M(M^nN)$$

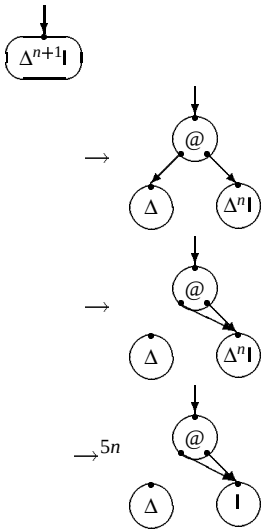
Then the evaluation of $\Delta^{n+1}\mathsf{I} \to^* \mathsf{I}$ is:

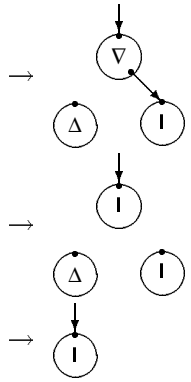
$$\Delta^{n+1}\mathsf{I} \to (\Delta^n\mathsf{I})(\Delta^n\mathsf{I}) \to^{2^n-2} \mathsf{I}(\Delta^n\mathsf{I}) \to \Delta^n\mathsf{I} \to^{2^n-2} \mathsf{I}$$

Thus, $\Delta^n\mathsf{I}$ takes 2^n-2 reductions to terminate. This exponential blow-up is caused by copying $\Delta^n\mathsf{I}$ in the reduction $\Delta^{n+1}\mathsf{I} \to (\Delta^n\mathsf{I})(\Delta^n\mathsf{I})$, and can clearly be seen if we draw the syntax trees for this reduction, where '@' denotes function application:



This inefficiency is caused by the implementation of β-reduction with substitution. When we reduce $(\lambda w.\,M)N \to M[N/w]$, we make a separate copy of N for each occurrence of w in M, and each copy then has to be reduced separately. We can remove this inefficiency if, rather than copying terms, we copy *pointers to* terms, that is we reduce syntax *graphs* rather than syntax *trees*. For example, the graph reduction of $\Delta^{n+1}\mathsf{I}$ is, where '∇' denotes a pointer or *indirection node*:





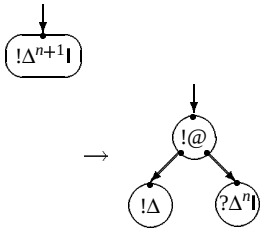
These steps are:

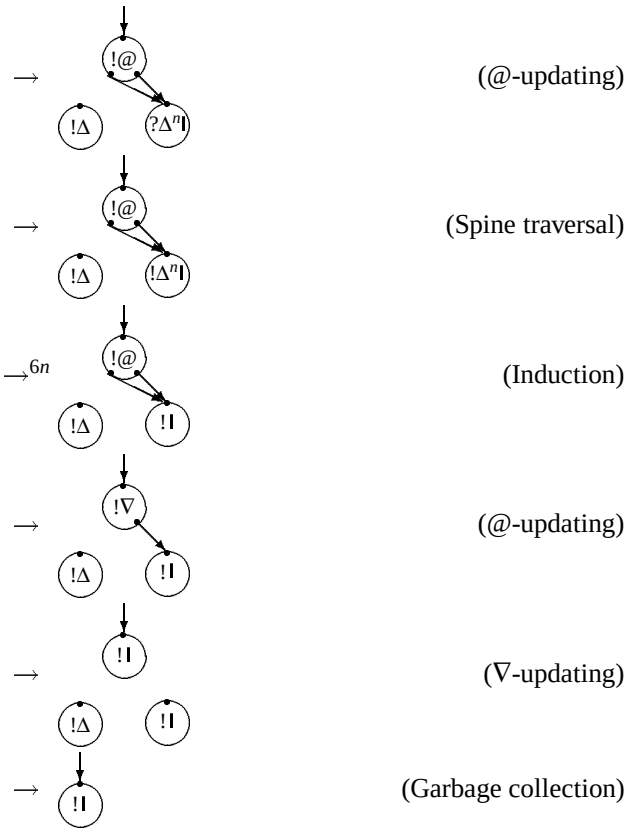
- *Graph building*, where we expand the definition of $\Delta^{n+1}\mathsf{I}$ and turn it into a graph.
- *Application updating*, (or β-reduction) where we apply the function Δ.
- By induction, we evaluate $\Delta^n\mathsf{I}$ to I in $5n$ steps.
- *Application updating*, where we apply the function I to produce an *indirection* node.
- *Indirection updating*, since the indirection node points to a node already in whnf, we can copy it.
- *Garbage collection*, where we remove any unwanted nodes.

Since each step of a graph reduction involves a small number of nodes in the graph, there is a fine grain of *granularity* and thus much scope for concurrency. In our simplified view of concurrent graph reduction, we will add a flag to each node of the graph indicating whether it is currently under evaluation. Thus each node is either of the form:

- $!M$ representing a *tagged* node which is being evaluated.
- $?M$ representing an *untagged* node which is not.

For example, the reduction of $\Delta^{n+1}\mathsf{I}$ carried out by one processor is:





The new reduction is:

- *Spine traversal*, where we tag an untagged node that is needed.

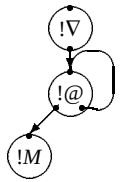
We now have a number of possible graphs for each λ-calculus term, depending on which nodes we wish to tag. There are (at least) two approaches to determining how many nodes should be tagged:

- *Sequential reduction* is achieved by initially only tagging one node in the graph, and not allowing graph building to introduce new tagged nodes. This means that (apart from garbage collection) there will only be one reduction possible at any one moment, and so we are using the tagging information only to record the *spine stack* of the graph (PEYTON JONES, 1987, Ch. 11).
- *Concurrent reduction* is achieved by initially tagging a number of nodes in the graph. These nodes can then be evaluated concurrently, and so we are using the tagging information to record the *blocking* information of the graph (PEYTON JONES, 1987, Ch. 24).

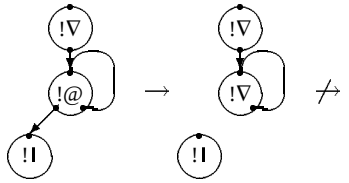
If we decide to use concurrent reduction, there are (at least) two approcahes to determining which nodes should be tagged:

- *Strictness analysis* (PEYTON JONES, 1987, Ch. 22) is an automated way of determining which nodes in a graph are guaranteed to be used. Any such node can always be tagged. Strictness analysis is in general undecidable, so any practical algorithm will fail to tag some nodes, but any nodes that are tagged are guaranteed to be used.
- *Program annotation* (PEYTON JONES, 1987, Ch. 24) places the burden of deciding which nodes to tag on the programmer. For example, this is the approach taken in Part 3 where we allow two forms of recursive declaration: tagged $\mathsf{rec}\,x := !M \,\mathsf{in}\, N$ and untagged $\mathsf{rec}\,x := ?M \,\mathsf{in}\, N$. This is obviously the simplest approach for the compiler writer (and semanticist!) to take.

As well as acyclic graphs, we can allow cyclic graphs, which allow for more efficient recursive programs. For example, rather than implement the fixed point of M as $\mathsf{Y}\,M$, we could use the cyclic graph:



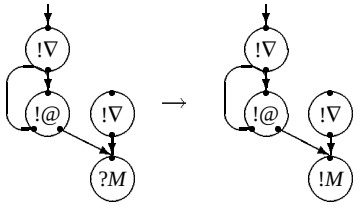
However, this presents a semantic problem not present in the λ-calculus, since $\mathsf{Y}\,\mathsf{I}$ *diverges*, whereas the cyclic fixed point *deadlocks* since:



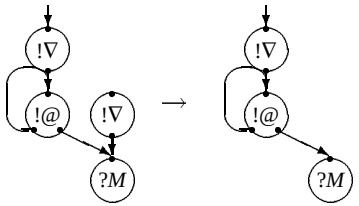
Such terms are called *black holes*, and one design decision in a semantics is whether or not to identify divergence and deadlock. Here, we will identify them, although in the author's (1993) semantics, they were distinguished.

Another semantic problem caused by concurrent graph reduction is that it is

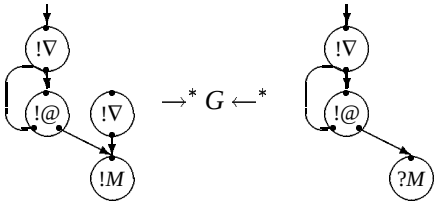
not *confluent* (or *Church–Rosser*), since by spine traversal:



and by garbage collection:



but there is no graph G such that:



This is unfortunate, since confluence is a very useful way of proving properties of operational semanitcs. However, we shall see in Section 3.6 that there is a *reduction strategy* for concurrent graph reduction which is confluent.

1.3 Full abstraction and graph reduction

We have now seen:

- A well-developed theory of fully abstract semantics.
- A well-developed practice of concurrent graph reduction.

However, there has been little work on relating these. There have been a number of proofs of *correctness* for graph reduction, which will be discussed further in Chapter 4:

- WADSWORTH (1971) showed that graph reduction of a λ-calculus term converges iff tree reduction converges. Since every tree context is a graph context, this means that the testing model for graph reduction is correct for tree reduction. However, not every graph context is a tree context, and so this does not show that the testing model for graph reduction is fully abstract for tree reduction.

- BARENDREGT *et al.* (1987) generalized WADSWORTH's result to an arbitrary graph rewriting system. There has since been much work on relating graph reduction to tree reduction, for example the correctness results of KENNAWAY *et al.* (1993a) and the other papers in SLEEP *et al.*'s (SLEEP *et al.*, 1993) book.
- LESTER (1989) has shown that a denotational semantics for the typed λ-calculus is correct for the operational semantics of JOHNSSON's (1984) *G*-machine
- LAUNCHBURY (1993) has shown that correct semantics for graph reduction including black holes can be given in the semantic domain $\mathbf{D} \simeq (\mathbf{D} \to \mathbf{D})_\perp$.
- PURUSHOTHAMAN and SEAMAN (1992) have shown that a denotational semantics for PCF with sharing is correct for an operational semantics with explicit closures.
- The author (1993) has shown that a variant of the semantics given in Chapter 3 is correct for tree reduction.

However, there have been no proofs of full abstraction for concurrent graph reduction. In this paper, we will follow ABRAMSKY (1989) when he said:

> Since current practice is well-motivated by efficiency considerations and is unlikely to be abandoned readily, it makes sense to see if a good modified theory can be developed for it.

In Chapter 3 we present a formal treatment of concurrent graph reduction, based on BERRY and BOUDOL's (1990) *Chemical Abstract Machine* (CHAM). This semantics includes:

- Tagged and untagged nodes.
- Garbage collection.
- Deadlocked graphs.

We also present a denotational semantics in $\mathbf{D} \simeq (\mathbf{D} \to \mathbf{D})_\perp$ in which:

- Whether a node is tagged or not is irrelevant.
- Garbage collection is semantically unimportant.
- Deadlock and divergence are identified.

We will then apply the techniques of Chapter 2 to show that this semantics is correct, and that by including *parallel convergence* nodes in the syntax, the semantics is complete. In order to show this, we give a program logic and proof system similar to ABRAMSKY and ONG's, and use this as a bridge between the operational semantics for graph reduction and the denotational semantics.

In order to carry out this proof, we have to show a number of subsidiary results about concurrent graph reduction:

- Garbage collection is semantically unimportant, so a graph can converge iff it can converge without garbage collecting. One would expect this to be true, since garbage collection is introduced only because of memory limitations.
- Tagging is semantically unimportant, so a graph can converge irrespective of whether its nodes are tagged or not. In particular, this means that concurrent evaluation is semantically equivalent to sequential evaluation.
- Referential transparency, which means that it is semantically unimportant if a graph contains a copy of a node, or a pointer to a node.

There are a number of applications for a fully abstract semantics:

VERIFYING COMPILER OPTIMIZATIONS. A number of compilers of non-strict functional languages, notably JOHNSSON's (1984) Lazy ML compiler for the *G*-machine, make use of optimizations. Many optimizers, notably *peephole optimizers* (PEYTON JONES, 1987, Ch. 20) replace one small term with another semantically equivalent, but more efficient term. If a semantics is correct, then we know that any such optimization will have the same operational behaviour in all contexts.

Unfortunately, if the semantics is not complete, then there may be valid optimizations that are not semantically equivalent, and there is a temptation for the compiler writer to use *ad-hoc* reasoning to justify a semantically invalid optimization, on the grounds that the semantics is too fine. If the semantics is fully abstract, then such reasoning is invalid, since we can always find a context which will distinguish inequivalent terms.

ANALYZING OTHER MODELS. Given a correct model, we know that any finer model must also be correct. For example, we might extend a denotational model to include sharing or strictness analysis, and we know that the resulting model will still be correct.

Similarly, given a complete model, we know that any coarser model must also be complete. For example, MYCROFT's (1981) *abstract interpretation* for strictness analysis is a coarser model than the standard denotational model. Thus if the standard model is complete, then we know that the abstract interpretation is also complete, without having to perform any operational reasoning.

PRODUCING DISTINGUISHING FORMULAE. If a denotational semantics has an equivalent program logic, we can use it to produce *distinguishing formulae*. That is, given two denotationally distinct terms, we can find a logical formula which one satisfies and the other does not.

Such distinguishing formulae can be used in proof tools to provide a form of debugging: if the tool proves that two terms are different, it can report this to the user along with a distinguishing formula which shows *why* the terms are different. This information is invaluable when using a proof tool as part of the design

process, rather than as *post hoc* verification. Distinguishing formulae have been used in process algebra tools such as the Concurrency WorkBench (CLEAVELAND *et al.*, 1989) and TAV (LARSEN *et al.*, 1989).

2 Tree reduction

This Chapter presents a summary of existing work on fully abstract models for leftmost-outermost reduction of the untyped λ-calculus. It concentrates on ABRAMSKY (1989) and ONG's (1988) work on the lazy λ-calculus, but also includes material from ABRAMSKY (1991), BARENDREGT (1984), BARENDREGT*et al.* (1983) BOUDOL (1992), PIERCE (1991) and PLOTKIN (1983).

2.1 The λ-calculus with P

In this Chapter, we will discuss the theory developed by ABRAMSKY and ONG, based on *leftmost-outermost* reduction. This is the semantic basis of the *non-strict* functional languages such as AUGUSTSSON's (1984) Lazy ML, FAIRBURN's (1982) Ponder, JONES's (1992) Gofer, TURNER's (1985) Miranda, and Haskell (HUDAK *et al.*, 1992).

In the untyped λ-calculus, all expressions are functions, and these functions take functions as inputs, and return other functions. We can regard this as a pure theory of computation, abstracted away from considerations of data.

The untyped λ-calculus has three forms of expression:

- A *free variable* x.
- An *application* MN.
- An *abstraction* $\lambda x\,.\,M$.

Such terms are *sequential* and the only form of computation is β-reduction, where an abstraction is applied $(\lambda x\,.\,M)N \to M[N/x]$. Following PLOTKIN (1977) we would expect that finding a fully abstract semantics will be much simpler if we add some form of parallel computation. There are a number of possible parallel combinators one can add: PLOTKIN used 'parallel conditional', ABRAMSKY and ONG used 'parallel convergence', and BOUDOL (1992) used 'parallel join'. We will follow ABRAMSKY and ONG, and extend the λ-calculus to the λ-calculus with P, and add:

- A *parallel convergence test* $\mathsf{P}\,MN$.

We will show below that we can implement BOUDOL's (1992) parallel join using P. Such a test will converge to the identity function iff either of its arguments converges. Such a test is similar to AUGUSTSSON's (1989) *oracular choice* except that AUGUSTSSON's choice returns a flag indicating which of its arguments terminated, thus introducing nondeterminism. We would like to preserve determinism (reflected by *confluence*, discussed in Section 3.6) and so we will use the weaker 'parallel convergence' test. In summary:

DEFINITION. Let V be an infinite set of variables ranged over by x, y and z. Then Λ_{P} is defined:

$$M ::= x \mid MM \mid \lambda x\,.\,M \mid \mathsf{P}\,MM$$

- M is in *weak head normal form* (whnf) iff $M = \lambda x\,.\,N$.
- Let $\mathsf{fv}M$ be the *free variables* of M.
- A *closed* term (or *program*) has $\mathsf{fv}M = \emptyset$.
- A *context* $C[\cdot]$ is a syntactic term with a number of 'holes' represented by $\cdot$. $C[M]$ is $C[\cdot]$ with each hole filled by M. $C[\cdot]$ is *closing* for M if $C[M]$ is closed.
- A *substitution* is a function $\rho : V \to \Lambda_{\mathsf{P}}$ which is almost everywhere the identity. Let $(M_1 \ldots M_n/x_1 \ldots x_n)$ be the substitution ρ such that $\rho x = M_i$ if $x = x_i$ and $\rho x = x$ otherwise.
- Let $M[\rho]$ be M with any free variable x replaced by ρx, with appropriate α-conversion to avoid capture of free variables. □

EXAMPLES.

- $\mathsf{I} = \lambda x\,.\,x$ is the *identity* combinator.
- $\mathsf{K} = \lambda x\,.\,\lambda y\,.\,x$ is the *constant* combinator.
- $\mathsf{Y}\,M = (\lambda x\,.\,M(xx))(\lambda x\,.\,M(xx))$ is the *fixed point* of M.
- $\Delta = \lambda x\,.\,xx$ is the *diagonal* combinator.
- $\Omega = \Delta\Delta$ is the *divergent* combinator which never converges.
- $\Upsilon = (\lambda x\,.\,\lambda y\,.\,xx)(\lambda x\,.\,\lambda y\,.\,xx)$ is the *ogre* combinator which always converges.
- $\lambda^{\nu}x\,.\,M = \lambda x\,.\,\mathsf{P}\,xxM$ is a *strict*(or *call-by-value*) abstraction.
- $\mathsf{J} = \mathsf{Y}(\lambda x\,.\,\lambda y\,.\,\lambda z\,.\,(\mathsf{P}\,yz(\lambda w\,.\,x(yw)(zw))))$ is the *join* combinator.
- $\mathsf{M} = \mathsf{Y}(\lambda x\,.\,\lambda^{\nu}y\,.\,\lambda^{\nu}z\,.\,\lambda w\,.\,x(yw)(zw))$ is the *meet* combinator.
- $\mathsf{A} = \lambda x\,.\,\lambda y\,.\,\lambda^{\nu}z\,.\,y(zx)$ is the *arrow* combinator. □

2.2 Operational semantics

Computation in the untyped λ-calculus is represented by β-reductions of the form $(\lambda x\,.\,M)N \to M[N/x]$, and the various operational semantics for the untyped λ-calculus differ only in where β-reduction can take place. In the standard theory presented by BARENDREGT (1984), β-reduction can take place anywhere in a term, whereas in ABRAMSKY and ONG's theory, reduction can only take place on the *left* of an application, and *outside* an abstraction. For example, $\mathsf{II} \to \mathsf{I}$, and $\mathsf{II}M \to \mathsf{I}M$, but $M(\mathsf{II}) \not\to M\mathsf{I}$ and $\lambda x\,.\,(\mathsf{II}) \not\to \lambda x\,.\,\mathsf{I}$.

The operational semantics for the λ-calculus with P is that of the untyped λ-calculus, with the addition that $\mathsf{P}MN \to \mathsf{I}$ iff M or N is in whnf, and that reduction is allowed inside either argument of P. This allows for *interleaved* concurrency,

since if $M \to M'$ and $N \to N'$ then:

$$\begin{array}{ccc} \mathsf{P}MN & \to & \mathsf{P}M'N \\ \downarrow & & \downarrow \\ \mathsf{P}MN' & \to & \mathsf{P}M'N' \end{array}$$

From this operational semantics, we can define the *may testing* preorder where a test is a closing context $C[\cdot]$ and M passes $C[\cdot]$ iff $C[M]$ converges. In summary:

DEFINITION. $\to$ is given by axioms:

$$\begin{array}{cc} (\beta) & (\lambda x\,.\,M)N \to M[N/x] \\ (\mathsf{P}a) & \mathsf{P}(\lambda x\,.\,M)N \to \mathsf{I} \\ (\mathsf{P}b) & \mathsf{P}M(\lambda x\,.\,N) \to \mathsf{I} \end{array}$$

and structural rules:

$$(@_{\mathrm{L}})\ \frac{M \to M'}{MN \to M'N} \qquad (\mathsf{P}_{\mathrm{L}})\ \frac{M \to M'}{\mathsf{P}MN \to \mathsf{P}M'N} \qquad (\mathsf{P}_{\mathrm{R}})\ \frac{N \to N'}{\mathsf{P}MN \to \mathsf{P}MN'}$$

- $M \Downarrow N$ iff $M \to^* N$ and N is in whnf.
- $M\Downarrow$ iff $\exists N\,.\,M \Downarrow N$.
- $M\Uparrow$ iff $\neg\exists N\,.\,M \Downarrow N$.
- $M \sqsubseteq_O N$ iff $C[M]\Downarrow \Rightarrow C[N]\Downarrow$ for any closing context C. □

EXAMPLES.

- $\mathsf{I}M \to M$.
- $\mathsf{K}MN \to^2 M$.
- $\mathsf{Y}M \to M(\mathsf{Y}M)$.
- $\Delta M \to MM$.
- $\Omega \to \Omega$, so $\Omega\Uparrow$.
- $\Upsilon \to \lambda x\,.\,\Upsilon$, so $\Upsilon\Downarrow$ and $\Upsilon M \to^2 \Upsilon$.
- If $N\Downarrow$ then $(\lambda^v x\,.\,M)N \to^* M[N/x]$. Otherwise $(\lambda^v x\,.\,M)N\Uparrow$.
- If $M\Downarrow$ or $N\Downarrow$ then $\mathsf{J}MN\Downarrow$ and $\mathsf{J}MNO \to^* \mathsf{J}(MO)(NO)$.
- If $M\Uparrow$ and $N\Uparrow$ then $\mathsf{J}MN\Uparrow$.
- If $M\Downarrow$ and $N\Downarrow$ then $\mathsf{M}MN\Downarrow$ and $\mathsf{M}MNO \to^* \mathsf{M}(MO)(NO)$.
- If $M\Uparrow$ or $N\Uparrow$ then $\mathsf{M}MN\Uparrow$.
- If $O\Downarrow$ then $\mathsf{A}MNO \to^* M(ON)$. Otherwise $\mathsf{A}MNO\Uparrow$. □

2.3 Denotational semantics

The denotational semantics for Λ_{P} is given in the domain $\mathbf{D}$ that is isomorphic to its own lifted continuous function space. Thus, any element of $\mathbf{D}$ is either $\bot$ (representing a divergent term such as Ω) or a continuous function from $\mathbf{D}$ to $\mathbf{D}$ (representing a convergent term such as $\lambda x\,.\,M$). This semantics identifies all divergent terms, and distinguishes divergent and convergent terms. In particular, Ω and $\lambda x\,.\,\Omega$ are distinguished, since the former diverges whilst the latter converges.

DEFINITION. $\mathbf{D}$ is the initial solution of:

$$\mathbf{D} \simeq (\mathbf{D} \to \mathbf{D})_\bot$$

where if X and Y are ω-cpos:

- $X_\bot$ is X with a new bottom element.
- $X \to Y$ is the continuous function space from X to Y.

This definition will be clarified in Section 2.5. Let the ω-continuous functions $\mathsf{unfold} : \mathbf{D} \to (\mathbf{D} \to \mathbf{D})_\bot$ and $\mathsf{fold} : (\mathbf{D} \to \mathbf{D})_\bot \to \mathbf{D}$ form this isomorphism. □

In Proposition 5 we shall show that $\mathbf{D}$ is a *complete lattice* and so every set of elements $A \subseteq \mathbf{D}$ has a *join* or *least upper bound* $\bigvee A$. In particular, this means that:

- There is a top element $\top = \bigvee \mathbf{D}$.
- There is a bottom element $\bot = \bigvee \emptyset$.
- Every pair of elements has a join $a \vee b = \bigvee\{a, b\}$.
- Every pair of elements has a meet $a \wedge b = \bigvee\{c \mid a \geq c \leq b\}$.

We can then define the denotational semantics of a term M to be $[\![M]\!]\sigma$, where $\sigma : V \to \mathbf{D}$ is an *environment* used to bind any free variables in M. For example, $[\![x]\!]\sigma = \sigma x$

DEFINITION. Let $\Sigma = V \to \mathbf{D}$. Then define $[\![M]\!]$ in $\Sigma \to \mathbf{D}$ as:

$$\begin{aligned} [\![x]\!] &= \mathsf{read}\,x \\ [\![MN]\!] &= \mathsf{split}(\mathsf{apply} \circ [\![M]\!])[\![N]\!] \\ [\![\lambda x\,.\,M]\!] &= \mathsf{fold} \circ \mathsf{lift} \circ \mathsf{fn}\,x[\![M]\!] \\ [\![\mathsf{P}MN]\!] &= \mathsf{split}(\mathsf{fork} \circ [\![M]\!])[\![N]\!] \end{aligned}$$

where:

$$\begin{aligned} \mathsf{read}\,x\sigma &= \sigma x \\ \mathsf{split}\ fg\sigma &= f\sigma(g\sigma) \\ \mathsf{fn}\,x f\sigma &= f \circ \mathsf{update}\,\sigma x \\ \mathsf{fork}\,ab &= \begin{cases} \bot & \text{if } a = b = \bot \\ \mathsf{fold}(\mathsf{lift}\,\mathsf{id}) & \text{otherwise} \end{cases} \\ \mathsf{apply}\,ab &= \begin{cases} fb \text{ if } \mathsf{unfold}\,a = \mathsf{lift}\ f \\ \bot \text{ otherwise} \end{cases} \\ \mathsf{update}\,\sigma xay &= \begin{cases} a & \text{if } x = y \\ \sigma y & \text{otherwise} \end{cases} \end{aligned}$$

Define $[\![\rho]\!]$ in $\Sigma \to \Sigma$ as:

$$[\![\rho]\!]\sigma x = [\![\rho x]\!]\sigma$$

Then $M \sqsubseteq_D N$ iff $[\![M]\!] \leq [\![N]\!]$. □

We have presented this semantics using higher-order functions such as split and apply, since this makes the denotational reasoning in Section 3.9 simpler. Expanding out the definition, we have a semantics which may be more familiar:

$$\begin{aligned}
[\![x]\!]\sigma &= \sigma x \\
[\![MN]\!]\sigma &= \mathsf{apply}([\![M]\!]\sigma)([\![N]\!]\sigma) \\
[\![\lambda x\,.\,M]\!]\sigma &= \mathsf{fold}(\mathsf{lift}(\mathsf{fn}\,x[\![M]\!]\sigma)) \\
[\![\mathsf{P}\,MN]\!]\sigma &= \mathsf{fork}([\![M]\!]\sigma)([\![N]\!]\sigma)
\end{aligned}$$

EXAMPLES.

- $[\![\mathsf{I}\,M]\!] = [\![M]\!]$.
- $[\![\mathsf{K}\,MN]\!] = [\![M]\!]$.
- $[\![\mathsf{Y}\,M]\!]\sigma$ is the least solution of $a = \mathsf{apply}([\![M]\!]\sigma)a$.
- $[\![\Omega]\!] = \bot$.
- $[\![\Upsilon]\!] = \top$.
- $[\![(\lambda^v x\,.\,M)N]\!]\sigma$ is $\bot$ if $[\![N]\!]\sigma = \bot$, and $[\![(\lambda x\,.\,M)N]\!]\sigma$ otherwise.
- $[\![\mathsf{J}\,MN]\!] = [\![M]\!] \vee [\![N]\!]$.
- $[\![\mathsf{M}\,MN]\!] = [\![M]\!] \wedge [\![N]\!]$.
- $[\![\mathsf{A}\,MNO]\!]\sigma$ is $\bot$ if $[\![O]\!]\sigma = \bot$, and $[\![N(OM)]\!]\sigma$ otherwise. □

PROPOSITION 1. $[\![M[\rho]]\!] = [\![M]\!] \circ [\![\rho]\!]$

PROOF. An induction on M. □

2.4 Program logic

In order to show that **D** is fully abstract, we need to find a link between the denotational and operational semantics. We will use a *program logic* Φ, with propositions:

- ω, which is satisfied by any closed term.
- $\phi \wedge \psi$, which is satisfied by any term that satisfies ϕ and ψ.
- $\phi \to \psi$, which is satisfied by any term that converges, and that when applied to any term satisfying ϕ the result satisfies ψ.

For example, a closed term satisfies $\gamma = \omega \to \omega$ iff it converges. The definition of 'satisfaction' can be generalized to open terms by saying that M satisfies ϕ in the *context* $(x_1 : \phi_1, \ldots, x_n : \phi_n)$ iff M satisfies ϕ whenever x_i is bound to a term satisfying ϕ_i.

DEFINITION. Φ is defined as:

$$\phi ::= \omega \mid \phi \wedge \phi \mid \phi \to \phi$$

For closed terms M, $\models M : \phi$ is defined by axiom:

$$(\omega\mathrm{I}) \quad \models M : \omega$$

and structural rules:

$$(\wedge\mathrm{I})\ \frac{\models M : \phi \quad \models M : \psi}{\models M : \phi \wedge \psi} \qquad (\to\mathrm{I})\ \frac{M{\Downarrow} \quad \forall N\,.\,\models N : \phi \Rightarrow \models MN : \psi}{\models M : \phi \to \psi}$$

A *context* Γ is a list $x_1 : \phi_1, \ldots, x_n : \phi_n$ with distinct x_i.

- Let $\mathsf{wv}(x_1 : \phi_1, \ldots, x_n : \phi_n) = \{x_1, \ldots, x_n\}$.
- Let $(\Gamma, x : \phi, \Delta)(x) = \phi$, and $\Gamma(x) = \omega$ when $x \notin \mathsf{wv}\,\Gamma$.
- Let $\models \rho : \Gamma$ iff $\forall x\,.\,\models \rho(x) : \Gamma(x)$.

Then $\Gamma \models M : \phi$ iff $\forall \rho\,.\,(\models \rho : \Gamma) \Rightarrow (\models M[\rho] : \phi)$. □

In addition to the operational interpretation of Φ, we can provide a denotational interpretation, by giving a semantics $[\![\phi]\!]$ in **D** for each proposition ϕ.

- The semantics of ω is $\bot$.
- The semantics of $\phi \wedge \psi$ is the join of ϕ and ψ.
- The semantics of $\phi \to \psi$ is a function which returns ψ whenever it is applied to an element that satisfies ϕ.

For example, $[\![\gamma]\!] = [\![\mathsf{K}\,\Omega]\!] = \bot \mapsto \bot$. Note that this relied on any a and b from **D** having a join $a \vee b$. This will be shown in Section 2.5.

DEFINITION. $[\![\phi]\!]$ in **D** is defined:

$$\begin{aligned}
[\![\omega]\!] &= \bot \\
[\![\phi \wedge \psi]\!] &= [\![\phi]\!] \vee [\![\psi]\!] \\
[\![\phi \to \psi]\!] &= [\![\phi]\!] \mapsto [\![\psi]\!]
\end{aligned}$$

where (for ω-compact a and b, defined in Section 2.6):

$$(a \Rightarrow b)c = \begin{cases} b & \text{if } a \leq c \\ \bot & \text{otherwise} \end{cases}$$

$$a \mapsto b = \mathsf{fold}(\mathsf{lift}(a \Rightarrow b))$$

The environment $[\![\Gamma]\!]$ is defined as $[\![\Gamma]\!]x = [\![\Gamma(x)]\!]$. □

For each proposition ϕ, we can also define a term M_ϕ with the same denotational semantics as ϕ. This is the core of the expressiveness result that allows us to show that **D** is fully abstract for Λ_{P}. Note that we use the P combinator in defining M_ϕ, and so this proof of full abstraction relies on the existence of P.

DEFINITION. Define M_ϕ as:

$$M_\omega = \Omega$$

$$M_{\phi\wedge\psi} = \mathsf{J}\, M_\phi M_\psi$$
$$M_{\omega\to\chi} = \mathsf{K}\, M_\chi$$
$$M_{(\phi\wedge\psi)\to\chi} = \mathsf{M}\, M_{\phi\to\chi} M_{\psi\to\chi}$$
$$M_{(\phi\to\psi)\to\chi} = \mathsf{A}\, M_\phi M_{\psi\to\chi}$$

We can show by induction on ϕ that $[\![M_\phi]\!]\sigma = [\![\phi]\!]$. □

A third interpretation of Φ is as a proof system for propositions $\Gamma \vdash M : \phi$. This is first given as a preorder $\vdash \phi \le \psi$, which characterizes when ψ is a refinement of ϕ. In Section 2.6 we shall see that $\vdash \phi \le \psi$ iff $[\![\psi]\!] \le [\![\phi]\!]$.

DEFINITION. The preorder $\le$ is given by axioms:

$$\begin{array}{ll}
(\text{ID}) & \vdash \phi \le \phi \\
(\omega\text{I}) & \vdash \phi \le \omega \\
(\wedge\text{E}a) & \vdash \phi\wedge\psi \le \phi \\
(\wedge\text{E}b) & \vdash \phi\wedge\psi \le \psi \\
(\to\omega) & \vdash \phi\to\omega \le \omega\to\omega \\
(\to\wedge) & \vdash (\phi\to\psi)\wedge(\phi\to\chi) \le \phi\to(\psi\wedge\chi)
\end{array}$$

and structural rules:

$$(\text{TRANS})\ \frac{\vdash \phi \le \psi \le \chi}{\vdash \phi \le \chi} \qquad (\wedge\text{I})\ \frac{\vdash \phi \le \psi \quad \vdash \phi \le \chi}{\vdash \phi \le (\psi\wedge\chi)}$$
$$(\to\le)\ \frac{\vdash \phi' \le \phi \quad \vdash \psi \le \psi'}{\vdash (\phi\to\psi) \le (\phi'\to\psi')}$$

Let $\vdash \phi = \psi$ iff $\vdash \phi \le \psi \le \phi$ and $\vdash \Gamma \le \Delta$ iff $\forall x.\, \vdash \Gamma(x) \le \Delta(x)$. □

For example, we can show that $\wedge$ is commutative, associative, idempotent and has unit ω in the equivalence $\vdash \phi = \psi$. The partial order $\vdash \phi \le \psi$ is used in defining the proof system $\Gamma \vdash M : \phi$, since all of the structural rules (such as CUT, WEAKENING and CONTRACTION) can be given by one rule $(\le)$. The proof system induces a preorder on terms given by $M \sqsubseteq_S N$ iff N satisfies any property that M satisfies.

DEFINITION. The proof system $\Gamma \vdash M : \phi$ is given by axioms:

$$\begin{array}{ll}
(\omega\text{I}) & \vdash M : \omega \\
(\text{ID}) & x : \phi \vdash x : \phi
\end{array}$$

and structural rules:

$$(\wedge\text{I})\ \frac{\Gamma \vdash M : \phi \quad \Gamma \vdash M : \psi}{\Gamma \vdash M : (\phi\wedge\psi)} \qquad (\le)\ \frac{\vdash \Gamma \le \Delta \quad \Delta \vdash M : \phi \quad \vdash \phi \le \psi}{\Gamma \vdash M : \psi}$$
$$(\to\text{E})\ \frac{\Gamma \vdash M : \phi\to\psi \quad \Gamma \vdash N : \phi}{\Gamma \vdash MN : \psi} \qquad (\to\text{I})\ \frac{\Gamma, x : \phi \vdash M : \psi}{\Gamma \vdash \lambda x.\, M : \phi\to\psi}$$
$$(\mathsf{P}a)\ \frac{\Gamma \vdash M : \gamma}{\Gamma \vdash \mathsf{P}MN : \phi\to\phi} \qquad (\mathsf{P}b)\ \frac{\Gamma \vdash N : \gamma}{\Gamma \vdash \mathsf{P}MN : \phi\to\phi}$$

Then $M \sqsubseteq_S N$ iff $\Gamma \vdash M : \phi \Rightarrow \Gamma \vdash N : \psi$ for all Γ and ϕ. □

EXAMPLES.

- $\vdash \mathsf{I} : \phi\to\phi$.
- $\vdash \mathsf{K} : \phi\to\psi\to\phi$.
- $\Gamma \vdash \mathsf{Y}M : \phi$ iff $\vdash \phi = \omega$ or $\Gamma \vdash M : \psi\to\phi$ and $\Gamma \vdash \mathsf{Y}M : \psi$.
- $\vdash \Omega : \omega$.
- $\vdash \Upsilon : \phi$.
- If $\Gamma, x : \phi \vdash M : \psi$ then $\Gamma \vdash \lambda^v x.\, M : (\phi\wedge\gamma)\to\psi$.
- $\vdash \mathsf{J} : \phi\to\psi\to(\phi\wedge\psi)$.
- $\vdash \mathsf{M} : \phi\to\phi\to\phi$.
- $\vdash \mathsf{A} : \phi\to(\psi\to\chi)\to(\phi\to\psi)\to\chi$. □

In Section 2.7 we show that the problem of full abstraction reduces to one of showing that $\Gamma \vdash M : \phi$ iff $\Gamma \models M : \phi$ iff $[\![\phi]\!] \le [\![M]\!][\![\Gamma]\!]$.

2.5 Categorical presentation of D

In Section 2.3 we asserted the existence of a domain $\mathbf{D} \simeq (\mathbf{D}\to\mathbf{D})_\perp$ which we used to give the denotational semantics for Λ_P. In this section we shall justify this assertion, by showing that such a domain must exist. This section is a summary of PIERCE's (1991) summary of PLOTKIN's (1983) *Pisa Notes*, and can be omitted by readers familiar with domain theory.

The reason why we need a domain isomorphic to its own function space is because of terms like $\mathsf{Y}M$ which provide a means of defining recursive functions. We said that the semantics of $\mathsf{Y}M$ was:

$$[\![\mathsf{Y}M]\!]\sigma \text{ is the least solution of } a = \mathsf{apply}([\![M]\!]\sigma)a$$

To show that such a solution must exist, we present it as the limit of the sequence $a_0 \le a_1 \le \cdots$ where:

$$a_0 = \perp \qquad a_{n+1} = \mathsf{apply}([\![M]\!]\sigma)a_n$$

That is:

$$a_n = (\mathsf{apply} \circ [\![M]\!]\sigma)^n \perp$$

However, we cannot always find a fixed point to a function f by defining a to be the limit of the sequence $f^n\perp$. For example if we define the function odd on the real interval $[0,1]$ as:

$$\mathsf{odd}\,x = \begin{cases} (1+x)/4 \text{ if } x < \frac{1}{2} \\ (1+x)/2 \text{ otherwise} \end{cases}$$

then the sequence $\mathsf{odd}^n 0$ is $0, \frac{1}{4}, \frac{3}{8}, \ldots$ which has limit $\frac{1}{2}$, but this is not a fixed point of odd since $\mathsf{odd}(\frac{1}{2}) = \frac{3}{4}$. In order to bar functions like this, we shall restrict

ourselves to ω-*continuous* functions, that is if:

$$a \text{ is the limit of } a_0 \leq a_1 \leq \cdots$$

then:

$$fa \text{ is the limit of } fa_0 \leq fa_1 \leq \cdots$$

For example, this bars the odd function since:

$$\tfrac{1}{2} \text{ is the limit of } 0 \leq \tfrac{1}{4} \leq \tfrac{3}{8} \leq \cdots$$

but:

$$\tfrac{3}{4} \text{ is not the limit of } \tfrac{1}{4} \leq \tfrac{3}{8} \leq \tfrac{7}{16} \leq \cdots$$

We define the denotational semantics of Λ_{P} in $\mathbf{D} \simeq (\mathbf{D} \to \mathbf{D})_\perp$. To show that such a $\mathbf{D}$ must exist, we present it as the limit of a sequence of finite domains $\mathbf{D}_0, \mathbf{D}_1, \ldots$ where:

$$\mathbf{D}_0 = 1 \qquad \mathbf{D}_{n+1} = (\mathbf{D}_n \to \mathbf{D}_n)_\perp$$

This can also be presented as the fixed point of a *functor* F between domains:

$$F\mathbf{D}_i = (\mathbf{D}_i \to \mathbf{D}_i)_\perp = \mathbf{D}_{i+1}$$

Then in order to show that $\mathbf{D}$ exists, we show that F is continuous. In order to do this, we present:

- A notion of *domain*, such that the one-point domain 1 is a domain, and F is a functor between domains.
- A notion of *order* between domains with least element 1 and where every chain of domains has a limit.
- A notion of *continuous functor* between domains, such that F is continuous.

Following PLOTKIN (1983), we will use the *category of* ω*-cpo's with embeddings* as the appropriate notion of ordered domains. Since F is a continuous functor, it must have a least fixed point, which we will use as our definition of $\mathbf{D}$.

The rest of this section will present the technical details of this construction. We shall begin with a short reminder of some simple category theory. Interested readers should consult MAC LANE's (1971) or PIERCE's (1991) textbooks.

DEFINITION. A *category* C is:

- a class of *objects* $\mathsf{obj}\,\mathrm{C}$.
- a class of *arrows* $\mathsf{arr}\,\mathrm{C}$.
- a *domain* object $\mathsf{dom}\,f$ for each arrow f.
- a *codomain* object $\mathsf{cod}\,f$ for each arrow f.
- an *identity* arrow id_A for each object A.
- a *composite* arrow $f \circ g$ whenever $\mathsf{dom}\,f = \mathsf{cod}\,g$.

such that:

- $\mathsf{cod}(\mathsf{id}_A) = \mathsf{dom}(\mathsf{id}_A) = A$.
- $\mathsf{cod}(f \circ g) = \mathsf{cod}\,f$ and $\mathsf{dom}(f \circ g) = \mathsf{dom}\,g$.
- $\circ$ is associative with unit id.

We shall write:

- $A \,\mathsf{in}\, \mathrm{C}$ iff A is an object in C.
- $f : \mathsf{dom}\,f \to \mathsf{cod}\,f \,\mathsf{in}\, \mathrm{C}$ iff f is an arrow in C.

A category is *small* if $\mathsf{obj}\,\mathrm{C}$ and $\mathsf{arr}\,\mathrm{C}$ are sets. □

EXAMPLES. SET is the category where:

- objects are sets.
- arrows are functions.

A *preorder* is a small category where:

- objects are members of the preorder.
- for any objects A and B there is at most one arrow $f : A \to B$, and we write $A \leq B$ for $\exists f : A \to B$.

A *poset* is a preorder where if $f \circ g = \mathsf{id}$ then $f = g = \mathsf{id}$.

- 0 is the poset with no objects.
- 1 is the poset with one object 0.
- 2 is the poset with two objects $0 < 1$.
- ω is the poset with objects $0 < 1 < \cdots$
- $\omega + 1$ is the poset with objects $0 < 1 < \cdots < \omega$.

If C is a category then $\mathrm{C}_\perp$ is the category with:

- objects $\perp$ and $\mathsf{lift}\,A$ for each $A \,\mathsf{in}\, \mathrm{C}$.
- arrows $!A : \perp \to A$ and $\mathsf{lift}\,f : \mathsf{lift}\,A \to \mathsf{lift}\,B$ for each $f : A \to B \,\mathsf{in}\, \mathrm{C}$.

If C and D are categories then $\mathrm{C} \times \mathrm{D}$ is the category with:

- objects (A, B) for each $A \,\mathsf{in}\, \mathrm{C}$ and $B \,\mathsf{in}\, \mathrm{D}$.
- arrows $(f, g) : (A, B) \to (A', B')$ for each $f : A \to A' \,\mathsf{in}\, \mathrm{C}$ and $g : B \to B' \,\mathsf{in}\, \mathrm{D}$.

In each case, the domain, codomain, identity and composition should be evident. □

DEFINITION. A functor $F : \mathrm{C} \to \mathrm{D}$ has:

- an object $FA \,\mathsf{in}\, \mathrm{D}$ for each $A \,\mathsf{in}\, \mathrm{C}$.
- an arrow $Ff : FA \to FB \,\mathsf{in}\, \mathrm{D}$ for each $f : A \to B \,\mathsf{in}\, \mathrm{C}$.

such that:

- $F(\mathsf{id}_A) = \mathsf{id}_{FA}$.
- $F(f \circ g) = Ff \circ Fg$. □

EXAMPLES. $\mathsf{lift} : \mathrm{C} \to \mathrm{C}_\perp$ is a functor since we have:

- an object $\mathsf{lift}\,A$ in $\mathrm{C}_\perp$ for each A in C.
- an arrow $\mathsf{lift}\,f : \mathsf{lift}\,A \to \mathsf{lift}\,B$ for each $f : A \to B$ in C.

If $F : \mathrm{C} \to \mathrm{D}$ is a functor then $F_\perp : \mathrm{C}_\perp \to \mathrm{D}_\perp$ is the functor with:

- objects $F_\perp \perp = \perp$ and $F_\perp(\mathsf{lift}\,A) = \mathsf{lift}(FA)$.
- arrows $F_\perp(!A) = !(FA)$ and $F_\perp(\mathsf{lift}\,f) = \mathsf{lift}(F\,f)$.

If C and D are posets, then $F : \mathrm{C} \to \mathrm{D}$ is a functor iff F is a monotone function. Let POSET be the category with:

- objects are posets.
- arrows are monotone functions.

In each case the identity and composition properties should be evident. □

DEFINITION. $\perp$ is the initial object of C iff there is a unique arrow $!A : \perp \to A$ for every object A in C. □

EXAMPLES. Many of these categorical definitions have POSET equivalents:

- $\emptyset$ is the initial object of SET.
- 0 is the initial object of POSET.
- The initial object of a poset is its least element.
- A poset has an initial object iff it is pointed.
- The initial object of $\mathrm{C}_\perp$ is $\perp$. □

DEFINITION.

- An ω-*chain* in C is a set of objects $\{A_i \text{ in } \mathrm{C} \mid i \text{ in } \omega\}$ and a set of arrows $\{f_i^j : A_i \to A_j \text{ in } \mathrm{C} \mid i \le j \text{ in } \omega\}$ such that $f_j^k \circ f_i^j = f_i^k$.
- A *cocone* of such an ω-chain is an object A in C and a set of arrows of the form $\{f_i : A_i \to A \text{ in } \mathrm{C} \mid i \text{ in } \omega\}$ such that $f_j \circ f_i^j = f_i$.
- The *colimit* of such an ω-chain is a cocone $\{f_i : A_i \to A \text{ in } \mathrm{C} \mid i \text{ in } \omega\}$ such that for any other cocone $\{g_i : A_i \to B \text{ in } \mathrm{C} \mid i \text{ in } \omega\}$ there is a unique $f : A \to B$ in C such that $f \circ f_i = g_i$.
- A category has all ω-colimits iff every ω-chain has a colimit.
- $F : \mathrm{C} \to \mathrm{D}$ preserves ω-colimits iff whenever $\{f_i : A_i \to A\}$ is the colimit of $\{f_i^j : A_i \to A_j\}$ then $\{F\,f_i : FA_i \to FA\}$ is the colimit of $\{F\,f_i^j : FA_i \to FA_j\}$.
- $A \simeq B$ in C iff we can find $f : A \to B$ in C and $g : B \to A$ in C such that $f \circ g = \mathsf{id}$ and $g \circ f = \mathsf{id}$.
- A in C is the initial fixed point of $F : \mathrm{C} \to \mathrm{C}$ iff $A \simeq FA$ and for any other $B \simeq FB$ there is a unique $f : A \to B$. □

EXAMPLES. Many of these categorical definitions have POSET equivalents:

- An ω-chain is a set of elements $\{x_0 \le x_1 \le \cdots\}$.

- A cocone of an ω-chain is an upper bound.
- The colimit of an ω-chain C is its join (or least upper bound) $\bigvee C$.
- A pointed poset has all ω-colimits iff it is an ω-cpo.
- A function preserves ω-colimits iff it is ω-continuous.
- $x \simeq y$ iff $x \le y \le x$, that is iff $x = y$.
- The initial fixed point of a function is its least fixed point.

For example, $\omega + 1$ is an ω-cpo, but ω is not, since the ω-chain $\{0 \le 1 \le 2 \le \cdots\}$ has no least upper bound. Let ωCPO be the category with:

- ω-cpo's as objects.
- ω-continuous functions as arrows. □

PROPOSITION 2. *If* C *has an initial object and all* ω*-colimits, then any functor* $F : \mathrm{C} \to \mathrm{C}$ *which preserves* ω*-colimits has an initial fixed point.*

PROOF. Let C be the ω-chain:

$$\{F^{(j-i)}(!(F^i\perp)) : F^i\perp \to F^j\perp \mid i \le j\}$$

An adaptation of the usual proof of TARSKI's fixed point theorem shows that the colimit of C is the initial fixed point of F. For a discussion of TARSKI's fixed point theorem, see a textbook such as (DAVEY and PRIESTLEY, 1990). See also (LASSEZ *et al.*, 1982) for a short discussion of the history of fixed point theorems. □

This allows us to find the fixed point of any functor that preserves ω-colimits of a category with an initial object. Unfortunately, ωCPO does not have an initial object, and there is no obvious definition of a 'function space' functor $(\to) : \omega\mathrm{CPO}^2 \to \omega\mathrm{CPO}$. However $(\to)$ can be defined in the subcategory of ωCPO where all the arrows are *embeddings*, so we shall use this as our category for solving domain equations:

DEFINITION. An *embedding* is an arrow $e : A \to B$ in ωCPO such that we can find $e^R : B \to A$ in ωCPO with:

$$e \circ e^R \le \mathsf{id} \qquad e^R \circ e = \mathsf{id}$$

Let ωCPOE be the category with:

- ω-cpo's as objects.
- embeddings as arrows. □

EXAMPLES. The identity function is an embedding, with:

$$\mathsf{id}^R = \mathsf{id}$$

If $e : A \to B$ and $f : B \to C$ are embeddings, then $f \circ e : A \to C$ is the embedding with:

$$(f \circ e)^R = e^R \circ f^R$$

The arrow e^R is uniquely defined, so if $e : A \to B$ in CPOE and $f : B \to A$ in ωCPOE then:

$$(e \circ f \leq \mathsf{id}, f \circ e = \mathsf{id}) \text{ implies } e^R = f$$

$()_\perp : \omega\text{CPOE} \to \omega\text{CPOE}$ is the lifting functor with:

- $A_\perp$ in ωCPOE for A in ωCPOE.
- $e_\perp : A_\perp \to B_\perp$ in ωCPOE for $e : A \to B$ in ωCPOE.

$\Delta : \omega\text{CPOE} \to \omega\text{CPOE}^2$ is the diagonal functor with:

- $\Delta A = (A, A)$ in ωCPOE^2 for A in ωCPOE.
- $\Delta f = (f, f) : \Delta A \to \Delta B$ in ωCPOE^2 for $f : A \to B$ in ωCPOE.

$(\to) : \omega\text{CPOE}^2 \to \omega\text{CPOE}$ is the ω-continuous function space functor with:

- $(A \to B)$ in ωCPOE for (A, B) in ωCPOE.
- $(e \to f) : (A \to B) \to (A' \to B')$ in ωCPOE for $(e, f) : (A, B) \to (A', B')$ in ωCPOE^2.

where $e \to f$ is defined:

$$(e \to f)g = f \circ g \circ e^R$$
$$(e \to f)^R g = e \circ g \circ f^R$$

1 is the initial object in ωCPOE. □

DEFINITION. A cocone $\{e_i : A_i \to A \text{ in } \omega\text{CPOE} \mid i \text{ in } \omega\}$ is *determined* iff $\bigvee\{e_i \circ e_i^R \mid i \text{ in } \omega\} = \mathsf{id}$. □

PROPOSITION 3. *Any determined cocone is a colimit.*

PROOF. Let $\{e_i : A_i \to A \mid i \text{ in } \omega\}$ be a determined cocone of an ω-chain $\{e_i^j : A_i \to A_j \mid i \leq j \text{ in } \omega\}$. Then for any other cocone $\{f_i : A_i \to B \mid i \text{ in } \omega\}$, define $g : A \to B$ as:

$$g = \bigvee\{f_i \circ e_i^R \mid i \text{ in } \omega\}$$
$$g^R = \bigvee\{e_i \circ f_i^R \mid i \text{ in } \omega\}$$

Then we can show that g is the unique embedding such that $g \circ e_i = f_i$. Thus $\{e_i : A_i \to A \mid i \text{ in } \omega\}$ is a colimit. □

PROPOSITION 4. *Any ω-chain in ωCPOE has a determined cocone.*

PROOF. Let $\{e_i^j : A_i \to A_j \mid i \leq j\}$ be an ω-chain. An *instantiation* of this chain is a function f such that:

$$\mathsf{dom}\, f = \omega \qquad fi \in A_i \qquad e_i^{jR}(fj) = fi$$

then define:

$$A = \{f \mid f \text{ is an instantiation}\}$$

with the pointwise ordering. This is an ω-cpo, with join:

$$\bigvee\{f_i \mid i \text{ in } \omega\}\, j = \bigvee\{f_i j \mid i \text{ in } \omega\}$$

Then define:

$$e_i a j = \begin{cases} e_i^j a & \text{if } i \leq j \\ e_j^{iR} a & \text{otherwise} \end{cases}$$
$$e_i^R f = fi$$

We can show that $\{e_i : A_i \to A \mid i \text{ in } \omega\}$ is a determined cocone. □

DEFINITION. **D** is the determined colimit of the ω-chain:

$$\mathbf{D}_0 = 1$$
$$\mathbf{D}_{i+1} = (\mathbf{D}_i \to \mathbf{D}_i)_\perp$$

with $e_i : \mathbf{D}_i \to \mathbf{D}$ in ωCPOE given by Proposition 4. Then **D** is the initial fixed point of the functor $()_\perp \circ (\to) \circ \Delta$ given by Proposition 2. □

2.6 Logical presentation of D

In Section 2.5, we gave an abstract presentation of **D**, using the category of ω-cpo's with embeddings. In this section, we provide a concrete presentation of **D**, similar to SCOTT's (1982) *information systems*. Following ABRAMSKY's (1991) *domain theory in logical form* we use the program logic Φ as an alternative presentation of **D**. In particular, we show that the ω-cpo of *filters* of Φ is equivalent to **D**.

DEFINITION. $\Psi \subseteq \Phi$ is a *filter* iff:

- $\omega \in \Psi$.
- If $\phi \in \Psi$ and $\vdash \phi \leq \psi$ then $\psi \in \Psi$.
- If $\phi, \psi \in \Psi$ then $\phi \wedge \psi \in \Psi$.

Let $\mathsf{Filt}\,\Phi$ be the ω-cpo of filters, ordered by $\subseteq$. □

Then we can show that $\mathsf{Filt}\,\Phi$ is isomorphic to **D**. In proving this, it is essential that **D** is *algebraic*, that is every element of **D** is determined by its ω-*compact* approximations.

DEFINITION. An element a is ω-*compact* iff, for any ω-chain C:

$$a \leq \bigvee C \text{ implies } \exists c \in C\,.\, a \leq c$$

Let $\mathsf{k}\,a = \{b \leq a \mid b \text{ is } \omega\text{-compact}\}$. **D** is *algebraic* iff every a is the join of $\mathsf{k}\,a$. □

The rest of this section shows that Φ precisely characterizes the ω-compact elements of **D**, and since **D** is algebraic, $\mathsf{Filt}\,\Phi$ is isomorphic to **D**, that is:

- $\vdash \phi \leq \psi$ iff $[\![\phi]\!] \leq [\![\psi]\!]$.
- a is ω-compact iff $\exists \phi \,.\, a = [\![\phi]\!]$.
- $\mathbf{D} \simeq \mathsf{Filt}\,\Phi$.

In Section 2.4 we gave a semantics $[\![\cdot]\!] : \Phi \to \mathbf{D}$, which assumed that every pair of elements in $\mathbf{D}$ had a join. We shall now show that this assumption is justified. In fact, we shall show that $\mathbf{D}$ is a *complete lattice*.

DEFINITION. $\mathbf{D}$ is a complete lattice iff every subset of $\mathbf{D}$ has a join. □

PROPOSITION 5. *$\mathbf{D}$ is a complete lattice.*

PROOF. We can show by induction on n that each $\mathbf{D}_n$ is a complete lattice, since $\mathbf{D}_0 = 1$ is a complete lattice, and $\mathbf{D}_{n+1}$ has join $\bigvee_{n+1}$ defined:

$$\bigvee_{n+1} A = \begin{cases} \bot & \text{if } A \subseteq \{\bot\} \\ \mathsf{lift}(\bigvee'_{n+1} A) & \text{otherwise} \end{cases}$$

where:

$$\bigvee'_{n+1} A b = \bigvee_n \{ f b \mid \mathsf{lift}\ f \in A \}$$

Then $\mathbf{D}$ has:

$$\bigvee A = \bigvee \{ e_n(\bigvee_n e_n^R[A]) \mid n \text{ in } \omega \}$$

From this we can show that $\mathbf{D}$ is a complete lattice. □

From the definition of $\bigvee_n$, apply respects arbitrary joins, that is:

$$\mathsf{apply}(\bigvee A) b = \bigvee \{ \mathsf{apply}\, a b \mid a \in A \} \tag{1}$$

but in general, ω-continuous functions do not necessarily respect arbitrary join, for example:

$$((a \vee b) \Rightarrow \top)(a \vee b) = \top \neq \bot = \bot \vee \bot = (((a \vee b) \Rightarrow \top) a) \vee (((a \vee b) \Rightarrow \top) b)$$

However, ω-continuous functions do respect countable *directed* joins.

DEFINITION. $A \subseteq \mathbf{D}$ is *directed* iff any $a_1, \ldots, a_n \in A$ have an upper bound in A. □

PROPOSITION 6. *If B is countable and directed then $f(\bigvee B) = \bigvee(f[B])$.*

PROOF. For any directed $B = \{b_i \mid i \text{ in } \omega\}$, let the ω-chain C be $c_i = b_0 \vee \cdots \vee b_n$. Then we can show that $f(\bigvee B) = f(\bigvee C)$ and $\bigvee(f[C]) = \bigvee(f[B])$, so the result follows from f being ω-continuous. □

We can then show that a is ω-compact iff there is some n such that a comes from $\mathbf{D}_n$, that is iff a has *depth n*.

DEFINITION. a has *depth n* iff $e_n(e_n^R a) = a$. □

PROPOSITION 7. *a is ω-compact iff a has depth n for some n.*

PROOF.

⇒ Since $\mathbf{D}$ is determined, $a \leq \bigvee \{ e_i(e_i^R a) \mid i \text{ in } \omega \}$, so since a is ω-compact there is an n such that $a \leq e_n(e_n^R a) \leq a$, so a has depth n.

⇐ If a has depth n and $a \leq \bigvee C$ for some ω-chain C then

$$a = e(e_n^R a) \leq e_n(e_n^R(\bigvee C)) = e_n(\bigvee(e_n^R[C]))$$

Since $\mathbf{D}_n$ is finite, $e_n^R[C]$ is finite. Since C is an ω-chain, $e_n^R[C]$ is an ω-chain. Since $e_n^R[C]$ is a finite ω-chain, it has a top $e_n^R b$ for some $b \in C$. Then $a \leq e_n(\bigvee(e_n^R[C])) = e_n(e_n^R b) \leq b$. Thus, a is ω-compact. □

We can use this to show that $\mathbf{D}$ is algebraic.

PROPOSITION 8. *$\mathbf{D}$ is algebraic.*

PROOF. By Proposition 7, $\mathsf{k}\, a = \{ e_i(e_i^R a) \mid i \text{ in } \omega \}$, so since $\mathbf{D}$ is determined, $a = \bigvee(\mathsf{k}\, a)$. □

In fact, we can prove a stronger statement than this, namely that $\mathbf{D}$ is *prime algebraic*, that is every element of $\mathbf{D}$ is determined by its ω-compact prime approximations.

DEFINITION. a is *prime* iff, for any finite $B \subseteq \mathbf{D}$:

$$a \leq \bigvee B \Rightarrow \exists b \in B \,.\, a \leq b$$

Let $\mathsf{kp}\, a = \{ b \in \mathsf{k}\, a \mid b \text{ is prime} \}$. $\mathbf{D}$ is *prime algebraic* iff every a is the join of $\mathsf{kp}\, a$. □

We can show that a is ω-compact prime iff $a = b \mapsto c$, b is ω-compact and c is ω-compact prime or $c = \bot$. For example, $\bot \mapsto \bot$ is prime, but $\bot$ is not, since $\bot \leq \bigvee \emptyset$.

PROPOSITION 9.

1. *If $b \neq \bot$ then $a \leq \mathsf{apply}\, b c$ iff $(c \mapsto a) \leq b$.*
2. *If $a \mapsto b$ has depth n then $b = \bot$ or a and b have depth $< n$.*
3. *$a = \bigvee \{ b \mapsto c \mid b \mapsto c \leq a \}$.*
4. *For any $B \neq \emptyset$, $a \mapsto \bigvee B = \bigvee \{ a \mapsto b \mid b \in B \}$.*
5. *$a \mapsto b$ is ω-compact.*
6. *If b is prime then $a \mapsto b$ is prime.*
7. *If a is ω-compact prime then $a = b \mapsto c$.*
8. *If $a \mapsto b$ is prime then $b = \bot$ or b is prime.*

Thus a is ω-compact prime iff $a = b \mapsto c$ and c is ω-compact prime or $c = \bot$.

PROOF.

1. Follows from the definition of $\mapsto$.

2. Follows from the definition of depth.
3. If $a = \bot$ then:
$$a = \bot = \bigvee \emptyset = \bigvee \{b \mapsto c \mid b \mapsto c \leq a\}$$
Otherwise, we can show that for any d:
$$\mathsf{apply}\, a d = \mathsf{apply}(\bigvee \{b \mapsto c \mid b \mapsto c \leq a\})d$$
and so $a = \bigvee \{b \mapsto c \mid b \mapsto c \leq a\}$.
4. If $a \mapsto b \leq \bigvee C$ for an ω-chain $C \subseteq \mathbf{D}$, then:
$$b = \mathsf{apply}(a \mapsto b)a \leq \mathsf{apply}(\bigvee C)a = \bigvee \{\mathsf{apply}\, c a \mid c \in C\}$$
Since b is ω-compact there is a $c \in C$ such that $b \leq \mathsf{apply}\, c a$ so:
$$a \mapsto b \leq a \mapsto \mathsf{apply}\, c a \leq c$$
Thus $a \mapsto b$ is ω-compact.
5. If $a \mapsto b \leq \bigvee A$ for a finite set $A \subseteq \mathbf{D}$, then:
$$b = \mathsf{apply}(a \mapsto b)a \leq \mathsf{apply}(\bigvee A)a = \bigvee \{\mathsf{apply}\, c a \mid c \in A\}$$
Since b is prime, there is a $c \in A$ such that $b \leq \mathsf{apply}\, c a$, so:
$$(a \mapsto b) \leq (a \mapsto \mathsf{apply}\, c a) \leq c$$
Thus $a \mapsto b$ is prime.
6. Let $A = \{b \mapsto c \mid b \mapsto c \leq a\}$, so $a = \bigvee A$. Let $a_0, a_1, \ldots$ be an enumeration of A, and let $C = \{c_i \mid i \text{ in } \omega\}$ be the ω-chain where $c_i = a_0 \vee \cdots \vee a_i$. Then since a is ω-compact and $a \leq \bigvee A = \bigvee C$ there is a $c_i \in C$ such that $a \leq c_i = a_0 \vee \cdots \vee a_i$. Since a is prime, there is a $j \leq i$ such that $a \leq a_j = b_j \mapsto c_j \leq a$.
7. For any c:
$$\begin{aligned}
&\mathsf{apply} \bigvee \{a \mapsto b \mid b \in B\} c \\
&\quad = \bigvee \{\mathsf{apply}(a \mapsto b)c \mid b \in B\} && \text{(Eqn 1)} \\
&\quad = \begin{cases} \bigvee B \text{ if } a \leq c \\ \bot \quad \text{otherwise} \end{cases} && \text{(Defn of } \mapsto\text{)} \\
&\quad = \mathsf{apply}(a \mapsto \bigvee B)c && \text{(Defn of } \mapsto\text{)}
\end{aligned}$$
Then since $B \neq \emptyset$, $\bigvee \{a \mapsto b \mid b \in B\} \neq \bot$, and so by part 1:
$$\bigvee \{a \mapsto b \mid b \in B\} = (a \mapsto \bigvee B)$$
8. Let $b = \bigvee B$ for finite B. If $B = \emptyset$ then $b = \bot$. Otherwise:
$$a \mapsto b = a \mapsto \bigvee B = \bigvee \{a \mapsto c \mid c \in B\}$$
so since $a \mapsto b$ is prime there is a $c \in B$ such that $a \mapsto b = a \mapsto c$ so $b = c$. Thus b is prime. □

We can use this to show that **D** is prime algebraic.

PROPOSITION 10. **D** *is prime algebraic.*

PROOF. Using Proposition 9 we can show by induction on the depth of a that for any ω-compact a that:
$$a = \bigvee \{b \mapsto c \mid b \mapsto c \leq a, c \text{ is prime}\} = \bigvee (\mathsf{kp}\, a)$$
Then since **D** is algebraic:
$$a = \bigvee (\mathsf{k}\, a) = \bigvee \{\bigvee (\mathsf{kp}\, b) \mid b \in \mathsf{k}\, a\} = \bigvee (\mathsf{kp}\, a)$$
Thus **D** is prime algebraic. □

We have shown that every ω-compact element is determined by its prime approximations, and so is of the form:
$$a = a_1 \mapsto b_1 \vee \cdots \vee a_n \mapsto b_n$$
Note that $\bot$ is covered by the case when $n = 0$. By examination of the semantics of Φ we can see that $[\![\phi]\!]$ can always be given in the form:
$$[\![\phi]\!] = [\![\phi_1 \to \psi_1 \wedge \cdots \wedge \phi_n \to \psi_n]\!]$$
Note that ω is covered by the case when $n = 0$. This allows us to show that our denotational semantics for Φ characterizes precisely the ω-compact elements of **D**. We will show this by proving a normal form result for propositions, using *factored* propositions for the normal form.

DEFINITION.

- ϕ is *factored* iff $\phi = \phi_1 \wedge \cdots \wedge \phi_n$ and each ϕ_i is prime.
- ϕ is *prime* iff $\phi = \gamma$ or $\phi = \psi \to \chi$, ψ is factored and χ is prime.

ϕ can be factored iff there is a factored ψ such that $\vdash \phi = \psi$. □

PROPOSITION 11.

1. *Any* ϕ *can be factored.*
2. $[\![\phi]\!]$ *is* ω*-compact*
3. *If* ϕ *is prime then* $[\![\phi]\!]$ *is prime.*
4. *If* a *is* ω*-compact prime then* $\exists$*prime* $\phi \,.\, a = [\![\phi]\!]$.
5. *If* a *is* ω*-compact then* $\exists$*factored* $\phi \,.\, a = [\![\phi]\!]$.

PROOF. Parts 1, 2 and 3 are an induction on ϕ. Parts 4 and 5 are an induction on the depth of a.

1. An induction on ϕ.
2. Follows from Proposition 9.
3. Follows from Proposition 9.

4. By Proposition 9, $a = b \mapsto c$ where c is prime or $c = \bot$. If $c = \bot$ then $[\![\gamma]\!] = \bot \mapsto \bot = b \mapsto \bot = a$. Otherwise, b and c have smaller depth than a, so by induction we can find factored ϕ and prime ψ such that $[\![\phi]\!] = b$ and $[\![\psi]\!] = c$. Then $[\![\phi \to \psi]\!] = b \mapsto c = a$.
5. Find ω-compact primes a_i such that $a = a_0 \vee \cdots \vee a_n$. By part 4 we can find prime ϕ_i such that $[\![\phi_i]\!] = a_i$. Then $[\![\phi_0 \wedge \cdots \wedge \phi_n]\!] = a$. □

We can then show that the inequational theory $\vdash \phi \leq \psi$ is sound and complete for the denotational semantics of Φ. This uses factored propositions as a normal form.

PROPOSITION 12. $\vdash \psi \leq \phi$ *iff* $[\![\phi]\!] \leq [\![\psi]\!]$.

PROOF.

SOUNDNESS ($\Rightarrow$). An induction on the proof of $\vdash \psi \leq \phi$.

COMPLETENESS ($\Leftarrow$). We first show by induction on ϕ and ψ that if ϕ and ψ are factored and $[\![\phi]\!] \leq [\![\psi]\!]$ then $\vdash \psi \leq \phi$. Then for any ϕ and ψ, by Proposition 11 we can find factored ϕ' and ψ' such that $\vdash \phi = \phi'$ and $\vdash \psi = \psi'$. Then by soundness, we have $\vdash \phi = \phi' \leq \psi' = \psi$. □

Finally, we can show that the filters of Φ form a concrete presentation of **D**.

PROPOSITION 13. $\mathbf{D} \simeq \mathsf{Filt}\,\Phi$.

PROOF. Define $\mathsf{con} : \mathbf{D} \to \mathsf{Filt}(\Phi)$ and $\mathsf{abs} : \mathsf{Filt}(\Phi) \to \mathbf{D}$ as:

$$\mathsf{con}\, d = \{\phi \mid [\![\phi]\!] \leq d\}$$
$$\mathsf{abs}\, \Psi = \bigvee\{[\![\psi]\!] \mid \psi \in \Psi\}$$

Then we can show that con and abs form an isomorphism. □

In particular, the semantics of any term M can be given in terms of the propositions that M satisfies.

PROPOSITION 14. $[\![M]\!]\sigma = \bigvee\{[\![\phi]\!] \mid [\![\phi]\!] \leq [\![M]\!][\![\Gamma]\!], [\![\Gamma]\!] \leq \sigma\}$.

PROOF. Follows from Proposition 13. □

2.7 Full abstraction

In this section we shall show that **D** is fully abstract for Λ_{P}. We shall do this in three parts:

- We show that $\Gamma \vdash M : \phi$ iff $[\![\phi]\!] \leq [\![M]\!][\![\Gamma]\!]$, thus showing that the proof system is sound and complete for the denotational semantics. This is Proposition 15.
- We then show that if $\Gamma \vdash M : \phi$ then $\Gamma \models M : \phi$, and that if $\Gamma \models M : \phi$ then $[\![\phi]\!] \leq [\![M]\!][\![\Gamma]\!]$. Thus the three presentations of the logic are equivalent. This is Proposition 18.

- Finally, we show that full abstraction is gained by proving the three logical presentations to be equivalent. This is Proposition 19.

The rest of this section provides proofs of these Propositions. This is a specific instance of ABRAMSKY's (1991) *domain theory in logical form*.

First, we give a sketch proof that the proof system for Λ_{P} is sound and complete for **D**.

PROPOSITION 15. $\Gamma \vdash M : \phi$ *iff* $[\![\phi]\!] \leq [\![M]\!][\![\Gamma]\!]$.

PROOF.

SOUNDNESS ($\Rightarrow$). An induction on the proof of $\Gamma \vdash M : \phi$.

COMPLETENESS ($\Leftarrow$). An induction on M. The difficult cases are:

- If $[\![\phi]\!] \leq [\![MN]\!][\![\Gamma]\!]$ then

$$\begin{array}{lll} [\![\phi]\!] & & \\ \quad \leq \mathsf{apply}([\![M]\!][\![\Gamma]\!])([\![N]\!][\![\Gamma]\!]) & & \text{(Hypothesis)} \\ \quad = \mathsf{apply}([\![M]\!][\![\Gamma]\!])(\bigvee\{[\![\phi]\!] \mid [\![\phi]\!] \leq [\![N]\!][\![\Gamma]\!]\}) & & (\mathbf{D} \text{ is algebraic}) \\ \quad = \bigvee\{\mathsf{apply}([\![M]\!][\![\Gamma]\!])[\![\psi]\!] \mid [\![\psi]\!] \leq [\![N]\!][\![\Gamma]\!]\} & & \text{(Continuity)} \end{array}$$

  so since $[\![\phi]\!]$ is compact there is a ψ such that:

$$[\![\psi]\!] \leq [\![N]\!][\![\Gamma]\!] \qquad [\![\phi]\!] \leq \mathsf{apply}([\![M]\!][\![\Gamma]\!])[\![\psi]\!]$$

  If $[\![M]\!][\![\Gamma]\!] = \bot$ then $\vdash \phi = \omega$ so $\Gamma \vdash MN : \phi$. Otherwise $[\![M]\!][\![\Gamma]\!] \neq \bot$ so $[\![\psi \to \phi]\!] \leq [\![M]\!][\![\Gamma]\!]$ so by induction $\Gamma \vdash N : \psi$ and $\Gamma \vdash M : \psi \to \phi$ and so $\Gamma \vdash MN : \phi$.

- If $[\![\phi]\!] \leq [\![\lambda x\,.\,M]\!][\![\Gamma]\!]$ then:

$$\begin{array}{lll} [\![\phi]\!] & & \\ \quad \leq [\![\lambda x\,.\,M]\!][\![\Gamma]\!] & & \text{(Hypothesis)} \\ \quad = \bigvee\{[\![\psi \to \chi]\!] \mid [\![\psi \to \chi]\!] \leq [\![\lambda x\,.\,M]\!][\![\Gamma]\!]\} & & \text{(Propn 9.3)} \\ \quad = \bigvee\{[\![\psi \to \chi]\!] \mid [\![\chi]\!] \leq \mathsf{apply}([\![\lambda x\,.\,M]\!][\![\Gamma]\!])[\![\psi]\!]\} & & \text{(Propn 9.1)} \\ \quad = \bigvee\{[\![\psi \to \chi]\!] \mid [\![\chi]\!] \leq [\![M]\!](\mathsf{update}[\![\Gamma]\!]x[\![\psi]\!])\} & & \text{(Defn of } [\![\lambda x\,.\,M]\!]) \\ \quad = \bigvee\{[\![\psi \to \chi]\!] \mid [\![\chi]\!] \leq [\![M]\!][\![\nu x\,.\,\Gamma, x : \psi]\!]\} & & \text{(Defn of update)} \end{array}$$

  Since ϕ is compact we can find ψ_i and χ_i such that

$$[\![\phi]\!] \leq [\![\psi_1 \to \chi_1]\!] \vee \cdots \vee [\![\psi_n \to \chi_n]\!] \qquad [\![\chi_i]\!] \leq [\![M]\!][\![\nu x\,.\,\Gamma, x : \psi_i]\!]$$

  Then by Proposition 12:

$$\vdash \psi_1 \to \chi_1 \wedge \cdots \wedge \psi_n \to \chi_n \leq \phi$$

  and so for each $1 \leq i \leq n$:

$$\begin{array}{ll} [\![\chi_i]\!] \leq [\![M]\!][\![\nu x\,.\,\Gamma, x : \psi_i]\!] & \\ \quad \Rightarrow \nu x\,.\,\Gamma, x : \psi_i \vdash M : \chi_i & \text{(Induction)} \end{array}$$

$$\Rightarrow \nu x\,.\,\Gamma \vdash \lambda x\,.\,M : \psi_i \to \chi_i \qquad (\to\text{I})$$
$$\Rightarrow \Gamma \vdash \lambda x\,.\,M : \psi_i \to \chi_i \qquad (\leq)$$

Thus by $(\wedge\text{I})$ and $(\leq)$, $\Gamma \vdash \lambda x\,.\,M : \phi$. □

This has tied together the denotational and proof theoretic presentations of the logic, and we can start to link these with the operational presentation. To begin with, we show that the denotational semantics respects the operational semantics (following BARENDREGT's definition of λ-*theory* we might call such a model a Λ_{P}-*theory*).

PROPOSITION 16. *If $M \to N$ then $[\![M]\!] = [\![N]\!]$*

PROOF. An induction on the proof of $M \to N$. □

This has an immediate corollary, which is that convergent terms do not have $\bot$ as their semantics.

PROPOSITION 17. *If $M{\Downarrow}$ then $[\![M]\!]\bot \neq \bot$.*

PROOF. If $M \Downarrow \lambda x\,.\,N$ then by Proposition 16, $[\![M]\!]\bot = [\![\lambda x\,.\,N]\!]\bot \neq \bot$. □

This is enough to show the equivalence of the three logical presentations. Note that relating the operational and denotational presentations requires the existence of the terms M_ϕ, and hence the P combinator.

PROPOSITION 18. $(\Gamma \vdash M : \phi) \Rightarrow (\Gamma \models M : \phi) \Rightarrow ([\![\phi]\!] \leq [\![M]\!][\![\Gamma]\!])$

PROOF.

SOUNDNESS $(1 \Rightarrow 2)$. An induction on the proof of $\Gamma \vdash M : \phi$.

COMPLETENESS $(2 \Rightarrow 3)$. We first show the case when M is closed, by induction on ϕ. The only difficult case is when $\models M : \phi \to \psi$, so $M{\Downarrow}$ and by Proposition 17 $[\![M]\!]\bot \neq \bot$ so:

$$\begin{array}{ll}
[\![M_\phi]\!] = [\![\phi]\!] & \\
\Rightarrow \vdash M_\phi : \phi & \text{(Propn 15)} \\
\Rightarrow \models M_\phi : \phi & \text{(Soundness)} \\
\Rightarrow MM_\phi : \psi & \text{(Defn of } \models\text{)} \\
\Rightarrow [\![\psi]\!] \leq [\![MM_\phi]\!]\bot & \text{(Induction)} \\
\Rightarrow [\![\psi]\!] \leq \mathsf{apply}([\![M]\!]\bot)[\![\phi]\!] & \text{(Defn of } [\![MN]\!]\text{)} \\
\Rightarrow [\![\phi]\!] \mapsto [\![\psi]\!] \leq [\![M]\!]\bot & \text{(Propn 9.1)} \\
\Rightarrow [\![\phi \to \psi]\!] \leq [\![M]\!]\bot & \text{(Defn of } [\![\phi \to \psi]\!]\text{)}
\end{array}$$

If M is open, and $\Gamma \models M : \phi$ then define ρ as $\rho(x) = M_{\Gamma(x)}$. Then:

$$\begin{array}{ll}
[\![\rho]\!]\bot = [\![\Gamma]\!] & \\
\Rightarrow \vdash \rho : \Gamma & \text{(Propn 15)} \\
\Rightarrow \models \rho : \Gamma & \text{(Soundness)}
\end{array}$$

$$\begin{array}{ll}
\Rightarrow \models M[\rho] : \phi & \text{(Defn of } \models\text{)} \\
\Rightarrow [\![\phi]\!] \leq [\![M[\rho]]\!]\bot & \text{(Above)} \\
\Rightarrow [\![\phi]\!] \leq [\![M]\!]([\![\rho]\!]\bot) & \text{(Propn 1)} \\
\Rightarrow [\![\phi]\!] \leq [\![M]\!][\![\Gamma]\!] & \text{(Defn of } \rho\text{)}
\end{array}$$

Thus we have completeness. □

We can then use the equivalence of the logical presentations to show full abstraction.

PROPOSITION 19. *$M \sqsubseteq_O N$ iff $M \sqsubseteq_S N$ iff $M \sqsubseteq_D N$.*

PROOF.

$(M \sqsubseteq_O N \Rightarrow M \sqsubseteq_D N)$ We first prove by structural induction on ϕ that if $M \sqsubseteq_O N$ and $[\![\phi]\!] \leq [\![M]\!][\![\Gamma]\!]$ then $[\![\phi]\!] \leq [\![N]\!][\![\Gamma]\!]$. The only difficult case is when we have $\phi = \psi \to \chi$, in which case $[\![N]\!][\![\Gamma]\!] \neq \bot$ and so:

$$\begin{array}{ll}
[\![\psi \to \chi]\!] \leq [\![M]\!][\![\Gamma]\!] & \\
\Rightarrow \mathsf{apply}[\![\psi \to \chi]\!][\![\psi]\!] \leq \mathsf{apply}([\![M]\!][\![\Gamma]\!])[\![\psi]\!] & \text{(Monotonicity)} \\
\Rightarrow [\![\chi]\!] \leq \mathsf{apply}([\![M]\!][\![\Gamma]\!])[\![\psi]\!] & \text{(Defn of } [\![\psi \to \chi]\!]\text{)} \\
\Rightarrow [\![\chi]\!] \leq \mathsf{apply}([\![M]\!][\![\Gamma]\!])([\![M_\psi]\!][\![\Gamma]\!]) & ([\![M_\phi]\!]\sigma = [\![\phi]\!]) \\
\Rightarrow [\![\chi]\!] \leq [\![MM_\psi]\!][\![\Gamma]\!] & \text{(Defn of } [\![MN]\!]\text{)} \\
\Rightarrow [\![\chi]\!] \leq [\![NM_\psi]\!][\![\Gamma]\!] & \text{(Induction)} \\
\Rightarrow [\![\chi]\!] \leq \mathsf{apply}([\![N]\!][\![\Gamma]\!])([\![M_\psi]\!][\![\Gamma]\!]) & \text{(Defn of } [\![MN]\!]\text{)} \\
\Rightarrow [\![\chi]\!] \leq \mathsf{apply}([\![N]\!][\![\Gamma]\!])[\![\psi]\!] & ([\![M_\phi]\!]\sigma = [\![\phi]\!]) \\
\Rightarrow [\![\psi \to \chi]\!] \leq [\![N]\!][\![\Gamma]\!] & \text{(Propn 9.1)}
\end{array}$$

Thus for any σ:

$$\begin{array}{ll}
[\![M]\!]\sigma & \\
= \bigvee\{[\![\phi]\!] \mid [\![\phi]\!] \leq [\![M]\!][\![\Gamma]\!], [\![\Gamma]\!] \leq \sigma\} & \text{(Propn 14)} \\
\leq \bigvee\{[\![\phi]\!] \mid [\![\phi]\!] \leq [\![N]\!][\![\Gamma]\!], [\![\Gamma]\!] \leq \sigma\} & \text{(Above)} \\
= [\![N]\!]\sigma & \text{(Propn 14)}
\end{array}$$

Thus if $M \sqsubseteq_O N$ then $M \sqsubseteq_D N$.

$(M \sqsubseteq_D N \Rightarrow M \sqsubseteq_O N)$ For any closing context C, if $M \sqsubseteq_D N$ then:

$$\begin{array}{ll}
C[M]{\Downarrow} & \\
\Rightarrow \models C[M] : \gamma & \text{(Defn of } \models\text{)} \\
\Rightarrow [\![\gamma]\!] \leq [\![C[M]]\!]\bot & \text{(Propn 18)} \\
\Rightarrow [\![\gamma]\!] \leq [\![C[N]]\!]\bot & \text{(Hypothesis)} \\
\Rightarrow \models C[N] : \gamma & \text{(Propn 18)} \\
\Rightarrow \models C[N]{\Downarrow} & \text{(Defn of } \models\text{)}
\end{array}$$

Thus if $M \sqsubseteq_D N$ then $M \sqsubseteq_O N$.

$(M \sqsubseteq_D N \Rightarrow M \sqsubseteq_S N)$ For any Γ and ϕ, if $M \sqsubseteq_D N$ then:

$$\begin{array}{lr}
\Gamma \vdash M : \phi & \\
\quad \Rightarrow [\![\phi]\!] \leq [\![M]\!][\![\Gamma]\!] & \text{(Propn 18)} \\
\quad \Rightarrow [\![\phi]\!] \leq [\![N]\!][\![\Gamma]\!] & \text{(Hypothesis)} \\
\quad \Rightarrow \Gamma \vdash M : \phi & \text{(Propn 18)}
\end{array}$$

Thus if $M \sqsubseteq_D N$ then $M \sqsubseteq_S N$.

$(M \sqsubseteq_S N \Rightarrow M \sqsubseteq_D N)$ For any σ, if $M \sqsubseteq_S N$ then:

$$\begin{array}{lr}
[\![M]\!]\sigma & \\
\quad = \bigvee\{[\![\phi]\!] \mid [\![\phi]\!] \leq [\![M]\!][\![\Gamma]\!], [\![\Gamma]\!] \leq \sigma\} & \text{(Propn 14)} \\
\quad = \bigvee\{[\![\phi]\!] \mid \Gamma \vdash M : \phi, [\![\Gamma]\!] \leq \sigma\} & \text{(Propn 18)} \\
\quad \leq \bigvee\{[\![\phi]\!] \mid \Gamma \vdash N : \phi, [\![\Gamma]\!] \leq \sigma\} & \text{(Hypothesis)} \\
\quad = \bigvee\{[\![\phi]\!] \mid [\![\phi]\!] \leq [\![N]\!][\![\Gamma]\!], [\![\Gamma]\!] \leq \sigma\} & \text{(Propn 18)} \\
\quad = [\![N]\!]\sigma & \text{(Propn 14)}
\end{array}$$

Thus if $M \sqsubseteq_S N$ then $M \sqsubseteq_D N$. □

Thus we have shown that **D** is fully abstract for leftmost-outermost tree reduction of the untyped λ-calculus with P.

3 Graph reduction

In this Chapter we present a formal model of concurrent graph reduction. To do this, we:

- Define an untyped λ-calculus with recursive declarations.
- Show that recursive declarations can be regarded as graphs.
- Provide an operational semantics for declarations, based on BERRY and BOUDOL's (1990) *Chemical Abstract Machine*.
- Define a denotational semantics and a program logic.
- Show how the proof techniques from Chapter 2 can be adapted to show the denotational semantics to be fully abstract.

In doing so, we need to show some operational properties about concurrent graph reduction:

- Although concurrent graph reduction is not confluent, we can find a semantically equivalent reduction strategy which is confluent.
- We can show that the concurrent behaviour of our graph reduction model is unimportant, by showing a semantically equivalent reduction strategy which models one-processor execution.
- We can show *referential transparency* for the operational semantics, using *simulation* between graphs.

Thus, the fully abstract model for the λ-calculus with P is also fully abstract for the λ-calculus with recursive declarations.

3.1 The λ-calculus with recursive declarations

The λ-calculus with recursive declarations is an extension of the λ-calculus with P, to include mutually recursive declarations such as:

$$\mathsf{rec}(x := ?M, y := ?N) \text{ in } x@y$$

which means declare x to be M, declare y to be N, and apply x to y. Terms from the λ-calculus with rec are:

- ∇x is an *indirection* pointing to x.
- $x@y$ is an *application* applying the function pointed to by x to the argument pointed to by y.
- $x \vee y$ is a *fork* which evaluates the terms pointed to by x and y and returns the identity function if one of them reaches weak head normal form. Semantically, this is $\mathsf{P}\,xy$ from Λ_{P}.
- $\lambda x\,.\,M$ is an *abstraction*.

- $\mathsf{rec} D \,\mathsf{in}\, M$ is a *recursive declaration* of D in M.

Recursive declarations are:

- $x := !M$ is an *tagged node* declaring x to be M, and that M should be evaluated immediately.
- $x := ?M$ is an *untagged node* declaring x to be M, and that M should not be evaluated until it is needed.
- ε is the *empty* declaration.
- D, E is the *concatenated* declaration of D and E.
- $\nu x \,.\, D$ is the declaration D with a *local variable* x.

For example, the term:

$$\mathsf{rec} x := ?M, y := ?N \,\mathsf{in}\, x@y$$

declares x to be M and y to be N, then applies x to y. This can be contrasted with the term:

$$\mathsf{rec} x := !M, y := !N \,\mathsf{in}\, x@y$$

which is semantically equivalent, but allows evaluation of M and N to be performed concurrently. This is similar to the annotation of nodes described by PEYTON JONES (1987, Ch. 24). In the declaration:

$$x_1 := !M_1, \ldots, x_m := !M_m, y_1 := ?N_1, \ldots, y_n := ?N_n$$

the terms M_i are tagged, and so they can all be evaluated concurrently, whereas the terms N_i are untagged, and so are evaluated when they are needed. All declarations are considered to be recursive, for example:

$$x := !\lambda y \,.\, x$$

declares a term which reaches weak head normal form, is given an argument, and returns itself. It has the same semantics as the ogre Υ in Λ_{P}.

We have allowed local variables in declarations, for example, the local declaration $\mathsf{local}\, x = ?M \,\mathsf{in}\, y = ?N$ can be implemented as:

$$\nu x \,.\, (x := ?M, y := ?N)$$

We will see below how this can be generalized, so we can define $\mathsf{local} D \,\mathsf{in}\, E$ in this language. The handling of local variables here is similar to *scope* in MILNER's (1991) polyadic π-calculus, and indeed has a very similar operational semantics.

We can think of declarations as a variant of HUGHES' (1984) *supercombinator* code. For example, the supercombinator code:

$x = \lambda w \,.\, ww$ $y = M$
$\$\mathsf{PROG} = xy$

can be given as the declaration:

$$\nu xy \,.\, (\$\mathsf{PROG} := !xy, x := ?\lambda w \,.\, ww, y := ?M)$$

In summary:

DEFINITION. Lam and Dec are defined:

$$M ::= \nabla x \mid x@y \mid x \vee y \mid \lambda x \,.\, M \mid \mathsf{rec} D \,\mathsf{in}\, M$$
$$D ::= x := !M \mid x := ?M \mid \varepsilon \mid D, D \mid \nu x \,.\, D$$

Let $D = E$ mean D and E are syntactically identical. □

EXAMPLES. Given a vector $\vec{x} = x_1 \ldots x_n$, we can define:

$$\nu \vec{x} \,.\, D = \nu x_1 \,.\, \ldots \,.\, \nu x_n \,.\, D$$

We can implement a 'black hole' term as:

$$\mho = \mathsf{rec} x := !\nabla x \,\mathsf{in}\, x$$

We can implement the λ-calculus with P as (for fresh x and y):

$$x = \nabla x$$
$$MN = \mathsf{rec} x := !M, y := ?N \,\mathsf{in}\, x@y$$
$$\mathsf{P}MN = \mathsf{rec} x := !M, y := !N \,\mathsf{in}\, x \vee y$$

We shall see later that this has the same semantics as Λ_{P}. □

Unfortunately, at the moment, there is nothing to prevent inconsistent declarations such as:

$$x := !M, x := !N$$

or declarations with dangling pointers such as:

$$\nu y \,.\, (x := !\nabla y)$$

We would like to avoid such terms, since their semantics is by no means obvious. We will achieve this by restricting our attention to *well-formed* expressions, with no inconsistency or dangling pointers.

DEFINITION. The *written* variables of a declaration are:

$$\mathsf{wv}(x := M) = \{x\} \qquad \mathsf{wv}\,\varepsilon = \emptyset$$
$$\mathsf{wv}(D, E) = \mathsf{wv} D \cup \mathsf{wv} E \qquad \mathsf{wv}(\nu x \,.\, D) = \mathsf{wv} D \setminus \{x\}$$

An expression is *well-formed* iff:

- every subexpression of the form D, E has $\mathsf{wv} D \cap \mathsf{wv} E = \emptyset$.
- every subexpression of the form $\nu x \,.\, D$ has $x \in \mathsf{wv} D$.

From now on, we shall only consider well-formed expressions. □

EXAMPLES.

- $x := !M, y := !N$ is well-formed.
- $x := !M, x := !N$ is not.
- $x := !M, \nu x\,.\,(x := !N)$ is well-formed.
- $\nu x\,.\,(x := !M, x := !N)$ is not.
- $x := !M, \nu x\,.\,(y := !N)$ is not. □

Similarly, we can define the read variables and free variables of an expression.

DEFINITION. The *read* variables of an expression are:

$$\begin{aligned} \mathsf{rv}(\nabla x) &= \{x\} & \mathsf{rv}(x@y) &= \{x,y\} \\ \mathsf{rv}(x \vee y) &= \{x,y\} & \mathsf{rv}(\lambda x\,.\,M) &= \mathsf{rv}\,M \setminus \{x\} \\ \mathsf{rv}(\mathsf{rec}\,D\,\mathsf{in}\,M) &= (\mathsf{rv}\,M \cup \mathsf{rv}\,D) \setminus \mathsf{wv}\,D \\ \mathsf{rv}(x := M) &= \mathsf{rv}\,M & \mathsf{rv}\,\varepsilon &= \emptyset \\ \mathsf{rv}(D,E) &= \mathsf{rv}\,D \cup \mathsf{rv}\,E & \mathsf{rv}(\nu x\,.\,D) &= \mathsf{rv}\,D \setminus \{x\} \end{aligned}$$

The *free* variables of an expression are:

$$\mathsf{fv}\,M = \mathsf{rv}\,M \quad \mathsf{fv}\,D = \mathsf{rv}\,D \cup \mathsf{wv}\,D$$

A declaration is *closed* iff $\mathsf{rv}\,D \subseteq \mathsf{wv}\,D$. □

In implementation terms, the read variables of a declaration are the pointers leading out of it, and the written variables are pointers leading into it. For example, x is a pointer into $x := !\nabla y$ and y is a pointer out of it.

DEFINITION. A *renaming* is a function $\rho : V \to V$ which is almost everywhere the identity.

- Let $M[\rho]$ be M with any read variable x replaced by ρx.
- Let $D[\rho]$ be D with any read variable x replaced by ρx.
- Let $[\rho]D$ be D with any written variable x replaced by ρx.

In each case we apply appropriate α-conversion to avoid capture of free variables. □

EXAMPLES. Some example renamings are:

$$\begin{aligned} (x := !\nabla x)[y/x] &= (x := !\nabla y) \\ [y/x](x := !\nabla x) &= (y := !\nabla x) \\ [y/x](x := !\nabla x)[y/x] &= (y := !\nabla y) \end{aligned}$$

We can α-convert a local variable (when y is fresh):

$$\nu x\,.\,D \quad \alpha\text{-converts to} \quad \nu y\,.\,([y/x]D[y/x])$$

For example:

$$\nu x\,.\,(x := !\nabla x) \quad \alpha\text{-converts to} \quad \nu y\,.\,(y := !\nabla y)$$

If $\mathsf{wv}\,D$ and $\mathsf{wv}\,E$ are disjoint then we can define a localized declaration as:

$$\mathsf{local}\,D\,\mathsf{in}\,E = \nu(\mathsf{wv}\,D)\,.\,(D,E)$$

This can be generalized to any declarations D and E by α-converting the written variables of D first. If $\mathsf{wv}\,D = \{x_1,\ldots,x_n\}$ and $y_1,\ldots,y_n$ are fresh then:

$$\mathsf{local}\,D\,\mathsf{in}\,E = \nu\vec{y}\,.\,([\vec{y}/\vec{x}]D[\vec{y}/\vec{x}], E[\vec{y}/\vec{x}])$$

for example:

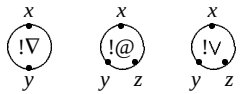
$$\mathsf{local}(x := ?\nabla x)\,\mathsf{in}\,(x := !\lambda w\,.\,x) = \nu y\,.\,(y := ?\nabla y, x := !\lambda w\,.\,y)$$

We shall see in Section 3.3 that $x := !(\mathsf{rec}\,D\,\mathsf{in}\,M)$ is semantically equivalent to $\mathsf{local}\,D\,\mathsf{in}\,(x := !M)$. □

DEFINITION. We can draw a declaration as a graph, in the fashion of MILNER's (1989) flow graphs for CCS. A declaration $x := !M$ with read variables $y_1,\ldots,y_n$ can be drawn:

Similarly, a declaration $x := ?M$ can be drawn:

When M is ∇y, $y@z$ or $y \vee z$ we will usually elide the read variables, drawing $x := !\nabla y$, $x := !y@z$ and $x := !y \vee z$ as:



A declaration ε can be drawn as the empty graph.

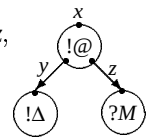
A declaration D,E can be drawn by superimposing D on E.

A declaration $\nu x\,.\,D$ can be drawn by drawing D and erasing any occurrence of x.

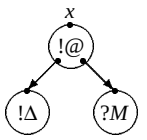
Whenever we have the same variable being read and written in a graph, we will draw an arrow from the read variable to the written variable. □

EXAMPLES. The application of Δ to M can be drawn:

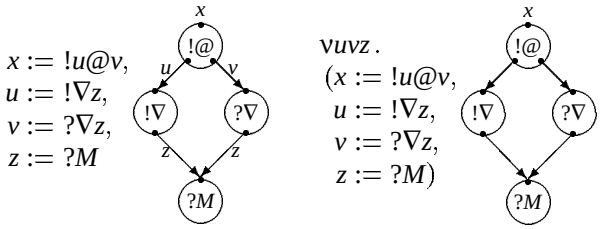
$x := !y@z,$
$y := !\Delta,$
$z := ?M$



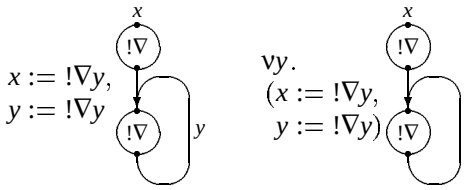
$\nu yz\,.$
$(x := !y@z,$
$y := !\Delta,$
$z := ?M)$



The application of M to itself, with sharing can be drawn:



A cyclic graph can be drawn:



We shall see that such tight cyclic graphs give rise to deadlock. □

3.2 Operational semantics

We will give our operational semantics in two parts, based on BERRY and BOUDOL's (1990) *Chemical Abstract Machine*. We shall first define a syntactic equivalence $\equiv$ on declarations, and then define an operational semantics up to $\equiv$. This allows us to abstract away from syntactic details such as associativity of concatenation, and present the 'bare bones' of the operational semantics.

A similar approach has been taken by MILNER (1991) in presenting the π-calculus, and we shall follow his example more closely than that of BERRY and BOUDOL.

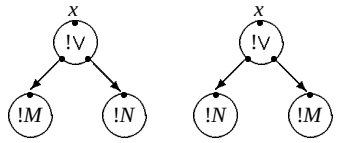
The syntactic equivalence $D \equiv E$ is given by:

- *Concatenation rules* which say that concatenation is an abelian monoid with unit ε.
- *Scope rules* which give properties about local variables:
  - Local variables can be α-converted.
  - The order of declaration of local variables is unimportant.
  - The scope of a local variable can *migrate* when this does not cause the capture of free variables.

  These rules for local variables are the same as MILNER's (1991) scope rules for the π-calculus, except that we omit $\nu x \,.\, \nu x \,.\, P \equiv \nu x \,.\, P$, since the declaration $\nu x \,.\, \nu x \,.\, D$ is not well-formed.
- A *fork rule* saying that fork is commutative.

- *Congruence rules* which say that $\equiv$ is an equivalence relation, and is respected by concatenation and local variables.

Many of these equivalences were implicitly used when we drew declarations as graphs. For example, $D,(E,F) \equiv (D,E),F$ corresponds to the fact that superimposition of graphs is associative. The only axiom which equates declarations with different graphs is ($\vee$COMM), which says that fork is commutative, and so we shall not distinguish between:



This axiom halves the number of rules required for fork.

DEFINITION. If $z \notin \mathsf{fv}\, D$ then $\equiv$ is given by axioms:

$$
\begin{array}{cc}
(\text{ASSOC}) & D,(E,F) \equiv (D,E),F \\
(\text{COMM}) & D,E \equiv E,D \\
(\text{UNIT}) & D,\varepsilon \equiv D \\
(\alpha) & \nu x \,.\, D \equiv \nu z \,.\, ([z/x]D[z/x]) \\
(\nu\text{SWAP}) & \nu x \,.\, \nu y \,.\, D \equiv \nu y \,.\, \nu x \,.\, D \\
(\nu\text{MIG}) & D,\nu z \,.\, E \equiv \nu z \,.\, (D,E) \\
(\vee\text{COMM}) & x := !(y \vee z) \equiv x := !(z \vee y) \\
(\text{REFL}) & D \equiv D
\end{array}
$$

and structural rules:

$$
(\text{SYMM})\ \frac{D \equiv E}{E \equiv D} \qquad (\text{TRANS})\ \frac{D \equiv E \equiv F}{D \equiv F}
$$

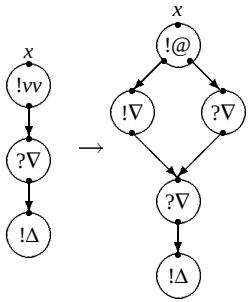
$$
(\text{L})\ \frac{D \equiv E}{D,F \equiv E,F} \qquad (\text{R})\ \frac{D \equiv E}{F,D \equiv F,E} \qquad (\nu)\ \frac{D \equiv E}{\nu x \,.\, D \equiv \nu x \,.\, E}
$$

Note that if $D \equiv E$ then $\mathsf{rv}\, D = \mathsf{rv}\, E$ and $\mathsf{wv}\, D = \mathsf{wv}\, E$. □

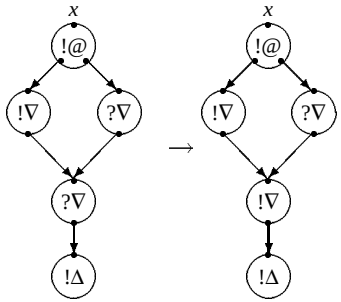
We can use the equivalence $\equiv$ to simplify the operational semantics for graph reduction. This can be given as eight axioms and three structural rules. The axioms can be broken down into four phases:

- *Graph building*, in which a recursive declaration is expanded into a graph, for

example:

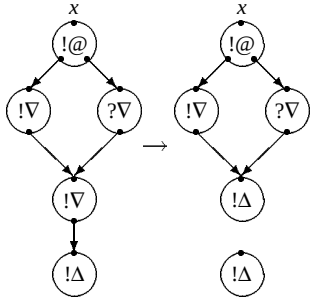


- *Spine traversal*, in which an untagged node pointed to by a tagged node becomes tagged, for example:



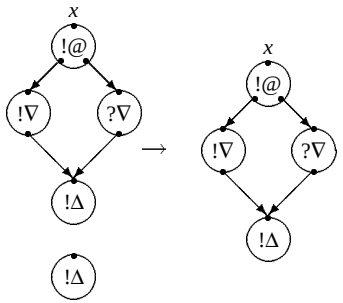
  There are three axioms, depending on whether the tagged node is an indirection, an application, or a fork.

- *Updating*, in which a node pointing to an abstraction is updated, for example:

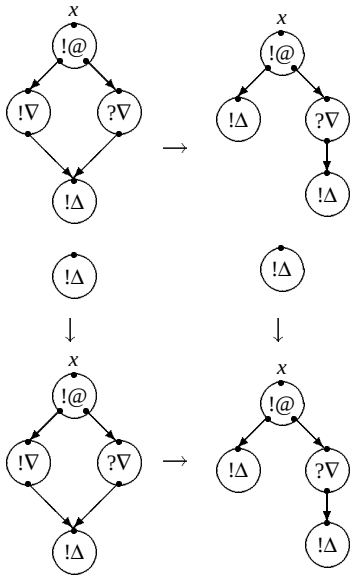


  There are three axioms, depending on whether the node is an indirection, an application, or a fork.

- *Garbage collection*, in which a sub-graph with no incoming pointers is removed, for example:



These phases are not sequential, and there may be more than one axiom which can be applied at any one point. Since each axiom uses a small number of nodes, there is much scope for concurrency, for example:



The operational semantics $\longrightarrow$ is first given by an operational semantics $\mapsto$, and $\longrightarrow$ is defined as $\equiv\mapsto\equiv$.

DEFINITION. $\mapsto$ is given by axioms:

$$
\begin{array}{cc}
(\text{BUILD}) & x := !(\mathsf{rec}\, D \,\mathsf{in}\, M) \mapsto \mathsf{local}\, D \,\mathsf{in}\, (x := !M) \\
(\nabla\text{TRAV}) & x := !\nabla y, y := ?M \mapsto x := !\nabla y, y := !M \\
(@\text{TRAV}) & x := !y@z, y := ?M \mapsto x := !y@z, y := !M \\
(\vee\text{TRAV}) & x := !y \vee z, y := ?M \mapsto x := !y \vee z, y := !M \\
(\nabla\text{UPD}) & x := !\nabla y, y := !\lambda w.\, M \mapsto x := !\lambda w.\, M, y := !\lambda w.\, M \\
(@\text{UPD}) & x := !y@z, y := !\lambda w.\, M \mapsto x := !M[z/w], y := !\lambda w.\, M \\
(\vee\text{UPD}) & x := !y \vee z, y := !\lambda w.\, M \mapsto x := !\,\mathsf{I}, y := !\lambda w.\, M \\
(\gamma) & \nu(\mathsf{wv}\, D).\, D \mapsto \varepsilon
\end{array}
$$

and structural rules:

$$
(\text{L})\ \frac{D \mapsto E}{D, F \mapsto E, F} \qquad (\text{R})\ \frac{D \mapsto E}{F, D \mapsto F, E} \qquad (\nu)\ \frac{D \mapsto E}{\nu x.\, D \mapsto \nu x.\, E}
$$

Note that if $D \mapsto E$ then $\mathsf{rv}\, D \supseteq \mathsf{rv}\, E$ and $\mathsf{wv}\, D = \mathsf{wv}\, E$.

- $D \to E$ iff $D \equiv \mapsto \equiv E$.
- $D \to^0 E$ iff $D \equiv E$, and $D \to^{n+1} E$ iff $D \to\to^n E$.
- $D \to^* E$ iff $\exists n.\, D \to^n E$.
- $D \to^{\leq i} E$ iff $\exists n \leq i.\, D \to^n E$. □

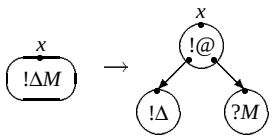
EXAMPLES. In the graph building phase, we take a term containing a recursive declaration and build a graph from it:

$$x := !(\mathsf{rec}\, D \,\mathsf{in}\, M) \mapsto \mathsf{local}\, D \,\mathsf{in}\, (x := !M)$$

For example, the deduction:

$$
\begin{array}{lr}
x := !\Delta M & \\
\quad \equiv x := !(\mathsf{rec}(y := !\Delta, z := ?M) \,\mathsf{in}\, (y@z)) & (\text{Defn of } \Delta M) \\
\quad \mapsto \mathsf{local}(y := !\Delta, z := ?M) \,\mathsf{in}\, (x := !y@z) & (\text{BUILD}) \\
\quad \equiv \nu yz.\, (x := !y@z, y := !\Delta, z := ?M) & (\text{Defn of } \mathsf{local})
\end{array}
$$

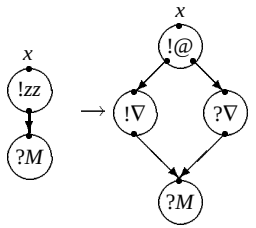
can be drawn graphically as:



Similarly, the deduction:

$$
\begin{array}{lr}
\nu z.\, (x := !zz, z := ?M) & \\
\quad \equiv \nu z.\, (x := !(\mathsf{rec}(u := !\nabla z, v := ?\nabla z) \,\mathsf{in}\, (u@v)), z := ?M) & (\text{Defn of } zz) \\
\quad \mapsto \nu z.\, (\mathsf{local}(u := !\nabla z, v := ?\nabla z) \,\mathsf{in}\, (x := !u@v), z := ?M) & (\text{BUILD}) \\
\quad \equiv \nu z.\, (\nu uv.\, (u := !\nabla z, v := ?\nabla z, x := !u@v), z := ?M) & (\text{Defn of } \mathsf{local}) \\
\quad \equiv \nu uvz.\, (u := !\nabla z, v := ?\nabla z, x := !u@v, z := ?M) & (\nu\text{MIG})
\end{array}
$$

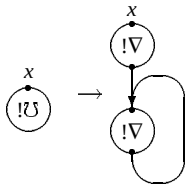
can be drawn graphically as:



We can build cyclic graphs, for example the deduction:

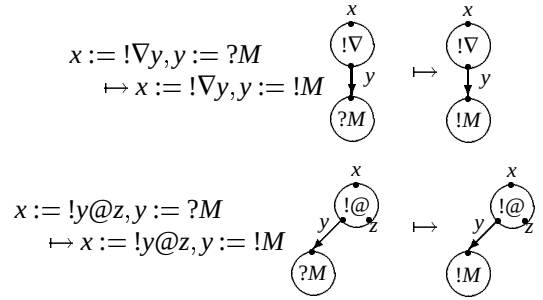
$$
\begin{array}{lr}
x := !\mho & \\
\quad \equiv x := !(\mathsf{rec}(y := !\nabla y) \,\mathsf{in}\, (\nabla y)) & (\text{Defn of } \mho) \\
\quad \mapsto \mathsf{local}(y := !\nabla y) \,\mathsf{in}\, (x := !\nabla y) & (\text{BUILD}) \\
\quad \equiv \nu y.\, (x := !\nabla y, y := !\nabla y) & (\text{Defn of } \mathsf{local})
\end{array}
$$

can be drawn graphically as:

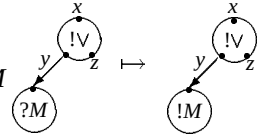


Note that this axiom can only be applied to tagged declarations $x := !(\mathsf{rec}\, D \,\mathsf{in}\, M)$ and not untagged declarations $x := ?(\mathsf{rec}\, D \,\mathsf{in}\, M)$. In implementation terms, this is because we only build a graph for terms currently under evaluation.

In the spine traversal phase, we find a tagged node which points to an untagged node, and tag it. Thus we have three axioms, depending on the form of the tagged node:

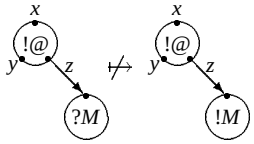


$$x := !y \vee z, y := ?M \mapsto x := !y \vee z, y := !M$$

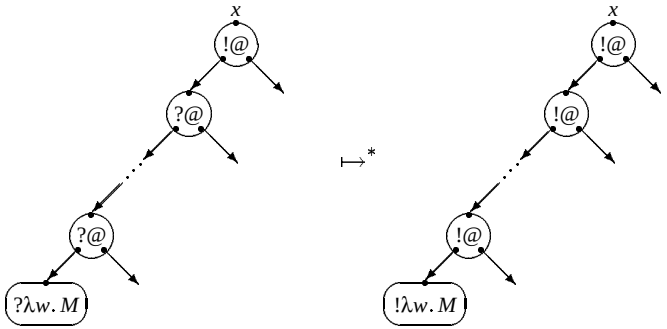


Note that since we are modeling lazy evaluation, we have:

$$x := !y@z, z := ?M \not\mapsto x := !y@z, z := !M$$

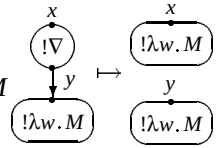


This phase is called 'spine traversal' because there will often be a 'spine' of untagged indirection, application, and fork nodes, which will all be tagged. For example:



This phase terminates when we reach a tagged function $\lambda w.M$, as in the above example, and we can perform updating. We have three axioms, depending on which kind of node is pointing to the function. If it is an indirection node, we make a copy of the function. Since the function is already in weak head normal form we are not losing any sharing:

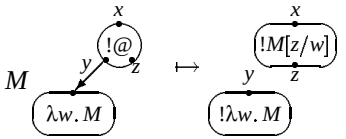
$$x := !\nabla y, y := !\lambda w.M \mapsto x := !\lambda w.M, y := !\lambda w.M$$



If we have an application node pointing to a function we can perform β-reduction. This is the 'work' of the operational semantics, and we can regard the other rules as manipulation to produce a graph where β-reduction can take place. Note that since we are using renaming rather than substitution to model β-reduction, we

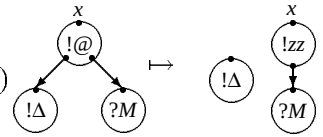
have modeled sharing:

$$x := !y@z, y := !\lambda w.M \mapsto x := !M[z/w], y := !\lambda w.M$$



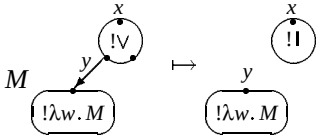
For example:

$$\nu yz.(x := !y@z, y := !\Delta, z := ?M) \mapsto \nu yz.(x := !zz, y := !\Delta, z := ?M)$$



If we have a fork node pointing to a function we can return the identity function, in the same way as the Λ_{P} rule for P.

$$x := !y \vee z, y := !\lambda w.M \mapsto x := !\mathsf{I}, y := !\lambda w.M$$



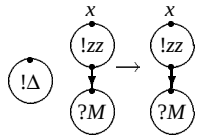
The final phase is *garbage collection* where any subgraphs with no incoming pointers are removed. This corresponds to the last axiom:

$$\nu(\mathsf{w} \nu D).D \mapsto \varepsilon$$

For example, the deduction:

$$\begin{aligned}
&\nu yz.(x := !zz, y := !\Delta, z := ?M) \\
&\quad \equiv \nu yz.(y := !\Delta, x := !zz, z := ?M) && (\text{COMM}) \\
&\quad \equiv \nu zy.(y := !\Delta, x := !zz, z := ?M) && (\nu\text{SWAP}) \\
&\quad \equiv \nu z.(\nu y.(y := !\Delta), x := !zz, z := ?M) && (\nu\text{MIG}) \\
&\quad \mapsto \nu z.(\varepsilon, x := !zz, z := ?M) && (\gamma) \\
&\quad \equiv \nu z.(x := !zz, z := ?M) && (\text{UNIT})
\end{aligned}$$

can be drawn graphically as:



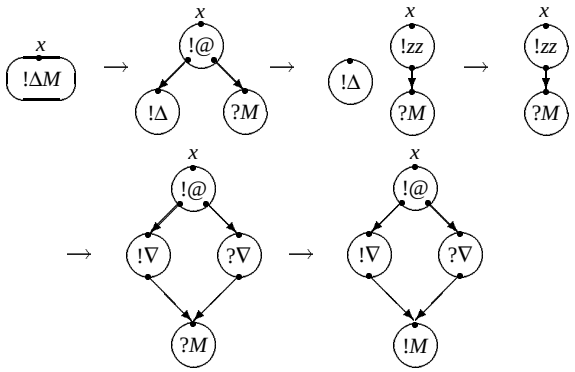
Note that although we have presented these phases sequentially, they can be carried out in any order. Since most of the axioms involve very small graphs, containing three or fewer nodes, we have a very small *granularity* and thus much scope for concurrency.

However, the axiom for garbage collection involves graphs of arbitrary size, and so has much larger granularity, and so less scope for concurrency. In implementation terms, this corresponds to the fact that much less concurrent graph reduction can take place during garbage collection. Indeed, many graph reduction engines suspend graph reduction completely during garbage collection.

We can combine these phases together to reduce any graph, for example the deduction:

$$
\begin{array}{lr}
x := !\Delta M & \\
\quad \equiv x := !(\mathsf{rec}\, y := !\Delta, z := ?M \,\mathsf{in}\, y@z) & \text{(Defn of } \Delta M\text{)} \\
\quad \mapsto \mathsf{local}\, y := !\Delta, z := ?M \,\mathsf{in}\, x :=!y@z & \text{(BUILD)} \\
\quad \equiv \nu yz\,.\,(x := !y@z, y := !\Delta, z := ?M) & \text{(Defn of } \mathsf{local}\text{)} \\
\quad \mapsto \nu yz\,.\,(x := !zz, y := !\Delta, z := ?M) & \text{(@UPD)} \\
\quad \equiv \nu z\,.\,(x := !zz, \nu y\,.\,(y := !\Delta), z := ?M) & (\nu\text{MIG}) \\
\quad \mapsto \nu z\,.\,(x := !zz, \varepsilon, z := ?M) & (\beta) \\
\quad \equiv \nu z\,.\,(x := !zz, z := ?M) & \text{(UNIT)} \\
\quad \equiv \nu z\,.\,(x := !(\mathsf{rec}\, u := !\nabla z, v := ?\nabla z \,\mathsf{in}\, u@v), z := ?M) & \text{(Defn of } zz\text{)} \\
\quad \mapsto \nu z\,.\,(\mathsf{local}\, u := !\nabla z, v := ?\nabla z \,\mathsf{in}\, x := !u@v, z := ?M) & \text{(BUILD)} \\
\quad \equiv \nu z\,.\,(\nu uv\,.\,(x := !u@v, u := !\nabla z, v := ?\nabla z), z := ?M) & \text{(Defn of } \mathsf{local}\text{)} \\
\quad \equiv \nu uvz\,.\,(x := !u@v, u := !\nabla z, v := ?\nabla z, z := ?M) & (\nu\text{MIG}) \\
\quad \mapsto \nu uvz\,.\,(x := !u@v, u := !\nabla z, v := ?\nabla z, z := !M) & (\nabla\text{TRAV})
\end{array}
$$

can be drawn graphically as:

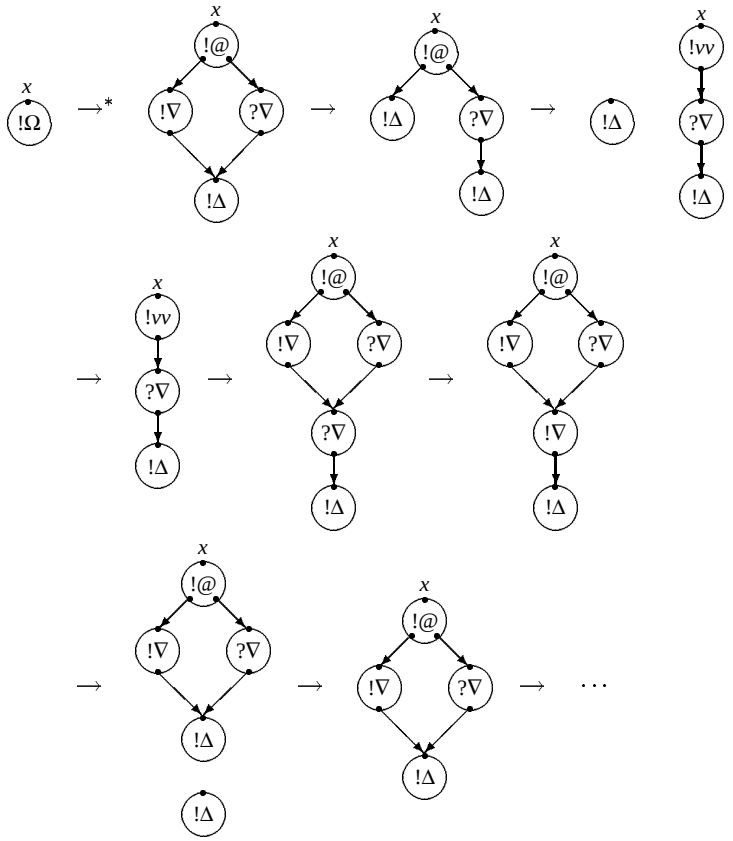


These steps are: graph building, β-reduction, garbage collection, graph building and spine traversal. We thus have:

$$
\begin{array}{lr}
x := !\Omega & \\
\quad \equiv\ x := !\Delta\Delta & \text{(Defn of } \Omega\text{)} \\
\quad \to^* \nu uvz\,.\,(x := !u@v, u := !\nabla z, v := ?\nabla z, z := !\Delta) & \text{(Above)} \\
\quad \to\ \nu uvz\,.\,(x := !u@v, u := !\Delta, v := ?\nabla z, z := !\Delta) & (\nabla\text{UPD})
\end{array}
$$

$$
\begin{array}{lr}
\quad \to\ \nu uvz\,.\,(x := !vv, u := !\Delta, v := ?\nabla z, z := !\Delta) & (@\text{UPD}) \\
\quad \to\ \nu vz\,.\,(x := !vv, v := ?\nabla z, z := !\Delta) & (\gamma) \\
\quad \to\ \nu vz\,.\,(x := !t@u, t := !\nabla v, u := ?\nabla v, v := ?\nabla z, z := !\Delta) & \text{(BUILD)} \\
\quad \to\ \nu vz\,.\,(x := !t@u, t := !\nabla v, u := ?\nabla v, v := !\nabla z, z := !\Delta) & (\nabla\text{TRAV}) \\
\quad \to\ \nu vz\,.\,(x := !t@u, t := !\nabla v, u := ?\nabla v, v := !\Delta, z := !\Delta) & (\nabla\text{UPD}) \\
\quad \to\ \nu vz\,.\,(x := !t@u, t := !\nabla v, u := ?\nabla v, v := !\Delta) & (\gamma) \\
\quad \to\ \cdots &
\end{array}
$$

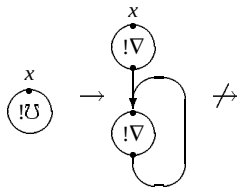
This can be drawn graphically:



Thus $x := !\Omega$ is divergent. This can be contrasted with the deduction:

$$(x := !\mho) \to \nu y\,.\,(x := !y, y := !y) \not\to$$

which can be drawn graphically as:



since the graph is fully built, the spine is tagged, there are no function nodes to reduce, and there is no garbage. Thus the declaration $x := !\mho$ is *deadlocked* rather than *divergent*. Denotationally, we shall identify the terms $\mho$ and Ω, since neither of them can reach weak head normal form, although operationally they are very different. □

We can define x to be in *weak head normal form* (whnf) in D iff D contains $x := !\lambda w.\, M$.

DEFINITION.

- x is in whnf in $(x := !\lambda w.\, M)$.
- x is in whnf in (D, E) if x is in whnf in D or E.
- x is in whnf in $\nu y.\, D$ if x is in whnf in D and $x \neq y$.

Note that if $D \equiv E$ and x is in whnf in D then x is in whnf in E. □

We can use this to define our notion of testing:

- A program is a closed declaration.
- A test is a closing context $C[\cdot]$ and a variable x.
- A term M passes a test iff, when we tag x in $C[M]$, the result reduces to weak head normal form at x.

DEFINITION.

- tag_x is defined (when $x \neq y$) as:

$$\begin{array}{ll} \mathsf{tag}_x(x := !M) = (x := !M) & \mathsf{tag}_x(x := ?M) = (x := !M) \\ \mathsf{tag}_x(y := !M) = (y := !M) & \mathsf{tag}_x(y := ?M) = (y := ?M) \\ \mathsf{tag}_x(\nu x.\, D) = \nu x.\, D & \mathsf{tag}_x(\nu y.\, D) = \nu y.\, (\mathsf{tag}_x D) \\ \mathsf{tag}_x \varepsilon = \varepsilon & \mathsf{tag}_x(D, E) = (\mathsf{tag}_x D), (\mathsf{tag}_x E) \end{array}$$

- For closed D, $D \Downarrow_x E$ iff $\mathsf{tag}_x D \rightarrow^* E$ and x is in whnf in E.
- $D\Downarrow_x$ iff $\exists E.\, D \Downarrow_x E$ and $D\Uparrow_x$ iff $\neg\exists E.\, D \Downarrow_x E$.
- $M \sqsubseteq_O N$ iff $C[M]\Downarrow_x \Rightarrow C[N]\Downarrow_x$ for any x and closing context C.
- $D \sqsubseteq_O E$ iff $\mathsf{wv} D = \mathsf{wv} E$ and $C[D]\Downarrow_x \Rightarrow C[E]\Downarrow_x$ for any x and closing context C. □

Note that *convergence* ($D\Downarrow_x$) and *termination* ($D \rightarrow^* \not\rightarrow$) are very different in this operational semantics, although they are equivalent in Λ_{P}. For example:

- $x := !(\mathsf{rec}\, y := !\Omega \,\mathsf{in}\, \lambda w.\, y)$ converges, but does not terminate.
- $x := !\mho$ terminates, but does not converge.

Since we are using convergence rather than termination as our definition of testing equivalence, we can identify Ω and $\mho$. The testing equivalence based on termination has been investigated by the author (1993).

3.3 Denotational semantics

The denotational semantics for Lam is given in the same domain $\mathbf{D} \simeq (\mathbf{D} \rightarrow \mathbf{D})_\perp$ as Λ_{P}. The semantics of Dec is given as $[\![D]\!] : \Sigma \rightarrow \Sigma$, so if σ is an environment, then so is $[\![D]\!]\sigma$. The main difference between the semantics of Lam and that of Λ_{P} is that the former makes explicit use of recursion. For example, if we define:

$$\mathsf{const}\, ab = a$$

Then we can show that the semantics of the 'ogre':

$$\mathsf{rec}\, x := !\lambda y.\, \nabla x \,\mathsf{in}\, \nabla x$$

is given as the least solution to:

$$f = \mathsf{fold} \circ \mathsf{lift} \circ \mathsf{const} \circ f$$

and so $[\![\mathsf{rec}\, x := !\lambda y.\, \nabla x \,\mathsf{in}\, \nabla x]\!] = \top$.

DEFINITION. Define $[\![M]\!] : \Sigma \rightarrow \mathbf{D}$ as:

$$\begin{aligned} [\![\nabla x]\!] &= \mathsf{read}\, x \\ [\![x@y]\!] &= \mathsf{split}(\mathsf{apply} \circ \mathsf{read}\, x)(\mathsf{read}\, y) \\ [\![x \vee y]\!] &= \mathsf{split}(\mathsf{fork} \circ \mathsf{read}\, x)(\mathsf{read}\, y) \\ [\![\lambda x.\, M]\!] &= \mathsf{fold} \circ \mathsf{lift} \circ \mathsf{fn}\, x [\![M]\!] \\ [\![\mathsf{rec} D \,\mathsf{in}\, M]\!] &= [\![M]\!] \circ [\![D]\!] \end{aligned}$$

Define $[\![D]\!] : \Sigma \rightarrow \Sigma$ as:

$$\begin{aligned} [\![x := !M]\!] &= \mathsf{fix}(\mathsf{set}\{x\}(x := [\![M]\!])) \\ [\![x := ?M]\!] &= \mathsf{fix}(\mathsf{set}\{x\}(x := [\![M]\!])) \\ [\![\varepsilon]\!] &= \mathsf{id} \\ [\![D, E]\!] &= \mathsf{fix}(\mathsf{set}(\mathsf{wv}(D, E))([\![D]\!] \circ [\![E]\!])) \\ [\![\nu x.\, D]\!] &= \mathsf{new}\, x [\![D]\!] \end{aligned}$$

where:

$$\mathsf{new}\, x f \sigma y = \begin{cases} \sigma x & \text{if } x = y \\ f \sigma y & \text{otherwise} \end{cases}$$

$$(x := f)\sigma y = \begin{cases} f\sigma & \text{if } x = y \\ \sigma y & \text{otherwise} \end{cases}$$

$$\mathsf{set}\,X\,f g\sigma x = \begin{cases} f(g\sigma)x & \text{if } x \in X \\ \sigma x & \text{otherwise} \end{cases}$$

$$\mathsf{fix}\,f = \bigvee\{f^n\bot \mid n \text{ in } \omega\}$$

- $M \sqsubseteq_D N$ iff $[\![M]\!] \leq [\![N]\!]$.
- $D \sqsubseteq_D E$ iff $\mathsf{wv}D = \mathsf{wv}E$ and $[\![D]\!] \leq [\![E]\!]$. □

EXAMPLES. We can show that the semantics of the 'ogre' term is $\top$, since:

$$\begin{aligned}
&[\![\mathsf{rec}x := !\lambda y.\,\nabla x \text{ in } \nabla x]\!] \\
&\quad = [\![\nabla x]\!] \circ [\![x := !\lambda y.\,\nabla x]\!] && \text{(Defn of } [\![\mathsf{rec}D \text{ in } M]\!]) \\
&\quad = \mathsf{read}\,x \circ [\![x := !\lambda y.\,\nabla x]\!] && \text{(Defn of } [\![\nabla x]\!]) \\
&\quad = \mathsf{read}\,x \circ \mathsf{fix}(\mathsf{set}\{x\}(x := [\![\lambda y.\,\nabla x]\!])) && \text{(Defn of } [\![x := !M]\!]) \\
&\quad = \mathsf{read}\,x \circ \mathsf{set}\{x\}[\![\lambda y.\,\nabla x]\!](\mathsf{fix}(\mathsf{set}\{x\}(x := [\![\lambda y.\,\nabla x]\!])) && \text{(Unfold)} \\
&\quad = \mathsf{read}\,x \circ \mathsf{set}\{x\}[\![\lambda y.\,\nabla x]\!][\![x := !\lambda y.\,\nabla x]\!] && \text{(Defn of } [\![x := !M]\!]) \\
&\quad = [\![\lambda y.\,\nabla x]\!] \circ [\![x := !\lambda y.\,\nabla x]\!] && \text{(Defn of read and set)} \\
&\quad = \mathsf{fold} \circ \mathsf{lift} \circ \mathsf{fn}\,y[\![\nabla x]\!] \circ [\![x := !\lambda y.\,\nabla x]\!] && \text{(Defn of } [\![\lambda x.\,M]\!]) \\
&\quad = \mathsf{fold} \circ \mathsf{lift} \circ \mathsf{const} \circ ((\mathsf{read}\,x) \circ [\![x := !\lambda y.\,\nabla x]\!]) && \text{(Defn of fn and const)} \\
&\quad = \mathsf{fold} \circ \mathsf{lift} \circ \mathsf{const} \circ ([\![\nabla x]\!] \circ [\![x := !\lambda y.\,\nabla x]\!]) && \text{(Defn of } [\![\nabla x]\!]) \\
&\quad = \mathsf{fold} \circ \mathsf{lift} \circ \mathsf{const} \circ ([\![\mathsf{rec}x := !\lambda y.\,\nabla x \text{ in } \nabla x]\!]) && \text{(Defn of } [\![\mathsf{rec}D \text{ in } M]\!])
\end{aligned}$$

The only function which satisfies this is:

$$[\![\mathsf{rec}x := !\lambda y.\,\nabla x \text{ in } \nabla x]\!] = \top$$

The semantics agrees with that of Λ_{P}:

$$\begin{aligned}
[\![x]\!] &= \mathsf{read}\,x \\
[\![MN]\!] &= \mathsf{split}(\mathsf{apply} \circ [\![M]\!])[\![N]\!] \\
[\![\lambda x.\,M]\!] &= \mathsf{fold} \circ \mathsf{lift} \circ \mathsf{fn}\,x[\![M]\!] \\
[\![\mathsf{P}MN]\!] &= \mathsf{split}(\mathsf{fork} \circ [\![M]\!])[\![N]\!]
\end{aligned}$$

This means we can define M_ϕ from Section 2.7 in Lam and that $[\![M_\phi]\!] = [\![\phi]\!]$. We can also define D_Γ as:

$$\begin{aligned}
D_\varepsilon &= \varepsilon \\
D_{\Gamma,\Delta} &= D_\Gamma, D_\Delta \\
D_{x:\phi} &= x := !M_\phi
\end{aligned}$$

then we can show by induction on Γ that $[\![D_\Gamma]\!]\sigma = [\![\Gamma]\!]$. □

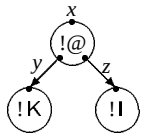
The properties of this denotational semantics are discussed in Section 3.9.

3.4 Program logic

The proof that **D** is fully abstract for Lam proceeds in much the same way as the proof in Chapter 2. We present a program logic, and use it as a link between

the denotational and operational semantics. The propositions we will use are the same as those from Chapter 2, and so we can use all of the material from Section 2.6. However, since we are looking at a different syntax, we need a different operational characterization and a different proof system.

Since the operational semantics for graph reduction is given between declarations rather than terms, the operational characterization of Φ is also given for declarations. So rather than defining $\models M : \phi$ for closed terms, we define $\models D : \Delta$ for closed declarations. The proposition $\models D : \Delta$ means that the term in D referred to by x satisfies $\Delta(x)$. For example, the graph:



satisfies $\phi \to \psi \to \psi$ at x, $\phi \to \psi \to \phi$ at y, and $\phi \to \phi$ at z, that is:

$$\models (x := !y@z, y := !\mathsf{K}, z := !\mathsf{I}) : (x : \phi \to \psi \to \psi, y : \phi \to \psi \to \phi, z : \phi \to \phi)$$

We define 'D satisfies Δ' as:

- Any declaration satisfies ε or $x : \omega$.
- If D satisfies Γ and Δ, then D satisfies $\Gamma \wedge \Delta$.
- If $D{\Downarrow_x}$ and any extension E of D and $z := !x@y$ which satisfies $y : \phi$ satisfies $z : \psi$, then D satisfies $\phi \to \psi$.

For example:

- D satisfies $x : \gamma$ iff $D{\Downarrow_x}$.
- $x := !\mathsf{I}$ satisfies $x : \phi \to \phi$ because any graph
  $$x := !\mathsf{I}, z := !x@y, D$$
  which satisfies $y : \phi$ also satisfies $z : \phi$.
- $x := !\mathsf{K}$ satisfies $x : \phi \to \psi \to \phi$ because any graph
  $$x := !\mathsf{K}, z := !x@y, w := !z@v, D$$
  which satisfies $y : \phi$ and $v : \psi$ also satisfies $w : \phi$.
- We can show by induction on ϕ that $(w := !\lambda y.\,w, x := !\lambda y.\,w)$ satisfies $x : \phi$. The only difficult case is when $\phi = \psi \to \chi$, in which case:
  $$(w := !\lambda y.\,w, x := !\lambda y.\,w){\Downarrow_x}$$
  and in any graph:
  $$w := !\lambda y.\,w, x := !\lambda y.\,w, z := !x@y, D$$

if y satisfies ψ then:

$$\begin{array}{l} w := !\lambda y\,.\,w, x := !\lambda y\,.\,w, z := !x@y, D \\ \quad \to w := !\lambda y\,.\,w, x := !\lambda y\,.\,w, z := !\nabla w, D \\ \quad \to w := !\lambda y\,.\,w, x := !\lambda y\,.\,w, z := !\lambda y\,.\,w, D \end{array}$$

which by induction satisfies $z : \chi$. Thus:

$$(w := !\lambda y\,.\,w, x := !\lambda y\,.\,w) : \phi$$

From this it is simple to show that $(w := !\lambda y\,.\,w) : (w : \phi)$.

This definition depends on the notion of 'graph extension', which is the preorder $D \sqsubseteq E$.

DEFINITION. $D \sqsubseteq E$ iff we can find $\vec{x}$, $\vec{y}$, D' and E' such that:

$$D \equiv \nu\vec{x}\,.\,D' \qquad E \equiv \nu\vec{x}\vec{y}\,.\,(D', E') \qquad \mathsf{fv}\,D \cap \vec{y} = \emptyset$$

Note that $\sqsubseteq$ is a preorder, and that $D \sqsubseteq E \sqsubseteq D$ iff $D \equiv E$. □

We can then define the the operational interpretation of the logic.

DEFINITION. For closed declarations, $\models D : \Delta$ is given by the axioms:

$$(\varepsilon\mathrm{I}) \quad \models D : \varepsilon \qquad (\omega\mathrm{I}) \quad \models D : (x : \omega)$$

and structural rules:

$$(\wedge\mathrm{I})\ \frac{\models D : \Gamma \quad \models D : \Delta}{\models D : \Gamma \wedge \Delta} \qquad (\to\mathrm{I})\ \frac{\begin{array}{c}\forall (z := !x@y) \sqsubseteq E \sqsupseteq D. \\ D\Downarrow_x \quad \models E : (y : \phi) \Rightarrow \models E : (z : \psi)\end{array}}{\models D : (x : \phi \to \psi)}$$

This can be generalized to any D by defining $\Gamma \models D : \Delta$ iff:

$$\forall E\,.\,(\models D, E : \nu(\mathsf{wv}D)\,.\,\Gamma) \text{ implies } (\models D, E : \Delta)$$

Similarly, $\Gamma \models M : \phi$ iff:

$$\forall D, z\,.\,(\models (D, z := !M) : \Gamma) \text{ implies } (\models (D, z := !M) : (z : \phi))$$

One consequence of full abstraction is that for λ-calculus terms, this operational definition agrees with the definition of Section 2.4. □

We can define a proof system for Lam as we did for Λ_{P}. This uses the same propositions, and will have judgements of the form $\Gamma \vdash M : \phi$ and $\Gamma \vdash D : \Delta$. The main difference between the proof system for Lam and that of Λ_{P} is the proof system for recursive declarations. Note that:

- The proof rules (!) and (?) for tagged and untagged declarations are the same. Semantically there is no difference between a tagged or an untagged node, although they have very different operational behaviour.

- We are considering declarations to be recursive, and so the proof rules (!), (?), (L) and (R) for declarations are recursive. For example, to show $\Gamma \vdash D, E : \Theta$, we are allowed to have a subgoal of $\Gamma \vdash D, E : \Delta$.

DEFINITION. The proof system $\Gamma \vdash M : \phi$ is given by axioms:

$$\begin{array}{cc} (\omega\mathrm{I}) & \vdash M : \omega \\ (\mathrm{ID}) & x : \phi \vdash \nabla x : \phi \\ (\to\mathrm{E}) & (x : \phi \to \psi) \wedge (y : \phi) \vdash x@y : \psi \\ (\vee a) & x : \gamma \vdash x \vee y : \phi \to \phi \\ (\vee b) & y : \gamma \vdash x \vee y : \phi \to \phi \end{array}$$

and structural rules:

$$(\wedge\mathrm{I})\ \frac{\Gamma \vdash M : \phi \quad \Gamma \vdash M : \psi}{\Gamma \vdash M : (\phi \wedge \psi)} \quad (\leq)\ \frac{\vdash \Gamma \leq \Delta \quad \Delta \vdash M : \phi \quad \vdash \phi \leq \psi}{\Gamma \vdash M : \psi}$$

$$(\to\mathrm{I})\ \frac{\Gamma, x : \phi \vdash M : \psi}{\Gamma \vdash \lambda x\,.\,M : \phi \to \psi} \qquad (\mathsf{rec})\ \frac{\Gamma \vdash D : \Delta \quad \Delta \vdash M : \phi}{\Gamma \vdash \mathsf{rec}\,D\,\mathsf{in}\,M : \phi}$$

The proof system $\Gamma \vdash D : \Delta$ is given by axiom:

$$(\bot) \quad \Gamma \vdash D : \nu(\mathsf{wv}D)\,.\,\Gamma$$

and structural rules:

$$(\wedge\mathrm{I})\ \frac{\Gamma \vdash D : \Delta \quad \Gamma \vdash D : \Theta}{\Gamma \vdash D : (\Delta \wedge \Theta)} \qquad (\leq)\ \frac{\vdash \Gamma \leq \Gamma' \quad \Gamma' \vdash D : \Delta' \quad \vdash \Delta' \leq \Delta}{\Gamma \vdash D : \Delta}$$

$$(!)\ \frac{\Gamma \vdash (x := !M) : \Delta \quad \Delta \vdash M : \phi}{\Gamma \vdash (x := !M) : (x : \phi)} \qquad (?)\ \frac{\Gamma \vdash (x := ?M) : \Delta \quad \Delta \vdash M : \phi}{\Gamma \vdash (x := ?M) : (x : \phi)}$$

$$(\mathrm{L})\ \frac{\Gamma \vdash D, E : \Delta \quad \Delta \vdash D : \Theta}{\Gamma \vdash D, E : \Theta} \qquad (\mathrm{R})\ \frac{\Gamma \vdash D, E : \Delta \quad \Delta \vdash E : \Theta}{\Gamma \vdash D, E : \Theta}$$

$$(\nu)\ \frac{\nu x\,.\,\Gamma \vdash D : \Delta}{\Gamma \vdash \nu x\,.\,D : \nu x\,.\,\Delta}$$

where $\nu x\,.\,(\Gamma, x : \phi, \Delta) = \Gamma, \Delta$ and $\nu x\,.\,\Gamma = \Gamma$ when $x \notin \mathsf{wv}\Gamma$. Then:

- $M \sqsubseteq_S N$ iff $\forall \Gamma, \phi\,.\,\Gamma \vdash M : \phi \Rightarrow \Gamma \vdash N : \psi$.
- $D \sqsubseteq_S E$ iff $\mathsf{wv}D = \mathsf{wv}E$ and $\forall \Gamma, \Delta\,.\,\Gamma \vdash D : \Delta \Rightarrow \Gamma \vdash E : \Delta$. □

EXAMPLES. The proof system for Lam is similar to that for Λ_{P}. Indeed, we can use this proof system to show:

$$\frac{}{x : \phi \vdash x : \phi} \qquad \frac{\Gamma \vdash M : \phi \to \psi \quad \Gamma \vdash N : \phi}{\Gamma \vdash MN : \psi}$$

$$\frac{\Gamma \vdash M : \gamma}{\Gamma \vdash \mathsf{P}MN : \phi \to \phi} \qquad \frac{\Gamma \vdash N : \gamma}{\Gamma \vdash \mathsf{P}MN : \phi \to \phi}$$

The proof system for Dec allows recursive proofs of properties of declarations. For example, we can prove by induction on ϕ that $\vdash (x := !\lambda w.\,\nabla x) : (x : \phi)$. The

only difficult case is when $\phi = \psi \to \chi$:

$$\dfrac{\dfrac{\dfrac{\vdots}{\vdash (x := !\lambda w.\nabla x) : (x : \chi)}\,(\text{Indn})}{\vdash (x := !\lambda w.\nabla x) : (x : \psi \to \chi)}\,(!) \qquad \dfrac{\dfrac{\dfrac{}{(x:\chi) \vdash \nabla x : \chi}\,(\text{ID})}{(x:\chi, w:\psi) \vdash \nabla x : \chi}\,(\leq)}{(x:\chi) \vdash (\lambda w.\nabla x) : (\psi \to \chi)}\,(\to 1)}$$

From this we can show that $\vdash \mathsf{rec}(x := !\lambda w.\nabla x)\ \mathsf{in}\ \nabla x : \phi$ for any proposition ϕ, and so the term $\mathsf{rec}(x := !\lambda w.\nabla x)\ \mathsf{in}\ \nabla x$ is a maximal element of $\sqsubseteq_S$. □

In Section 3.11 we can show that the problem of full abstraction reduces to one of showing that $\Gamma \vdash D : \Delta$ iff $\Gamma \models D : \Delta$ iff $[\![\Delta]\!] \leq [\![D]\!][\![\Gamma]\!]$.

3.5 Operational properties: structural equivalence

In the following four sections we shall look at four properties of the operational semantics for graph reduction:

- This section looks at *structural equivalence*.
- Section 3.6 looks at *confluence*.
- Section 3.7 looks at *tagging*.
- Section 3.8 looks at *referential transparency*.

In this section we will look more closely at the structural equivalence $D \equiv E$ that we used to define $D \to E$. Although the structural equivalence is a great help in presenting $D \to E$, it makes proving properties about $D \to E$ harder, since we are always working *modulo* $\equiv$. MILNER (1991) faced a similar problem in proving properties about the polyadic π-calculus, which he solved by giving an independent presentation based on a *commitment* relation.

In this paper, we shall continue to prove properties about declarations up to $\equiv$, but we will need to know some properties about $\equiv$. In particular, we shall present a model for declarations modulo $\equiv$, and use this to show that:

- If $(x := !M) \equiv (D, E)$ then $D \equiv \varepsilon$ or $E \equiv \varepsilon$.
- If $(D, E) \equiv (F, G)$ then we can find DF, DG, EF and EG such that $D \equiv (DF, DG)$, $E \equiv (EF, EG)$, $F \equiv (DF, EF)$ and $G \equiv (DG, EG)$.
- If $\nu x.D \equiv (E, F)$ then either $E \equiv \nu x.G$ and $D \equiv (G, F)$ or $F \equiv \nu x.G$ and $D \equiv (E, G)$.
- If $\nu x.D \equiv \nu y.E$ and $x \neq y$ then either $D \equiv [x/y]E[x/y]$ or $E \equiv \nu x.F$ and $D \equiv \nu y.F$.
- $\nu x.D \not\equiv (x := !M)$

We can use $\equiv$ to convert any declaration into *standard form* $\nu\vec{x}.D$ where D contains no local variables. For example:

$$(x := !\nabla y, \mathsf{local}\, x := !\nabla w\ \mathsf{in}\ y := !x) \equiv \nu z.(x := !\nabla y, y := !\nabla z, z := !\nabla w)$$

DEFINITION. Let the ν-*less* declarations be:

- ε is ν-less.
- $x := !M$ and $x := ?M$ are ν-less.
- D, E is ν-less iff D and E are ν-less.

D is *standard* iff $D = \nu\vec{x}.E$ and E is ν-less. D can be *standardized* iff $D \equiv E$ and E is standard. □

PROPOSITION 20. *Any D can be standardized.*

PROOF. An induction on D. □

The model for declarations consists of a 4-tuple (X, Y, Z, f) where:

- $X \subseteq V$ is the set of written variables.
- $Y \subseteq V \setminus X$ is the set of hidden variables.
- $Z \subseteq X \cup Y$ is the set of tagged variables.
- $f : X \cup Y \to \mathsf{Lam}$ is the term associated with each variable.

For example, the model for $\nu y.(x := !M, y := ?N)$ is:

$$(\{x\}, \{y\}, \{x\}, \{(x, M), (y, N)\})$$

We shall consider this model up to α-conversion of hidden variables, for example if z is fresh then we shall equate:

$$\begin{aligned}&(\{x\}, \{y\}, \{x\}, \{(x, M), (y, N)\})\\ &= (\{x\}, \{z\}, \{x\}, \{(x, M[z/y]), (y, N[z/y])\})\end{aligned}$$

In summary:

DEFINITION. An *abstract declaration* is a 4-tuple (X, Y, Z, f) such that:

$$X \subseteq V \qquad Y \subseteq V \setminus X \qquad Z \subseteq X \cup Y \qquad f : X \cup Y \to \mathsf{Lam}$$

$(X, Y, Z, f) = (X, Y', Z', f')$ iff we can find orderings $\vec{x}$ of Y and $\vec{y}$ of Y', and fresh $\vec{z}$ such that:

$$[\vec{z}/\vec{x}][Z] = [\vec{z}/\vec{y}][Z'] \qquad \forall x.\,(f([\vec{x}/\vec{z}]x))[\vec{z}/\vec{x}] = (f'([\vec{y}/\vec{z}]x))[\vec{z}/\vec{y}]$$

Then the semantics of declarations is given by:

$$\begin{aligned}\partial[\![\varepsilon]\!] &= (\emptyset, \emptyset, \emptyset, \emptyset)\\ \partial[\![x := !M]\!] &= (\{x\}, \emptyset, \{x\}, \{x \mapsto M\})\\ \partial[\![x := ?M]\!] &= (\{x\}, \emptyset, \emptyset, \{x \mapsto M\})\end{aligned}$$

$$\partial[\![D,E]\!] = (X \cup X', Y \cup Y', Z \cup Z', f \cup f')$$
$$\partial[\![\nu x\,.\,D]\!] = (X \setminus \{x\}, Y \cup \{x\}, Z, f)$$

where $\partial[\![D]\!] = (X,Y,Z,f)$, $\partial[\![E]\!] = (X',Y',Z',f')$ and X, Y, X' and Y' are all disjoint. □

Then we can show that this semantics is fully abstract for $\equiv$.

PROPOSITION 21. *$D \equiv E$ iff $\partial[\![D]\!] = \partial[\![E]\!]$.*

PROOF.

$\Rightarrow$ This proof consists of showing each of the axioms and structural rules for $\equiv$ to be sound.

$\Leftarrow$ This proof consists of showing that standard declarations provide a normal form, up to (νSWAP), (α), (ASSOC), (COMM) and (UNIT). □

We can use this to show the required results about $\equiv$.

PROPOSITION 22.

1. *If $(x := !M) \equiv (D,E)$ then $D \equiv \varepsilon$ or $E \equiv \varepsilon$.*
2. *If $(D,E) \equiv (F,G)$ then we can find DF, DG, EF and EG such that $D \equiv (DF,DG)$, $E \equiv (EF,EG)$, $F \equiv (DF,EF)$ and $G \equiv (DG,EG)$.*
3. *If $\nu x\,.\,D \equiv (E,F)$ then either $E \equiv \nu x\,.\,G$ and $D \equiv (G,F)$ or $F \equiv \nu x\,.\,G$ and $D \equiv (E,G)$.*
4. *If $\nu x\,.\,D \equiv \nu y\,.\,E$ and $x \neq y$ then either $D \equiv [x/y]E[x/y]$ or $E \equiv \nu x\,.\,F$ and $D \equiv \nu y\,.\,F$.*
5. *$\nu x\,.\,D \not\equiv (x := !M)$*

PROOF. Each of these has a simple proof, based on abstract declarations. For example, if $(x := !M) \equiv (D,E)$ then let $\partial[\![D]\!] = (X,Y,Z,f)$, $\partial[\![E]\!] = (X',Y',Z',f')$ for disjont X, X', Y and Y', and so:

$$(X \cup X', Y \cup Y', Z \cup Z', f \cup f') = (\{x\}, \emptyset, \{x\}, \{x \mapsto f\})$$

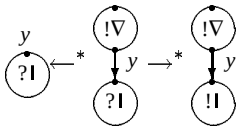
and so either $X = \emptyset$ (and so $\partial[\![D]\!] = \partial[\![\varepsilon]\!]$) or $X' = \emptyset$ (and so $\partial[\![E]\!] = \partial[\![\varepsilon]\!]$). This completes the proof for part 1, and the others follow similarly. □

3.6 Operational properties: confluence

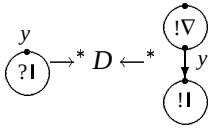
This section looks at the problems raised because the operational semantics given in Section 3.2 is not *confluent*.

DEFINITION. A relation $\mathcal{R}$ is *confluent* iff $x\,\mathcal{R}\,\mathcal{R}^{-1}\,\mathcal{R}\,y$ implies $x\,\mathcal{R}\,\mathcal{R}^{-1}\,y$. □

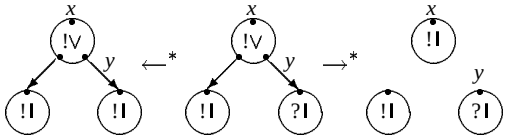
Confluence is (as we shall see below) very useful in proving results about an operational semantics. There are two reasons why $\rightarrow^*$ is not confluent. The first, mentioned in Section 1.2, is due to garbage collection, since:



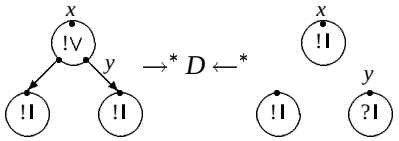
but there is no declaration D such that:



The second is due to fork updating, since:



but there is no declaration D such that:



In this section we will present a confluent convergent reduction strategy for graph reduction.

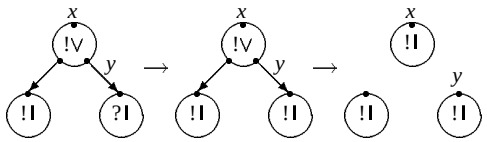
DEFINITION.

- A *reduction strategy* is a relation $\rightarrow_R \subseteq \rightarrow$.
- $D{\Downarrow}_x^R$ iff $\mathtt{tag}_x\,D \rightarrow_R^* E$ and x is in whnf in E.
- $\rightarrow_R$ is *convergent* iff $D{\Downarrow}_x \Leftrightarrow D{\Downarrow}_x^R$ for any closed D. □

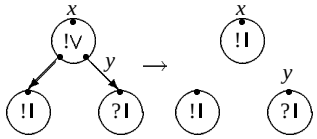
The reduction strategy we will present in this section is the same as $\rightarrow$, except that:

- There is no garbage collection. This bars our first counterexample.
- Fork updating can only take place when both of the nodes pointed to by the fork have been tagged. This bars our second counterexample.

For example, we will allow the reduction:



but not:



Note that we need three axioms to replace the axiom:

$$(x := !y \vee z, y := !\lambda w.\, M) \mapsto (x := !\,\mathsf{I}, y := !\lambda w.\, M)$$

since we have to consider the cases when $x = z$, $y = z$, and $x \neq z \neq y$.

DEFINITION. $\mapsto_c$ is given by axioms:

$$\begin{array}{rc}
(\text{BUILD}) & x := !(\mathsf{rec}D \,\mathsf{in}\, M) \mapsto_c \mathsf{rec}D \,\mathsf{in}\, (x := !M) \\
(\nabla\text{TRAV}) & x := !\nabla y, y := ?M \mapsto_c x := !\nabla y, y := !M \\
(@\text{TRAV}) & x := !y@z, y := ?M \mapsto_c x := !y@z, y := !M \\
(\vee\text{TRAV}) & x := !y \vee z, y := ?M \mapsto_c x := !y \vee z, y := !M \\
(\nabla\text{UPD}) & x := !\nabla y, y := !\lambda w.\, M \mapsto_c x := !\lambda w.\, M, y := !\lambda w.\, M \\
(@\text{UPD}) & x := !y@z, y := !\lambda w.\, M \mapsto_c x := !M[z/w], y := !\lambda w.\, M \\
(\vee\text{UPD}a) & x := !y \vee z, y := !\lambda w.\, M, z := !N \mapsto_c x := !\,\mathsf{I}, y := !\lambda w.\, M, z := !N \\
(\vee\text{UPD}b) & x := !y \vee y, y := !\lambda w.\, M \mapsto_c x := !\,\mathsf{I}, y := !\lambda w.\, M \\
(\vee\text{UPD}c) & x := !y \vee x, y := !\lambda w.\, M \mapsto_c x := !\,\mathsf{I}, y := !\lambda w.\, M
\end{array}$$

and structural rules:

$$(\text{L})\ \frac{D \mapsto_c E}{D, F \mapsto_c E, F} \qquad (\text{R})\ \frac{D \mapsto_c E}{F, D \mapsto_c F, E} \qquad (\nu)\ \frac{D \mapsto_c E}{\nu x.\, D \mapsto_c \nu x.\, E}$$

Then $D \to_c E$ iff $D \equiv \mapsto_c \equiv E$. □

In the rest of this section, we shall show that $\to_c$ is convergent and that $\to_c^*$ is confluent.

To begin with, we can show some properties of tag:

PROPOSITION 23.

1. $\mathsf{tag}_x(\mathsf{tag}_y D) = \mathsf{tag}_y(\mathsf{tag}_x D)$
2. $\mathsf{tag}_x(\mathsf{tag}_x D) = \mathsf{tag}_x D$
3. *If* $D \equiv E$ *then* $\mathsf{tag}_x D \equiv \mathsf{tag}_x E$.
4. *If* $D \to E$ *then* $\mathsf{tag}_x D \to^{\leq 1} \mathsf{tag}_x E$.

5. *If x is in whnf in D then x is in whnf in* $\mathsf{tag}_y D$.
6. *If $x \neq y$ then x is in whnf in D iff x is in whnf in* $\mathsf{tag}_y D$.

PROOF.

1. An induction on D.
2. An induction on D.
3. An induction on the proof of $D \equiv E$.
4. An induction on the proof of $D \to E$.
5. An induction on the proof that x is in whnf in D.
6. $\Rightarrow$ An induction on the proof that x is in whnf in D.
   $\Leftarrow$ An induction on the proof that x is in whnf in $\mathsf{tag}_y D$. □

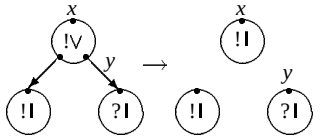
We can show that reduction is independent of the choice of variables, so if y is fresh then $[y/x]D[y/x]$ has the same behaviour as D:

PROPOSITION 24. *If y is fresh and $x \neq z$ then:*

1. *If* $D \to E$ *then* $[y/x]D[y/x] \to [y/x]E[y/x]$.
2. $D{\Downarrow_z}$ *iff* $[y/x]D[y/x]{\Downarrow_z}$.
3. $D{\Downarrow_x}$ *iff* $[y/x]D[y/x]{\Downarrow_y}$.

PROOF. Part 1 is an induction on the proof of $D \to E$, and parts 2 and 3 follow. □

Any reduction $D \to E$ is a reduction of the form $\nu\vec{x}.\,(F, G) \to \nu\vec{x}.\,(F, H)$ where $G \mapsto H$ is an axiom. For example, the reduction:



can be given as:

$$\nu z.\,(x := !z \vee y, z := !\,\mathsf{I}, y := ?\,\mathsf{I}) \to \nu z.\,(x := !\,\mathsf{I}, z := !\,\mathsf{I}, y := ?\,\mathsf{I})$$

and $(x := !z \vee y, z := !\,\mathsf{I}) \mapsto (x := !\,\mathsf{I}, z := !\,\mathsf{I})$ is an axiom.

PROPOSITION 25.

1. *If* $D \to E$ *then* $D \equiv \nu\vec{x}.\,(F, G)$, $E \equiv \nu\vec{x}.\,(F, H)$ *and* $G \mapsto H$ *is an axiom.*
2. *If* $D \to_c E$ *then* $D \equiv \nu\vec{x}.\,(F, G)$, $E \equiv \nu\vec{x}.\,(F, H)$ *and* $G \mapsto_c H$ *is an axiom.*

PROOF.

1. An induction on the proof of $D \to E$.
2. An induction on the proof of $D \to_c E$. □

We can use this to show that any reduction $D \equiv \nu x.\, D' \to_c E$ must have come from a reduction $D' \to_c E'$ and $E \equiv \nu x.\, E'$. This means that whether a variable is

local or global makes no differene to the reduction stragegy $\rightarrow_c$. This is not true of $\rightarrow$, because of garbage collection.

PROPOSITION 26. *If $D \equiv \nu x\,.\,D' \rightarrow_c E$ then $D' \rightarrow_c E'$ and $E \equiv \nu x\,.\,E'$.*

PROOF. By Proposition 25.2:

$$\nu x\,.\,D' \equiv \nu \vec{x}\,.\,(F,G) \qquad E \equiv \nu \vec{x}\,.\,(F,H) \qquad G \mapsto_c H \quad \text{is an axiom}$$

Then we can α-convert so that $x \notin \vec{x}$, so by Proposition 22.4:

- either we have:

  $$\vec{x} = \vec{y}y\vec{z} \qquad D' \equiv \nu\vec{y}\vec{z}\,.\,([x/y](F,G)[x/y])$$

  and so, since all the axioms for $\mapsto_c$ are preserved by α-conversion:

  $$D' \equiv \nu\vec{y}\vec{z}\,.\,([x/y](F,G)[x/y]) \mapsto_c \nu\vec{y}\vec{z}\,.\,([x/y](F,H)[x/y])$$

  and:

  $$E \equiv \nu\vec{x}\,.\,(F,H) \equiv \nu x\,.\,\nu\vec{y}\vec{z}\,.\,([x/y](F,H)[x/y])$$

- or we have:

  $$(F,G) \equiv \nu x\,.\,D'' \qquad D' \equiv \nu\vec{x}\,.\,D''$$

  and so, by Proposition 22.3 and the fact that all the axioms for $\mapsto_c$ involve ν-less declarations:

  $$F \equiv \nu x\,.\,F' \qquad D' \equiv \nu\vec{x}\,.\,(F',G)$$

  and so:

  $$D' \equiv \nu\vec{x}\,.\,(F',G) \mapsto_c \nu\vec{x}\,.\,(F',H)$$

  and:

  $$E \equiv \nu\vec{x}\,.\,(F,H) \equiv \nu x\,.\,\nu\vec{x}\,.\,(F',H)$$

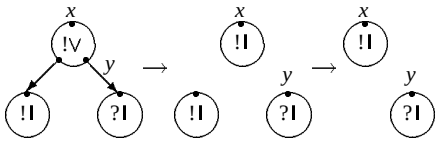
In either case we have found an E' such that $D' \rightarrow_c E'$ and $E \equiv \nu x\,.\,E'$. □

We would now like to show that $\rightarrow_c$ is convergent, that is $D\Downarrow_x^c$ iff $D\Downarrow_x$. Unfortunately, it is *not* the case that any reduction $D \rightarrow^* E$ can be matched by a reduction $D \rightarrow_c^* E$, since:

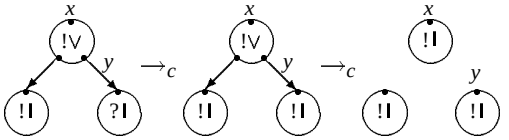
- The reduction $D \rightarrow^* E$ might include garbage collection.
- The reduction $D \rightarrow^* E$ may include fork updating with an untagged node.

However, it is the case that any reduction $D \rightarrow^* E$ can be matched by a reduction $D \rightarrow_c^* F$, where F can be garbage collected to a declaration with fewer untagged

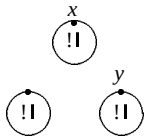
nodes than E. For example, the reduction:



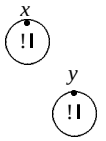
can be matched by the reduction:



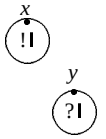
and:



can be garbage collected to:



which has fewer untagged nodes than:



More formally, we shall show that if $D \rightarrow^* E$ then $D \rightarrow_c^* \rightarrow_\gamma \leq_? E$, where $D \rightarrow_\gamma E$ means 'D can be garbage collected to E' and $D \leq_? E$ means 'D has fewer untagged nodes than E'.

DEFINITION. $D \leq_? E$ is given by axioms:

$$(\text{REFL}) \quad D \leq_? D \qquad (?) \quad (x := !M) \leq_? (x := ?M)$$

and structural rules:

$$(\text{LR}) \ \frac{D \leq_? E \quad D' \leq_? E'}{D, D' \leq_? E, E'} \qquad (\nu) \ \frac{D \leq_? E}{\nu x\,.\,D \leq_? \nu x\,.\,E}$$

$D \rightarrow_\gamma E$ iff $D \rightarrow E$ is proved using the (γ) axiom. □

PROPOSITION 27. *For closed D:*

1. $\leq_?$ *is a partial order.*
2. *If* $D \leq_? \equiv E$ *iff* $D \equiv \leq_? E$.
3. *If* $D \to_\gamma \to_c E$ *then* $D \to_c \to_\gamma E$.
4. *If* $D \leq_? \to_c E$ *then* $D \to_c^{\leq 1} \leq_? E$.
5. *If* $D \leq_? \to_\gamma E$ *then* $D \to_\gamma \leq_? E$.
6. *If* $D \to E$ *then* $D \to_c^* \to_\gamma^* \leq_? E$.
7. *If* $D \to^* E$ *then* $D \to_c^* \to_\gamma^* \leq_? E$.

PROOF.

1. By definition, $\leq_?$ is reflexive. By induction on the proof of $D \leq_? E$, we can show that if $D \leq_? E \leq_? D$ then $D = E$, and so $\leq_?$ is antisymmetric. By induction on the proof of $D \leq_? E$, we can show that if $D \leq_? E \leq_? F$ then $D \leq_? F$, and so $\leq_?$ is transitive.

2. An induction on the proof of $\equiv$.

3. If $D \to_\gamma F \to_c E$ then by Proposition 25.1:

$$D \equiv \nu\vec{x}.(G, \nu(\mathsf{wv}H).H) \qquad F \equiv \nu\vec{x}.G \tag{2}$$

Then by Proposition 26:

$$E \equiv \nu\vec{x}.I \qquad G \to_c I \tag{3}$$

Thus:

$$\begin{array}{ll} D & \\ \quad \equiv & \nu\vec{x}.(G, \nu(\mathsf{wv}H).H) \quad \text{(Eqn 2)} \\ \quad \to_c & \nu\vec{x}.(I, \nu(\mathsf{wv}H).H) \quad \text{(Eqn 3)} \\ \quad \to_\gamma & \nu\vec{x}.I \quad (\gamma) \\ \quad \equiv & E \quad \text{(Eqn 3)} \end{array}$$

4. If $D \leq_? F \to_c E$ then by Proposition 25.2 we have:

$$F \equiv \nu\vec{x}.(G,H) \qquad E \equiv \nu\vec{x}.(G,I) \qquad H \mapsto_c I \quad \text{is an axiom} \tag{4}$$

Then by part 2, and the definition of $\leq_?$:

$$D \equiv \nu\vec{x}.(G',H') \qquad G' \leq_? G \qquad H' \leq_? H \tag{5}$$

Then by analysis of the axiom $H \mapsto_c I$, we can find:

$$I' \leq_? I \qquad H' \to_c^* I' \tag{6}$$

Thus:

$$\begin{array}{ll} D & \\ \quad \equiv & \nu\vec{x}.(G',H') \quad \text{(Eqn 5)} \\ \quad \to_c^* & \nu\vec{x}.(G',I') \quad \text{(Eqn 6)} \end{array}$$

$$\begin{array}{ll} & \leq_? \ \nu\vec{x}.(G,I') \quad \text{(Eqn 5)} \\ & \leq_? \ \nu\vec{x}.(G,I) \quad \text{(Eqn 6)} \\ & \equiv \ E \quad \text{(Eqn 4)} \end{array}$$

And so by part 2, $D \to_c^* \leq_? E$.

5. Similar.

6. If $D \to E$ then by Proposition 25.1 we have:

$$D \equiv \nu\vec{x}.(F,G) \qquad E \equiv \nu\vec{x}.(F,H) \qquad G \mapsto H \quad \text{is an axiom} \tag{7}$$

Then we proceed by analysis on the axiom $G \mapsto H$:

(γ) $D \to_\gamma E$, so $D \to_c^* \to_\gamma^* \leq_? E$.

($\vee$UPD) This axiom has:

$$G = x := !y \vee z, y := !\lambda w.M$$
$$H = x := !\mathsf{I}, y := !\lambda w.M$$

There are three subcases:

($x = z$) So by ($\vee$UPDc), $G \mapsto_c H$, and so $D \to_c^* \to_\gamma^* \leq_? E$.

($y = z$) So by ($\vee$UPDb), $G \mapsto_c H$, and so $D \to_c^* \to_\gamma^* \leq_? E$.

($x \neq z \neq y$) Since D is closed, $z \in \mathsf{wv}F$, and so *either*:

$$F \equiv \nu\vec{y}.(F', z := !M) \tag{8}$$

and we can α-convert so $\vec{y} \cap \mathsf{fv}\,G = \emptyset$, and so:

$$\begin{array}{ll} D & \\ \quad \equiv & \nu\vec{x}.(F,G) \quad \text{(Eqn 7)} \\ \quad \equiv & \nu\vec{x}.(\nu\vec{y}.(F', z := !M), G) \quad \text{(Eqn 8)} \\ \quad \equiv & \nu\vec{x}\vec{y}.(F', z := !M, G) \quad (\nu\text{MIG}) \\ \quad \to_c & \nu\vec{x}\vec{y}.(F', z := !M, H) \quad (\vee\text{UPD}a) \\ \quad \equiv & \nu\vec{x}.(\nu\vec{y}.(F', z := !M), H) \quad (\nu\text{MIG}) \\ \quad \equiv & \nu\vec{x}.(F,H) \quad \text{(Eqn 8)} \\ \quad \equiv & E \quad \text{(Eqn 7)} \end{array}$$

or:

$$F \equiv \nu\vec{y}.(F', z := ?M) \tag{9}$$

and we can α-convert so $\vec{y} \cap \mathsf{fv}\,G = \emptyset$, and so:

$$\begin{array}{ll} D & \\ \quad \equiv & \nu\vec{x}.(F,G) \quad \text{(Eqn 7)} \\ \quad \equiv & \nu\vec{x}.(\nu\vec{y}.(F', z := ?M), G) \quad \text{(Eqn 9)} \\ \quad \equiv & \nu\vec{x}\vec{y}.(F', z := ?M, G) \quad (\nu\text{MIG}) \end{array}$$

$$
\begin{array}{rll}
\to_c \nu\vec{x}\vec{y}.\,(F', z := !M, G) & & (\vee\text{TRAV}) \\
\to_c \nu\vec{x}\vec{y}.\,(F', z := !M, H) & & (\vee\text{UPD}a) \\
\equiv \nu\vec{x}.\,(\nu\vec{y}.\,(F', z := !M), H) & & (\nu\text{MIG}) \\
\leq_? \nu\vec{x}.\,(\nu\vec{y}.\,(F', z := ?M), H) & & (\text{Defn of } \leq_?) \\
\equiv \nu\vec{x}.\,(F, H) & & (\text{Eqn 9}) \\
\equiv E & & (\text{Eqn 7})
\end{array}
$$

Thus, $D \to_c^* \to_\gamma^* \leq_? E$.

(OTHERS) The other axioms are axioms of $\mapsto_c$, and so $D \to_c^* \to_\gamma^* \leq_? E$.

7. Let $D \to^n E$, and proceed by induction on n:

   ($n = 0$) When $n = 0$, $D \equiv E$, so trivially $D \to_c^* \to_\gamma^* \leq_? E$.

   ($n > 0$) When $n > 0$, we have:

$$
\begin{array}{rll}
D \to^n E & & \\
\Rightarrow D \to^{n-1} \to E & & (\text{Defn of } \to^n) \\
\Rightarrow D \to_c^* \to_\gamma^* \leq_? \to E & & (\text{Indn}) \\
\Rightarrow D \to_c^* \to_\gamma^* \leq_? \to_c^* \to_\gamma^* \leq_? E & & (\text{Part 6}) \\
\Rightarrow D \to_c^* \to_\gamma^* \to_c^* \leq_? \to_\gamma^* \leq_? E & & (\text{Part 4}) \\
\Rightarrow D \to_c^* \to_c^* \to_\gamma^* \leq_? \to_\gamma^* \leq_? E & & (\text{Part 3}) \\
\Rightarrow D \to_c^* \to_c^* \to_\gamma^* \to_\gamma^* \leq_? \leq_? E & & (\text{Part 5}) \\
\Rightarrow D \to_c^* \to_\gamma^* \leq_? E & & (\text{Transitivity})
\end{array}
$$

Thus $D \to_c^* \to_\gamma^* \leq_? E$. □

We can then show that $\to_c$ is convergent.

PROPOSITION 28. *For closed D:*

1. *If x is in whnf in D and $E \leq_? D$ then x is in whnf in E.*
2. *If x is in whnf in D and $E \to_\gamma D$ then x is in whnf in E.*
3. *If x is in whnf in D and $D \to E$ then x is in whnf in E.*
4. *$\to_c$ is convergent.*

PROOF.

1. An induction on the proof of $E \leq_? D$.
2. By Proposition 25.1 $D \equiv \nu\vec{x}.\,(F, \nu(\mathsf{wv}G).\,G)$ and $E \equiv \nu\vec{x}.\,F$. Then since x is in whnf in D, x is in whnf in F, so x is in whnf in E.
3. By Proposition 25.1 $D \equiv \nu\vec{x}.\,(F, G)$, $E \equiv \nu\vec{x}.\,(F, H)$ and $G \mapsto H$ is an axiom. Then since x is in whnf in D, either:
   - x is in whnf in F, so x is in whnf in E.
   - x is in whnf in G, so by case analysis of each axiom, x is in whnf in H, so x is in whnf in E.

4. From the above:

$$
\begin{array}{rll}
D\Downarrow_x^c & & \\
\Rightarrow \mathsf{tag}_x D \to_c^* E, x \text{ is in whnf in } E & & (\text{Defn of } \Downarrow_x^c) \\
\Rightarrow \mathsf{tag}_x D \to^* E, x \text{ is in whnf in } E & & (\to_c \subseteq \to) \\
\Rightarrow D\Downarrow_x & & (\text{Defn of } \Downarrow_x) \\
\Rightarrow \mathsf{tag}_x D \to^* E, x \text{ is in whnf in } E & & (\text{Defn of } \Downarrow_x) \\
\Rightarrow \mathsf{tag}_x D \to_c^* F \to_\gamma^* G \leq_? E, x \text{ is in whnf in } E & & (\text{Propn 27.7}) \\
\Rightarrow \mathsf{tag}_x D \to_c^* F \to_\gamma^* G \leq_? E, x \text{ is in whnf in } G & & (\text{Part 1}) \\
\Rightarrow \mathsf{tag}_x D \to_c^* F \to_\gamma^* G \leq_? E, x \text{ is in whnf in } F & & (\text{Part 2}) \\
\Rightarrow D\Downarrow_x^c & & (\text{Defn of } \Downarrow_x^c)
\end{array}
$$

Thus $\to_c$ is convergent. □

We can use the fact that $\to_c$ is convergent to show that convergence is not affected by local variables:

PROPOSITION 29. *For closed D, if $w \neq x$ then $D\Downarrow_x$ iff $\nu w.\,D\Downarrow_x$.*

PROOF.

$\Rightarrow$ If $D\Downarrow_x$ then we can find E such that $\mathsf{tag}_x D \to^* E$ and x is in whnf in E, so $\mathsf{tag}_x(\nu w.\,D) \to^* \nu w.\,E$, and x is in whnf in $\nu w.\,E$. Thus $\nu w.\,D\Downarrow_x$.

$\Leftarrow$ If $\nu w.\,D\Downarrow_x$ then since $\to_c$ is convergent, $\mathsf{tag}_x(\nu w.\,D) \to_c^* E$, and x is in whnf in E, so by Proposition 26 $E \equiv \nu w.\,F$ and $\mathsf{tag}_x D \to_c^* F$. Since x is in whnf in E, x is in whnf in F, and so $D\Downarrow_x$. □

PROPOSITION 30. *For closed D:*

1. *If $D \equiv (D', x := !\,\mathsf{rec}\,G\,\mathsf{in}\,M) \to_c E$ then $E \equiv (D', \mathsf{local}\,G\,\mathsf{in}\,x := !M)$ or $E \equiv (E', x := !\,\mathsf{rec}\,G\,\mathsf{in}\,M)$ and $\forall N.\,(D', x := !N) \to_c (E', x := !N)$.*
2. *If $D \equiv (D', x := !\nabla y) \to_c E$*
   then $D' \equiv \nu\vec{x}.\,(D'', y := ?M)$ and $E \equiv \nu\vec{x}.\,(D'', y := !M, x := !\nabla y)$
   or $D' \equiv \nu\vec{x}.\,(D'', y := !\lambda w.\,M)$ and $E \equiv \nu\vec{x}.\,(D'', y := !\lambda w.\,M, x := !\lambda w.\,M)$
   or $E \equiv (E', x := !\nabla y)$ and $\forall M.\,(D', x := !M) \to_c (E', x := !M)$.
3. *If $D \equiv (D', x := !y@z) \to_c E$*
   then $D' \equiv \nu\vec{x}.\,(D'', y := ?M)$ and $E \equiv \nu\vec{x}.\,(D'', y := !M, x := !y@z)$
   or $D' \equiv \nu\vec{x}.\,(D'', y := !\lambda w.\,M)$ and $E \equiv \nu\vec{x}.\,(D'', y := !\lambda w.\,M, x := !M[z/w])$
   or $E \equiv (E', x := !y@z)$ and $\forall M.\,(D', x := !M) \to_c (E', x := !M)$.
4. *If $D \equiv (D', x := !y \vee z) \to_c E$*
   then $D' \equiv \nu\vec{x}.\,(D'', y := ?M)$ and $E \equiv \nu\vec{x}.\,(D'', y := !M, x := !y \vee z)$
   or $D' \equiv \nu\vec{x}.\,(D'', z := ?M)$ and $E \equiv \nu\vec{x}.\,(D'', z := !M, x := !y \vee z)$
   or $D' \equiv \nu\vec{x}.\,(D'', y := !\lambda w.\,M)$ and $E \equiv \nu\vec{x}.\,(D'', y := !\lambda w.\,M, x := !\,\mathsf{I})$
   or $D' \equiv \nu\vec{x}.\,(D'', z := !\lambda w.\,M)$ and $E \equiv \nu\vec{x}.\,(D'', z := !\lambda w.\,M, x := !\,\mathsf{I})$
   or $E \equiv (E', x := !y \vee z)$ and $\forall M.\,(D', x := !M) \to_c (E', x := !M)$.

5. *If* $D \equiv (D', x := !\lambda w.M) \rightarrow_c E$ *then* $E' \equiv (E', x := !\lambda w.M)$.
6. *If* $D \equiv (D', x := ?M) \rightarrow_c E$ *then* $E \equiv (D', x := !M)$ *or* $E \equiv (E', x := ?M)$ *and* $D' \rightarrow_c E'$.

PROOF. These all have similar proofs, we shall prove part 1 as an example. If $(D', x := !\,\mathsf{rec}\,G\,\mathsf{in}\,M) \rightarrow_c E$ then by Proposition 25.1 we have:

$$\begin{array}{c}(D', x := !\,\mathsf{rec}\,G\,\mathsf{in}\,M) \equiv \nu\vec{x}.(H,I) \\ E \equiv \nu\vec{x}.(H,J) \qquad I \mapsto J \quad \text{is an axiom}\end{array} \tag{10}$$

Then by Propositions 22.3 and 22.5:

$$D' \equiv \nu\vec{x}.D'' \qquad (D'', x := !\,\mathsf{rec}\,G\,\mathsf{in}\,M) \equiv (H,I) \tag{11}$$

So by Propositions 22.1 and 22.2 either:

- we have:

$$H \equiv (K, x := !\,\mathsf{rec}\,G\,\mathsf{in}\,M) \qquad D'' \equiv (K,I) \tag{12}$$

and so:

$$\begin{array}{ll} E & \\ \quad \equiv \nu\vec{x}.(H,J) & \text{(Eqn 10)} \\ \quad \equiv \nu\vec{x}.(K, x := !\,\mathsf{rec}\,G\,\mathsf{in}\,M, J) & \text{(Eqn 12)} \\ \quad \equiv \nu\vec{x}.(K,J), x := !\,\mathsf{rec}\,G\,\mathsf{in}\,M & (\nu\text{MIG}) \end{array}$$

and for any N:

$$\begin{array}{ll} D', x := !N & \\ \quad \equiv\ \nu\vec{x}.D'', x := !N & \text{(Eqn 11)} \\ \quad \equiv\ \nu\vec{x}.(K,I), x := !N & \text{(Eqn 12)} \\ \quad \rightarrow_c \nu\vec{x}.(K,J), x := !N & (I \mapsto_c J) \end{array}$$

- or we have:

$$I \equiv (K, x := !\,\mathsf{rec}\,G\,\mathsf{in}\,M) \qquad D'' \equiv (H,K) \tag{13}$$

so the axiom is either:

(BUILD) so we have:

$$x := !\,\mathsf{rec}\,G\,\mathsf{in}\,M \mapsto \mathsf{local}\,G\,\mathsf{in}\,x := !M$$

in which case:

$$I \equiv x := !\,\mathsf{rec}\,G\,\mathsf{in}\,M \qquad J \equiv \mathsf{local}\,G\,\mathsf{in}\,x := !M \qquad K \equiv \varepsilon \tag{14}$$

and so:

$$\begin{array}{ll} E & \\ \quad \equiv \nu\vec{x}.(H,J) & \text{(Eqn 10)} \\ \quad \equiv \nu\vec{x}.(H, \mathsf{local}\,G\,\mathsf{in}\,x := !M) & \text{(Eqn 14)} \end{array}$$

$$\begin{array}{ll} \quad \equiv (\nu\vec{x}.H, \mathsf{local}\,G\,\mathsf{in}\,x := !M) & (\nu\text{MIG}) \\ \quad \equiv (\nu\vec{x}.(H,K), \mathsf{local}\,G\,\mathsf{in}\,x := !M) & \text{(Eqn 14)} \\ \quad \equiv (\nu\vec{x}.(D''), \mathsf{local}\,G\,\mathsf{in}\,x := !M) & \text{(Eqn 13)} \\ \quad \equiv (D', \mathsf{local}\,G\,\mathsf{in}\,x := !M) & \text{(Eqn 11)} \end{array}$$

(∨UPDa) We have:

$$\begin{array}{l} z := !y \vee x, y := !\lambda w.N, x := !\,\mathsf{rec}\,G\,\mathsf{in}\,M \\ \quad \mapsto z := !\,\mathsf{I}, y := !\lambda w.N, x := !\,\mathsf{rec}\,G\,\mathsf{in}\,M \end{array}$$

in which case:

$$\begin{array}{l} I \equiv z := !y \vee x, y := !\lambda w.N, x := !\,\mathsf{rec}\,G\,\mathsf{in}\,M \\ J \equiv z := !\,\mathsf{I}, y := !\lambda w.N, x := !\,\mathsf{rec}\,G\,\mathsf{in}\,M \\ K \equiv z := !y \vee x, y := !\lambda w.N \end{array} \tag{15}$$

and so:

$$\begin{array}{ll} E & \\ \quad \equiv \nu\vec{x}.(H,J) & \text{(Eqn 10)} \\ \quad \equiv \nu\vec{x}.(H, z := !\,\mathsf{I}, y := !\lambda w.N, x := !\,\mathsf{rec}\,G\,\mathsf{in}\,M) & \text{(Eqn 15)} \\ \quad \equiv \nu\vec{x}.(H, z := !\,\mathsf{I}, y := !\lambda w.N), x := !\,\mathsf{rec}\,G\,\mathsf{in}\,M & (\nu\text{MIG}) \end{array}$$

and for any N:

$$\begin{array}{ll} D', x := !N & \\ \quad \equiv\ \nu\vec{x}.D'', x := !N & \text{(Eqn 11)} \\ \quad \equiv\ \nu\vec{x}.(H,K), x := !N & \text{(Eqn 13)} \\ \quad \equiv\ \nu\vec{x}.(H, z := !y \vee x, y := !\lambda w.N), x := !N & \text{(Eqn 15)} \\ \quad \equiv\ \nu\vec{x}.(H, z := !y \vee x, y := !\lambda w.N, x := !N) & (\nu\text{MIG}) \\ \quad \rightarrow_c \nu\vec{x}.(H, z := !\,\mathsf{I}, y := !\lambda w.N, x := !N) & (\vee\text{UPD}a) \\ \quad \equiv\ \nu\vec{x}.(H, z := !\,\mathsf{I}, y := !\lambda w.N), x := !N & (\nu\text{MIG}) \end{array}$$

The other propositions are proved similarly. □

PROPOSITION 31. $\rightarrow_c^{\leq 1}$ *is confluent.*

PROOF. If $D \rightarrow_c^{\leq 1} E$ and $D \rightarrow_c^{\leq 1} F$ then either $D \equiv E$, $D \equiv F$, or $D \rightarrow_c E$ and $D \rightarrow_c F$. The first two cases are trivial.

If $D \rightarrow_c E$ and $D \rightarrow_c F$ then by Propositions 25.1 we can find:

$$D \equiv \nu\vec{x}.(G,I) \qquad E \equiv \nu\vec{x}.(G,J) \qquad I \mapsto J \quad \text{is an axiom} \tag{16}$$

and by 26 we can find:

$$F \equiv \nu\vec{x}.H \qquad (G,I) \rightarrow_c H \tag{17}$$

Then we proceed by case analysis on which axiom was used to show $I \mapsto J$. These all have similar proofs, so we shall just show the case for (BUILD). We have:

$$I \equiv x := !\,\mathsf{rec}K\,\mathsf{in}\,M \qquad J \equiv \mathsf{local}\,K\,\mathsf{in}\,(x := !M) \tag{18}$$

Then by Proposition 30.1 either:

- we have:

$$H \equiv (G, \mathsf{local}\, K \,\mathsf{in}\, x := !M) \tag{19}$$

in which case:

$$\begin{array}{lll} E & & \\ & \equiv \nu\vec{x}.(G,J) & \text{(Eqn 16)} \\ & \equiv \nu\vec{x}.(G,\mathsf{local}\, K \,\mathsf{in}\, x := !M) & \text{(Eqn 18)} \\ & \equiv \nu\vec{x}.H & \text{(Eqn 19)} \\ & \equiv F & \text{(Eqn 17)} \end{array}$$

- or we have:

$$H \equiv (L, x := !\,\mathsf{rec} K \,\mathsf{in}\, M \tag{20}$$

and for any N:

$$(G, x := !N) \rightarrow_c (L, x := !N) \tag{21}$$

Then let $\vec{y}$ be $\mathsf{wv} K$ and let $\vec{z}$ be fresh so:

$$\begin{array}{lll} E & & \\ & \equiv \nu\vec{x}.(G,J) & \text{(Eqn 16)} \\ & \equiv \nu\vec{x}.(G,\mathsf{local}\, K \,\mathsf{in}\, x := !M) & \text{(Eqn 18)} \\ & \equiv \nu\vec{x}.(G,\nu\vec{z}.([\vec{z}/\vec{y}]K[\vec{z}/\vec{y}], x := !M[\vec{z}/\vec{y}])) & \text{(Defn of local)} \\ & \equiv \nu\vec{x}\vec{z}.(G,[\vec{z}/\vec{y}]K[\vec{z}/\vec{y}], x := !M[\vec{z}/\vec{y}]) & (\nu\text{MIG}) \\ & \rightarrow_c \nu\vec{x}\vec{z}.(L,[\vec{z}/\vec{y}]K[\vec{z}/\vec{y}], x := !M[\vec{z}/\vec{y}]) & \text{(Eqn 21)} \\ & \equiv \nu\vec{x}.(L,\nu z.([\vec{z}/\vec{y}]K[\vec{z}/\vec{y}], x := !M[\vec{z}/\vec{y}])) & (\nu\text{MIG}) \\ & \equiv \nu\vec{x}.(L,\mathsf{local}\, K \,\mathsf{in}\, x := !M) & \text{(Defn of local)} \\ & \leftarrow_c \nu\vec{x}.(L, x := !\,\mathsf{rec} K \,\mathsf{in}\, M) & (\text{BUILD}) \\ & \equiv \nu\vec{x}.H & \text{(Eqn 20)} \\ & \equiv F & \text{(Above)} \end{array}$$

The other cases are similar, and so $\rightarrow_c^{\leq 1}$ is confluent. □

PROPOSITION 32. *For closed D, if $D \rightarrow_c E$ then $D{\Downarrow_x}$ iff $E{\Downarrow_x}$.*

PROOF.

$\Rightarrow$ If $D \rightarrow_c E$ then we have the following diagram:

$$\begin{array}{ccc} D & \rightarrow_c \cdots \rightarrow_c & F \\ \downarrow_c & & \\ E & & \end{array}$$

where x is in whnf in F. Then since $\rightarrow_c^{\leq 1}$ is confluent we can complete the diagram as:

$$\begin{array}{ccccc} D & \rightarrow_c & \cdots & \rightarrow_c & F \\ \downarrow_c & \downarrow_c^{\leq 1} & & \downarrow_c^{\leq 1} & \downarrow_c^{\leq 1} \\ E & \rightarrow_c^{\leq 1} & \cdots & \rightarrow_c^{\leq 1} & G \end{array}$$

Since x is in whnf in F, x is in whnf in G, and so $E{\Downarrow_x}$.

$\Leftarrow$ Follows from the definition of $D \rightarrow_x$. □

3.7 Operational properties: independence from tagging

The denotational semantics for tagged declarations $(x := !M)$ and untagged declarations $(x := ?M)$ is the same, despite the fact that tagged and untagged declarations have very different operational behaviour. For example the declaration:

$$\nu y.(x := !\,\mathsf{I}, y := !\Omega)$$

can diverge, whereas the declaration:

$$\nu y.(x := !\,\mathsf{I}, y := ?\Omega)$$

cannot. However, both of them can reach whnf at x, and since the testing equivalence is based on reaching whnf, they are testing equivalent. In this section, we will show that *convergence is independent of tagging*, that is:

$$D{\Downarrow_x} \text{ iff } \mathsf{tag}_y D{\Downarrow_x}$$

and that this means that *convergence is independent of reduction*, that is:

$$\text{if } D \rightarrow E \text{ then } D{\Downarrow_x} \Leftrightarrow E{\Downarrow_x}$$

For example, even though the declaration:

$$\nu y.(x := !\,\mathsf{I}, y := !\Omega)$$

can diverge, and the declaration:

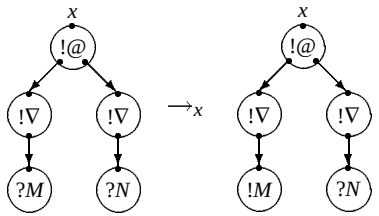
$$x := !\,\mathsf{I}$$

cannot, they have the same convergent behaviour, since:

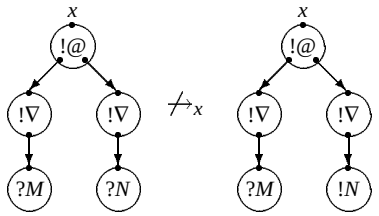
$$\nu y.(x := !\,\mathsf{I}, y := !\Omega) \rightarrow (x := !\,\mathsf{I})$$

In order to show that convergence is independent of tagging, we shall present a reduction strategy $\rightarrow_x$, where a reduction $D \rightarrow_x E$ will take place only when the

reduction is needed in order to evaluate x. For example, we will allow:



since we need to evaluate M in order to evaluate x, but:



since we may not need to evaluate N in order to evaluate x. In the rest of this section we will:

- Define the reduction strategy $\rightarrow_x$
- Show that $D\Downarrow_x$ iff $\mathsf{tag}_x D \rightarrow_x^* E$ and x is in whnf in E.
- Show that if $D \geq_? E \rightarrow_x^* F$ and x is tagged in D then $D \rightarrow_x^* \geq_? F$.
- From this, show that if $D \geq_? E$ then $D\Downarrow_x$ iff $E\Downarrow_x$.
- Show that if $D \leftarrow_\gamma \rightarrow_x E$ then $D \rightarrow_x \leftarrow_\gamma E$.
- From this, show that if $D \rightarrow E$ then $D\Downarrow_x$ iff $E\Downarrow_x$.

First we can define the reduction strategy $\rightarrow_x \subseteq \rightarrow_c$:

DEFINITION. $\rightarrow_x$ is given by axioms:

$$
\begin{array}{rr}
(\text{BUILD}) & D, x := !\,\mathsf{rec} E \text{ in } M \rightarrow_x D, \mathsf{local} E \text{ in } (x := !M) \\
(\nabla\text{TRAV}) & D, x := !\nabla y, y := ?M \rightarrow_x D, x := !\nabla y, y := !M \\
(@\text{TRAV}) & D, x := !y@z, y := ?M \rightarrow_x D, x := !y@z, y := !M \\
(\vee\text{TRAV}) & D, x := !y \vee z, y := ?M \rightarrow_x D, x := !y \vee z, y := !M \\
(\nabla\text{UPD}) & D, x := !\nabla y, y := !\lambda w.\, M \rightarrow_x D, x := !\lambda w.\, M, y := !\lambda w.\, M \\
(@\text{UPD}) & D, x := !y@z, y := !\lambda w.\, M \rightarrow_x D, x := !M[z/w], y := !\lambda w.\, M \\
(\vee\text{UPD}a) & D, x := !y \vee z, y := !\lambda w.\, M, z := !N \rightarrow_x D, x := !\,\mathsf{I}, y := !\lambda w.\, M, z := !N \\
(\vee\text{UPD}b) & D, x := !y \vee y, y := !\lambda w.\, M \rightarrow_x D, x := !\,\mathsf{I}, y := !\lambda w.\, M \\
(\vee\text{UPD}c) & D, x := !y \vee x, y := !\lambda w.\, M \rightarrow_x D, x := !\,\mathsf{I}, y := !\lambda w.\, M
\end{array}
$$

and structural rules:

$$
(\nabla_{\text{IND}})\ \frac{D, x := !\nabla y \rightarrow_y E}{D, x := !\nabla y \rightarrow_x E} \qquad (@_{\text{IND}})\ \frac{D, x := !y@z \rightarrow_y E}{D, x := !y@z \rightarrow_x E}
$$

$$
(\vee_{\text{IND}})\ \frac{D, x := !y \vee z \rightarrow_y E}{D, x := !y \vee z \rightarrow_x E} \qquad (\equiv)\ \frac{D \equiv \rightarrow_x \equiv E}{D \rightarrow_x E}
$$

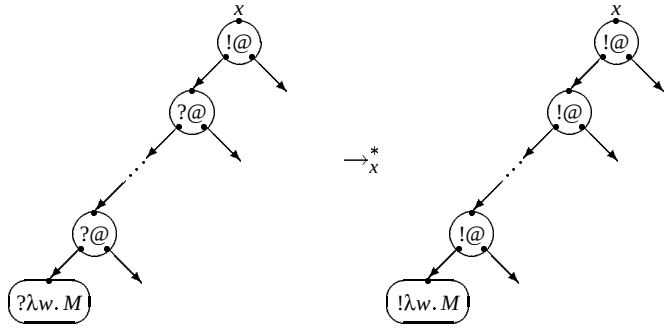
$$
(\nu)\ \frac{D \rightarrow_x E}{\nu y.\, D \rightarrow_x \nu y.\, E}\,[x \neq y]
$$

Let $D \rightarrow_{\neg x} E$ iff $D \rightarrow_c E$ and $D \not\rightarrow_x E$. □

PROPOSITION 33. *If $D \rightarrow_x E$ then $D, F \rightarrow_x E, F$.*

PROOF. An induction on the proof of $D \rightarrow_x E$. □

Informally, $D \rightarrow_x E$ if the reduction $D \rightarrow_c E$ takes place on the x-spine of D, for example:



since each of the nodes that are tagged are on the x-spine. More formally, we can define the x-spine of D to be all the variables where $D \vdash x \prec y$:

DEFINITION. $D \vdash x \prec y$ is given by axioms:

$$
\begin{array}{rr}
(\text{REFL}) & D \vdash x \prec x \\
(\nabla\text{IND}) & D, x := !\nabla y \vdash x \prec y \\
(@\text{IND}) & D, x := !y@z \vdash x \prec y \\
(\vee\text{IND}) & D, x := !y \vee z \vdash x \prec y
\end{array}
$$

and structural rules:

$$
(\text{TRANS})\ \frac{D \vdash x \prec y \prec z}{D \vdash x \prec z} \qquad (\equiv)\ \frac{D \equiv E \vdash x \prec y}{D \vdash x \prec y}
$$

$$
(\nu)\ \frac{D \vdash x \prec y}{\nu z.\, D \vdash x \prec y}\,[x \neq z \neq y]
$$

$D \vdash x \prec y$ is pronounced 'In D, x spines to y'. □

PROPOSITION 34.

1. *If $D \vdash x \prec y$ then $D, E \vdash x \prec y$.*
2. *If $\nu x \,.\, D \vdash y \prec z$ then $D \vdash y \prec z$.*
3. *If $x \neq y \neq z$, w is fresh and $D \vdash x \prec z$ then $[w/y]D[w/y] \vdash x \prec z$.*

PROOF. Inductions on the proof of $\prec$. □

Then we can show that $D \rightarrow_x E$ iff there is a reduction on the x-spine of D:

PROPOSITION 35. *$D \rightarrow_x E$ iff $D \equiv \nu \vec{x} \,.\, F$, $E \equiv \nu \vec{x} \,.\, G$, $F \rightarrow_y G$ is an axiom, and $F \vdash x \prec y$.*

PROOF.

$\Rightarrow$ An induction on the proof of $D \rightarrow_x E$.
$\Leftarrow$ An induction on the proof of $F \vdash x \prec y$. □

PROPOSITION 36. *If $D \vdash x \prec y$ and $D \rightarrow_y E$ then $D \rightarrow_x E$.*

PROOF. By Proposition 35, $D \equiv \nu \vec{x} \,.\, F$, $E \equiv \nu \vec{x} \,.\, G$, $F \vdash y \prec z$ and $F \rightarrow_z G$ is an axiom. Then by Proposition 34.2, $F \vdash x \prec y$, so by (TRANS), $F \vdash x \prec z$, so by Proposition 35, $D \rightarrow_x E$. □

PROPOSITION 37.

1. *If $D \equiv (D', D'')$, $D \vdash x \prec z$, $x \in \mathsf{wv} D'$ and $z \in \mathsf{wv} D''$ then $\exists y \in \mathsf{rv} D' \cap \mathsf{wv} D'' \,.\, D' \vdash x \prec y$.*
2. *If $D \equiv (D', D'')$, $D \vdash x \prec z$, and $x, z \in \mathsf{wv} D'$ then $D' \vdash x \prec z$ or $\exists y \in \mathsf{rv} D' \cap \mathsf{wv} D'' \,.\, D' \vdash x \prec y$.*

PROOF. An induction on the proof of $D \vdash x \prec z$.

1. The only difficult cases are (ν) and (TRANS). In the case of (ν) we have:

$$D = \nu w \,.\, E \qquad E \vdash x \prec z \qquad x \neq w \neq z$$

Then by Proposition 22.3 either:

- $D' \equiv \nu w \,.\, E'$ and $E \equiv (E', D'')$ so by induction we can find $y \in \mathsf{rv} E' \cap \mathsf{wv} D''$ such that $E' \vdash x \prec y$. Then $y \in \mathsf{wv} D''$ so $y \neq w$, so $y \in \mathsf{rv} D' \cap \mathsf{wv} D''$ and by (ν) $D' \vdash x \prec y$.
- $D'' \equiv \nu w \,.\, E''$ and $E \equiv (D', E'')$ so by induction we can find $y \in \mathsf{rv} D' \cap \mathsf{wv} E''$ such that $D' \vdash x \prec y$. Then $y \in \mathsf{rv} D'$ so $y \neq w$, so $y \in \mathsf{rv} D' \cap \mathsf{wv} D''$.

In the case of (TRANS) we have:

$$D \vdash x \prec w \prec z$$

Then either:

- $w \in \mathsf{wv} D'$ so by induction on Part 2 either:

  - $D' \vdash x \prec w$, and by induction we can find $y \in \mathsf{rv} D' \cap \mathsf{wv} D''$ such that $D' \vdash w \prec y$, so by (TRANS), $D' \vdash x \prec y$.
  - $\exists y \in \mathsf{rv} D' \cap \mathsf{wv} D'' \,.\, D' \vdash x \prec y$.
- $w \in \mathsf{wv} D''$ so by induction we can find $y \in \mathsf{rv} D' \cap \mathsf{wv} D''$ such that $D' \vdash x \prec y$.

The other cases are simpler.

2. Is similar. □

DEFINITION.

- x is tagged in $x := !M$.
- x is untagged in $x := ?M$.
- x is (un)tagged in D, E iff x is (un)tagged in D or E.
- x is (un)tagged in $\nu y \,.\, D$ iff $x \neq y$ and x is (un)tagged in D. □

PROPOSITION 38. *For closed D:*

1. *If $D \rightarrow_c (E', y := ?M) \equiv E$ then $D \equiv (D', y := ?M)$ and $D' \rightarrow_c E'$.*
2. *If $D \rightarrow_c E$, y is untagged in D and tagged in E then $D \equiv \nu \vec{x} \,.\, (F, y := ?M)$ and $E \equiv \nu \vec{x} \,.\, (F, y := !M)$.*
3. *If $D \rightarrow_x E$ then x is tagged in D.*
4. *If $D \rightarrow_c E \equiv (E', x := !M)$ and x is tagged in D then $D \equiv (D', x := !M)$ or $D \rightarrow_x E$.*
5. *If $D \rightarrow_c E \equiv \nu y \,.\, E'$ then $D \equiv \nu y \,.\, D'$ and $D' \rightarrow_c E'$ or $D \equiv \nu \vec{x} \,.\, (D', z := !\,\mathsf{rec} F \text{ in } M)$, $E \equiv \nu \vec{x} \,.\, (D', \mathsf{local} F \text{ in } z := !M)$, $E' \equiv \nu \vec{x} \,.\, (D', F')$ and $\nu y \,.\, F' \equiv \mathsf{local} F \text{ in } z := !M$.*
6. *If $D \equiv (D', x := !M) \rightarrow_c (E', x := !M) \equiv E$ then $D \rightarrow_x E$ or $\forall N \,.\, (D', x := !N) \rightarrow_c (E', x := !N)$.*
7. *If $\nu x \,.\, D' \equiv D \rightarrow_y E$ then $E \equiv \nu x \,.\, E'$ and $D' \rightarrow_y E'$.*
8. *If $D \equiv (D', D'') \rightarrow_x E$ then $E \equiv (E', D'')$ and $D' \rightarrow_x E'$ or $\exists y \in \mathsf{wv} D'' \,.\, D' \vdash x \prec y$.*

PROOF.

1. By Proposition 25.1 we have:

$$D \equiv \nu \vec{x} \,.\, (F, G) \qquad E \equiv \nu \vec{x} \,.\, (F, H) \qquad G \mapsto_c H \text{ is an axiom} \tag{22}$$

Then by Propositions 22.3 and 22.5 we have:

$$E' \equiv \nu \vec{x} \,.\, E'' \qquad (F, H) \equiv (E'', y := ?M) \tag{23}$$

Then by Propositions 22.2 and 22.1 either:

- we have:

$$F \equiv (F', y := ?M) \qquad (F', H) \equiv E'' \tag{24}$$

and so:

$$\begin{aligned} & D \\ & \equiv \nu\vec{x}.(F,G) && \text{(Eqn 22)} \\ & \equiv \nu\vec{x}.(F',y := ?M,G) && \text{(Eqn 24)} \\ & \equiv \nu\vec{x}.(F',G),y := ?M && (\nu\text{MIG}) \end{aligned}$$

and:

$$\begin{aligned} & \nu\vec{x}.(F',G) \\ & \quad \rightarrow_c \nu\vec{x}.(F',H) && \text{(Eqn 22)} \\ & \quad \equiv \nu\vec{x}.E'' && \text{(Eqn 24)} \\ & \quad \equiv E' && \text{(Eqn 23)} \end{aligned}$$

- or we have:

$$H \equiv (H', y := ?M) \qquad (F,H') \equiv E''$$

but by analysis of each axiom, there is no axiom $G \mapsto_c H$ where H contains an untagged node, and so we have a contradiction.

2. By Proposition 25.1 we have:

$$D \equiv \nu\vec{x}.(G,H) \qquad E \equiv \nu\vec{x}.(G,I) \qquad H \mapsto_c I \text{ is an axiom}$$

Then since y is untagged in D and tagged in E, this means y must be tagged in H and untagged in I, so the only axioms which could give $H \mapsto_c I$ are the axioms for spine traversal. Thus we can find J such that $H \equiv (J, y := ?M)$ and $I \equiv (J, y := !M)$. The result follows from setting F to be (G,J).

3. An induction on the proof of $D \rightarrow_x E$.

4. By Proposition 25.1:

$$D \equiv \nu\vec{x}.(F,G) \qquad E \equiv \nu\vec{x}.(F,H) \qquad G \mapsto_c H \text{ is an axiom} \qquad (25)$$

Then by Propositions 22.3 and 22.5:

$$E' \equiv \nu\vec{x}.E'' \qquad E'', x := !M \equiv F,H \qquad (26)$$

Then by Propositions 22.2 and 22.1, either:

- we have:

$$F \equiv (F', x := !M) \qquad E'' \equiv (F', H) \qquad (27)$$

so:

$$\begin{aligned} & D \\ & \equiv \nu\vec{x}.(F,G) && \text{(Eqn 25)} \\ & \equiv \nu\vec{x}.(F',x := !M,G) && \text{(Eqn 27)} \\ & \equiv \nu\vec{x}.(F',G),x := !M && (\nu\text{MIG}) \end{aligned}$$

- or we have:

$$H \equiv (H', x := !M) \qquad E'' \equiv (F,H') \qquad (28)$$

so by case analysis on which axiom could give $G \mapsto_c H$, either:

  - we have $G \rightarrow_x H$ and so:

$$\begin{aligned} & G \rightarrow_x H \\ & \quad \Rightarrow F,G \rightarrow_x F,H && \text{(Propn 33)} \\ & \quad \Rightarrow \nu\vec{x}.(F,G) \rightarrow_x \vec{x}.(F,H) && (\nu) \\ & \quad \Rightarrow D \rightarrow_x E && \text{(Eqn 25)} \end{aligned}$$

  - we have $G \equiv (G', x := !M)$ and so:

$$\begin{aligned} & G \equiv (G', x := !M) \\ & \quad \Rightarrow \nu\vec{x}.(F,G) \equiv \nu\vec{x}.(F,G', x := !M) && (\text{(L)}, \text{(R)} \text{ and } (\nu)) \\ & \quad \Rightarrow D \equiv \nu\vec{x}.(F,G', x := !M) && \text{(Eqn 25)} \\ & \quad \Rightarrow D \equiv \nu\vec{x}.(F,G'),x := !M && (\nu\text{MIG}) \end{aligned}$$

5. By Proposition 25.1:

$$D \equiv \nu\vec{w}.(G,H) \qquad E \equiv \nu\vec{w}.(G,I) \qquad H \mapsto_c I \text{ is an axiom} \qquad (29)$$

Then we can α-convert so that $y \notin \vec{y}$, and by Propositions 22.4 and 22.3 either:

- we have:

$$\vec{w} = \vec{y}w\vec{z} \qquad E' \equiv \nu\vec{y}\vec{z}.[y/w](G,I)[y/w] \qquad (30)$$

so:

$$\begin{aligned} & D \\ & \equiv \nu\vec{w}.(G,H) && \text{(Eqn 29)} \\ & \equiv \nu\vec{y}w\vec{z}.(G,H) && \text{(Eqn 30)} \\ & \equiv \nu\vec{y}y\vec{z}.[y/w](G,H)[y/w] && (\alpha) \\ & \equiv \nu y\vec{y}\vec{z}.[y/w](G,H)[y/w] && (\nu\text{SWAP}) \end{aligned}$$

and:

$$\begin{aligned} & \nu\vec{y}\vec{z}.[y/w](G,H)[y/w] \\ & \quad \rightarrow_c \nu\vec{y}\vec{z}.[y/w](G,I)[y/w] && \text{(Eqn 29 and Propn 24.1)} \\ & \quad \equiv E' && \text{(Eqn 30)} \end{aligned}$$

- or we have:

$$G \equiv \nu y.G' \qquad E' \equiv \nu\vec{w}.(G',I) \qquad (31)$$

so:

$$\begin{aligned} & D \\ & \equiv \nu\vec{w}.(G,H) && \text{(Eqn 29)} \\ & \equiv \nu\vec{w}.(\nu y.G',H) && \text{(Eqn 31)} \end{aligned}$$

$$\equiv \nu y\vec{w}.(G',H) \qquad ((\nu\text{MIG}) \text{ and } (\nu\text{SWAP}))$$

and:

$$\begin{aligned} &\nu\vec{w}.(G',H) \\ &\quad\to_c \nu\vec{w}.(G',I) && \text{(Eqn 29)} \\ &\quad\equiv E' && \text{(Eqn 31)} \end{aligned}$$

- or we have:

$$I \equiv \nu y.I' \qquad E' \equiv \nu\vec{w}.(G,I')$$

so by analysis of each axiom that could give $H \mapsto_c I$, we find that the only possibility is (BUILD) in which case:

$$\begin{aligned} D &\equiv \nu\vec{w}.(G, z := !\,\mathsf{rec}F \text{ in } M) \\ E &\equiv \nu\vec{w}.(G, \mathsf{local}F \text{ in } z := !M) \\ E' &\equiv \nu\vec{w}.(G,I') \\ \nu y.I' &\equiv \mathsf{local}F \text{ in } z := !M \end{aligned}$$

6. By Proposition 25.1:

$$D \equiv \nu\vec{y}.(G,H) \qquad E \equiv \nu\vec{y}.(G,I) \qquad H \mapsto_c I \text{ is an axiom} \qquad (32)$$

Then by Propositions 22.3 and 22.5 we have:

$$\begin{aligned} D' &\equiv \nu\vec{y}.D'' \\ (D'', x := !M) &\equiv (G,H) \\ E' &\equiv \nu\vec{y}.E'' \\ (E'', x := !M) &\equiv (G,I) \end{aligned} \qquad (33)$$

Then by Propositions 22.2 and 22.1 either:

- we have:

$$G \equiv (G', x := !M) \qquad D'' \equiv (G',H) \qquad E'' \equiv (G',I) \qquad (34)$$

so for any N:

$$\begin{aligned} &D', x := !N \\ &\quad\equiv (\nu\vec{y}.D''), x := !N && \text{(Eqn 33)} \\ &\quad\equiv (\nu\vec{y}.(G',H)), x := !N && \text{(Eqn 34)} \\ &\quad\mapsto_c (\nu\vec{y}.(G',I)), x := !N && \text{(Eqn 32)} \\ &\quad\equiv (\nu\vec{y}.E''), x := !N && \text{(Eqn 34)} \\ &\quad\equiv E', x := !N && \text{(Eqn 33)} \end{aligned}$$

- or we have:

$$H \equiv (H', x := !M) \quad D'' \equiv (G,H') \quad I \equiv (I', x := !M) \quad E'' \equiv (G,I')$$

and by case analysis of each axiom which could give $H \to_c I$, we find that either:

- $G,H \to_x G,I$ and so $D \to_x E$.
- For any N, $H', x := !N \to I', x := !N$, and so $D', x := !N \to E', x := !N$.

7. By Proposition 35:

$$D \equiv \nu\vec{x}.F \qquad E \equiv \nu\vec{x}.G \qquad F \vdash y \prec z \qquad F \to_z G \text{ is an axiom} \qquad (35)$$

and we can α-convert so that $x \notin \vec{x}$. Then by Proposition 22.4 either:

- we have:

$$\vec{x} = \vec{y}w\vec{z} \qquad D' \equiv \nu\vec{y}\vec{z}.[x/w]F[x/w] \qquad (36)$$

and so:

$$\begin{aligned} &E \\ &\quad\equiv \nu\vec{x}.G && \text{(Eqn 35)} \\ &\quad\equiv \nu\vec{y}w\vec{z}.G && \text{(Eqn 36)} \\ &\quad\equiv \nu x\vec{y}\vec{z}.[x/w]G[x/w] && ((\alpha) \text{ and } (\nu\text{SWAP})) \end{aligned}$$

by Proposition 24.1:

$$[x/w]F[x/w] \to_z [x/w]G[x/w]$$

by Proposition 34.3:

$$[x/w]F[x/w] \vdash y \prec z$$

and so:

$$\begin{aligned} &D' \\ &\quad\equiv \nu\vec{y}\vec{z}.[x/w]F[x/w] && \text{(Eqn 36)} \\ &\quad\to_y \nu\vec{y}\vec{z}.[x/w]G[x/w] && \text{(Propn 35)} \end{aligned}$$

- or we have:

$$F \equiv \nu x.F' \qquad D' \equiv \nu\vec{x}.F' \qquad (37)$$

so by analysis of each axiom:

$$G \equiv \nu x.G' \qquad F' \to_z G' \qquad (38)$$

so:

$$\begin{aligned} &E \\ &\quad\equiv \nu\vec{x}.G && \text{(Eqn 35)} \\ &\quad\equiv \nu\vec{x}x.G' && \text{(Eqn 38)} \\ &\quad\equiv \nu x\vec{x}.G' && (\nu\text{SWAP}) \end{aligned}$$

by Proposition 34.2:

$$F' \vdash y \prec z$$

and so:

$$\begin{array}{rll} & D' & \\ \equiv & \nu\vec{x}\,.\,F' & \text{(Eqn 37)} \\ \rightarrow_y & \nu\vec{x}\,.\,G' & \text{(Propn 35)} \end{array}$$

8. An induction on the proof of $D \rightarrow_x E$. The only difficult case is (ν), in which case:

$$D \equiv \nu z\,.\,F \qquad E \equiv \nu z\,.\,G \qquad F \rightarrow_x G$$

So by Proposition 22.3 either:

- $D' \equiv \nu z\,.\,F'$ and $F \equiv (F', D'')$ so by induction either:
  - $G \equiv (G', D'')$ and $F' \rightarrow_x G'$, so $D' \equiv \nu z\,.\,F' \rightarrow_x \nu z\,.\,G'$ and $E \equiv \nu z\,.\,G \equiv \nu z\,.\,(G', D'') \equiv (\nu z\,.\,G', D'')$.
  - $\exists y \in \mathsf{wv} D''\,.\,F' \vdash x \prec y$ so by (ν), $D' \vdash x \prec y$.
- $D'' \equiv \nu z\,.\,F''$ and $F \equiv (D', F'')$ so by induction either:
  - $G \equiv (G', F'')$ and $D' \rightarrow_x G'$ so $E \equiv \nu z\,.\,G \equiv \nu z\,.\,(G', F'') \equiv (G', D'')$.
  - $\exists y \in \mathsf{wv} F''\,.\,D' \vdash x \prec y$. Then either:
    - $x \in \mathsf{wv} D'$ so by Proposition 37.1 we can find a variable $w \in \mathsf{rv} D' \cap \mathsf{wv} F''$ such that $D' \vdash x \prec w$. Then since $w \in \mathsf{rv} D'$, $w \in \mathsf{rv} D$ and so $w \neq z$, and so by (ν), $D \vdash x \prec w$.
    - $x \in \mathsf{wv} D''$ and by (SYM), $D \vdash x \prec x$. □

PROPOSITION 39. *For closed D, if $D \rightarrow_c \rightarrow_x F$ and x is tagged in D, then we have $D \rightarrow_x \rightarrow_c F$.*

PROOF. Assume $D \rightarrow_c E \rightarrow_x F$. Then we proceed by induction on the proof of $E \rightarrow_x F$.

(BUILD) We have:

$$E = E', x := !\,\mathsf{rec} G \,\mathsf{in}\, M \qquad F = E', \mathsf{local}\, G \,\mathsf{in}\, x := !M \tag{39}$$

We can α-convert G so that $\mathsf{wv} G \cap \mathsf{fv} E' = \emptyset$. Then by Proposition 38.4 either:

- we have:

$$D \equiv (D', x := !\,\mathsf{rec} G \,\mathsf{in}\, M) \tag{40}$$

so by Proposition 38.6 either:

- we have $D \rightarrow_x E$, and so $D \rightarrow_x \rightarrow_c F$.

- or we have:

$$\forall N\,.\,(D', x := !N) \rightarrow_c (E', x := !N) \tag{41}$$

and so:

$$\begin{array}{rll} & D & \\ \equiv & D', x := !\,\mathsf{rec} G \,\mathsf{in}\, M & \text{(Eqn 40)} \\ \rightarrow_x & D', \mathsf{local}\, G \,\mathsf{in}\, x := !M & \text{(BUILD)} \\ \equiv & \mathsf{local}\, G \,\mathsf{in}\, (D', x := !M) & (\nu\text{MIG}) \\ \rightarrow_c & \mathsf{local}\, G \,\mathsf{in}\, (E', x := !M) & \text{(Eqn 41)} \\ \equiv & E', \mathsf{local}\, G \,\mathsf{in}\, x := !M & (\nu\text{MIG}) \\ \equiv & F & \text{(Eqn 39)} \end{array}$$

- $D \rightarrow_x E$ and so $D \rightarrow_x \rightarrow_c F$.

(∇TRAV) We have:

$$E = E', x := !\nabla y, y := ?M \qquad F = E', x := !\nabla y, y := !M \tag{42}$$

By Proposition 38.1:

$$D \equiv D', y := ?M \qquad D' \rightarrow_c E', x := !\nabla y \tag{43}$$

Then by Proposition 38.4 either:

- we have:

$$D' \equiv D'', x := !\nabla y \tag{44}$$

and so:

$$\begin{array}{rll} & D & \\ \equiv & D', y := ?M & \text{(Eqn 43)} \\ \equiv & D'', x := !\nabla y, y := ?M & \text{(Eqn 44)} \\ \rightarrow_x & D'', x := !\nabla y, y := !M & (\nabla\text{TRAV}) \\ \equiv & D', y := !M & \text{(Eqn 44)} \\ \rightarrow_c & E', x := !\nabla y, y := !M & \text{(Eqn 43)} \\ \equiv & F & \text{(Eqn 42)} \end{array}$$

- or we have $D' \rightarrow_x E', x := !\nabla y$, so by Proposition 33 $D \rightarrow_x E$, and so $D \rightarrow_x \rightarrow_c F$.

$(@\text{TRAV})$ Is similar.

$(\vee\text{TRAV})$ Is similar.

(∇UPD) We have:

$$E = E', x := !\nabla y, y := !\lambda w.\,M \qquad F = E', x := !\lambda w.\,M, y := !\lambda w.\,M \tag{45}$$

Then by Proposition 38.4 either:

- $D \equiv D', x := !\nabla y$, so by Proposition 38.4 either:
  - $D \equiv D'', y := !\lambda w.\, M$, so by Propositions 22.2 and 22.1:

    $$D \equiv D''', x := !\nabla y, y := !\lambda w.\, M$$

    Then by Proposition 38.6 either:
    - we have $D \to_x E$, and so $D \to_x \to_c F$.
    - or we have:

      $$\forall N.\, (D''', x := !N, y := !\lambda w.\, M) \to_c (E', x := !N, y := !\lambda w.\, M) \tag{46}$$

      and so:

      $$\begin{array}{rll} & D & \\ \equiv & (D''', x := !\nabla y, y := !\lambda w.\, M) & \text{(Eqn 46)} \\ \to_x & (D''', x := !\lambda w.\, M, y := !\lambda w.\, M) & (\nabla\text{UPD}) \\ \to_c & (E', x := !\lambda w.\, M, y := !\lambda w.\, M) & \text{(Eqn 46)} \\ \equiv & F & \text{(Eqn 45)} \end{array}$$

  - $D \to_y E$, so by ∇_{IND} $D \to_x E$, and so $D \to_x \to_c F$.
- $D \to_x E$, and so $D \to_x \to_c F$.

(@UPD) Is similar.

($\vee$UPDa) Is similar.

($\vee$UPDb) Is similar.

($\vee$UPDc) Is similar.

($\equiv$) We have $E \equiv E' \to_x F' \equiv F$ so $D \to_c E' \to_x F'$, and so by induction $D \to_x \to_c F' \equiv F$.

(ν) We have:

$$E = \nu y.\, E' \qquad F = \nu y.\, F' \qquad E' \to_x F' \tag{47}$$

Then by Proposition 38.5 either:

- we have $D \equiv \nu y.\, D'$ and $D' \to_c E'$. Then x is tagged in D', so by induction $D' \to_x \to_c F'$, and so $D \to_x \to_c F$.
- or we have:

$$\begin{aligned} D &\equiv \nu \vec{x}.\, (D', z := !\,\mathsf{rec}\, G \,\mathsf{in}\, M) \\ E &\equiv \nu \vec{x}.\, (D', \mathsf{local}\, G \,\mathsf{in}\, z := !M) \\ E' &\equiv \nu \vec{x}.\, (D', G') \\ \nu y.\, G' &\equiv \mathsf{local}\, G \,\mathsf{in}\, z := !M \end{aligned} \tag{48}$$

Then:

$$\begin{array}{ll} & E' \to_x F' \\ \Rightarrow & \nu \vec{x}.\, (D', G') \to_x F' \qquad \text{(Eqn 47)} \\ \Rightarrow & \nu y \vec{x}.\, (D', G') \to_x \nu y.\, F' \qquad (\nu) \\ \Rightarrow & \nu \vec{x}.\, (D', \nu y.\, G') \to_x \nu y.\, F' \qquad (\nu\text{MIG}) \end{array}$$

Then by Proposition 38.7:

$$\nu y.\, F' \equiv \nu \vec{x}.\, F'' \qquad (D', \nu y.\, G') \to_x F'' \tag{49}$$

so by Proposition 38.8 either:

- we have:

  $$F'' \equiv (F''', \nu y.\, G') \qquad D' \to_x F''' \tag{50}$$

  Then:

  $$\begin{array}{rll} & D & \\ \equiv & \nu \vec{x}.\, (D', z := !\,\mathsf{rec}\, G \,\mathsf{in}\, M) & \text{(Eqn 48)} \\ \to_x & \nu \vec{x}.\, (F''', z := !\,\mathsf{rec}\, G \,\mathsf{in}\, M) & \text{(Eqn 50)} \\ \to_x & \nu \vec{x}.\, (F''', \mathsf{local}\, G \,\mathsf{in}\, z := !M) & \text{(BUILD)} \\ \equiv & \nu \vec{x}.\, (F''', \nu y.\, G') & \text{(Eqn 48)} \\ \equiv & \nu \vec{x}.\, F'' & \text{(Eqn 50)} \\ \equiv & \nu y.\, F' & \text{(Eqn 49)} \\ \equiv & F & \text{(Eqn 47)} \end{array}$$

- or we can find $w \in \mathsf{wv}(\nu y.\, G')$ such that $D' \vdash x \prec w$, and since $\mathsf{wv}(\nu y.\, G') = \{z\}$, this means $w = z$, so $D' \vdash x \prec z$. Then:

  $$\begin{array}{ll} & \mathsf{true} \\ \Rightarrow & (D', z := !\,\mathsf{rec}\, G \,\mathsf{in}\, M) \to_z (D', \mathsf{local}\, G \,\mathsf{in}\, z := !M) \qquad \text{(BUILD)} \\ \Rightarrow & (D', z := !\,\mathsf{rec}\, G \,\mathsf{in}\, M) \to_x (D', \mathsf{local}\, G \,\mathsf{in}\, z := !M) \qquad \text{(Propn 36)} \\ \Rightarrow & \nu \vec{x}.\, (D', z := !\,\mathsf{rec}\, G \,\mathsf{in}\, M) \to_x \nu \vec{x}.\, (D', \mathsf{local}\, G \,\mathsf{in}\, z := !M) \qquad (\nu) \\ \Rightarrow & D \to_x E \qquad \text{(Eqn 48)} \end{array}$$

(∇_{IND}) We have:

$$E = E', x := !\nabla y \qquad E \to_y F$$

Then by Proposition 38.4 either:

- $D \equiv D', x := !\nabla y$, so either:
  - y is tagged in D, so by induction $D \to_y \to_c F$, and so by (∇_{IND}) $D \to_x \to_c F$.
  - y is untagged in D, so by Propositions 22.2, 22.1 and 38.2:

    $$D \equiv (D'', x := !\nabla y, y := ?M) \qquad E \equiv (D'', x := !\nabla y, y := !M)$$

so by (∇IND), $D \to_x E$, and so $D \to_x \to_c F$.

- $D \to_x E$, and so $D \to_x \to_c F$.

(@IND) Is similar.

($\vee$IND) Is similar. □

PROPOSITION 40. *For closed D, if x is tagged in D and $D \to_c^* E$ then $D \to_x^* \to_{\neg x}^* E$.*

PROOF. Let $D \to_c^n E$, and proceed by induction on n.

- If $n = 0$ then $D \equiv E$ so $D \to_x^* \to_{\neg x}^* E$.
- If $n > 0$ then $D \to_c F \to_c^{n-1} E$, and by Proposition 23.4 x is tagged in F so by induction $F \to_x^* \to_{\neg x}^* E$, so by Proposition 39 $D \to_x^* \to_c \to_{\neg x}^* E$, and so $D \to_x^* \to_{\neg x}^* E$. □

PROPOSITION 41. *For closed D, if x is tagged in D, $D \to_{\neg x} E$ and x is in whnf in E then x is in whnf in D.*

PROOF. By Proposition 25.1 we have:

$$D \equiv \nu\vec{x}.(F,G) \qquad E \equiv \nu\vec{x}.(F,H) \qquad G \mapsto_c H \text{ is an axiom}$$

Then by the definition of whnf, $x \notin \vec{x}$ and either:

- x is in whnf in F, so x is in whnf in D.
- x is in whnf in H, and by inspection of the axioms which could result in $G \mapsto_c H$ we have either:
  - $G \to_x H$ and so $D \to_x E$ which is a contradiction.
  - x is in whnf in G and so x is in whnf in D. □

PROPOSITION 42. *For closed D, $D{\Downarrow_x}$ iff $\mathsf{tag}_x D \to_x^* E$ and x is in whnf in E.*

PROOF.

$\Rightarrow$ We have $\mathsf{tag}_x D \to_c^* F$ and x is in whnf in F. By Proposition 40 we know that $\mathsf{tag}_x D \to_x^* E \to_{\neg x}^* F$. By Proposition 23.4 x is tagged in E, so by Proposition 41 x is in whnf in E.

$\Leftarrow$ We have $\mathsf{tag}_x D \to_x^* E$ and so $\mathsf{tag}_x D \to^* E$. Thus $D{\Downarrow_x}$. □

PROPOSITION 43. *If $\mathsf{tag}_y D' = D \to_x E$ and $x \neq y$ then $E = \mathsf{tag}_y E'$ and $D' \to_x^* E'$.*

PROOF. An induction on the proof of $D \to_x E$. □

PROPOSITION 44. *$D{\Downarrow_x}$ iff $\mathsf{tag}_y D{\Downarrow_x}$.*

PROOF.

$\Rightarrow$ We have:

$$\begin{array}{ll}
D{\Downarrow_x} & \\
\Rightarrow \mathsf{tag}_x D \to^* E, x \text{ is in whnf in } E & \text{(Defn of } {\Downarrow_x}) \\
\Rightarrow \mathsf{tag}_y(\mathsf{tag}_x D) \to^* \mathsf{tag}_y E, x \text{ is in whnf in } E & \text{(Propn 23.4)} \\
\Rightarrow \mathsf{tag}_y(\mathsf{tag}_x D) \to^* \mathsf{tag}_y E, x \text{ is in whnf in } \mathsf{tag}_y E & \text{(Propn 23.5)} \\
\Rightarrow \mathsf{tag}_x(\mathsf{tag}_y D) \to^* \mathsf{tag}_y E, x \text{ is in whnf in } \mathsf{tag}_y E & \text{(Propn 23.1)} \\
\Rightarrow \mathsf{tag}_y D{\Downarrow_x} & \text{(Defn of } {\Downarrow_x})
\end{array}$$

$\Leftarrow$ If $x = y$ then:

$$\begin{array}{ll}
\mathsf{tag}_y D{\Downarrow_x} & \\
\Rightarrow \mathsf{tag}_x D{\Downarrow_x} & (x = y) \\
\Rightarrow \mathsf{tag}_x(\mathsf{tag}_x D) \to^* E, x \text{ is in whnf in } E & \text{(Defn of } {\Downarrow_x}) \\
\Rightarrow \mathsf{tag}_x D \to^* E, x \text{ is in whnf in } E & \text{(Propn 23.2)} \\
\Rightarrow D{\Downarrow_x} & \text{(Defn of } {\Downarrow_x})
\end{array}$$

If $x \neq y$ then:

$$\begin{array}{ll}
\mathsf{tag}_y D{\Downarrow_x} & \\
\Rightarrow \mathsf{tag}_x(\mathsf{tag}_y D) \to_x^* E, x \text{ is in whnf in } E & \text{(Propn 42)} \\
\Rightarrow \mathsf{tag}_y(\mathsf{tag}_x D) \to_x^* E, x \text{ is in whnf in } E & \text{(Propn 23.1)} \\
\Rightarrow \mathsf{tag}_x D \to_x^* F, E \equiv \mathsf{tag}_y F, x \text{ is in whnf in } E & \text{(Propn 43)} \\
\Rightarrow \mathsf{tag}_x D \to_x^* F, E \equiv \mathsf{tag}_y F, x \text{ is in whnf in } F & \text{(Propn 23.6)} \\
\Rightarrow D{\Downarrow_x} & \text{(Propn 42)}
\end{array}$$

Thus $\mathsf{tag}_y D{\Downarrow_x}$ iff $D{\Downarrow_x}$. □

COROLLARY 45. *If $D \leq_? E$ then $D{\Downarrow_x}$ iff $E{\Downarrow_x}$.*

PROPOSITION 46. *If $D \leftarrow_\gamma \to_x E$ then $D \to_x \leftarrow_\gamma E$.*

PROOF. By Proposition 25.1 we can find F and G such that:

$$D \equiv \nu\vec{x}.F \qquad \nu\vec{x}.(F, \nu(\mathsf{w}\nu G).G) \to_x E \tag{51}$$

Then by Proposition 38.7 we can find H such that:

$$E \equiv \nu\vec{x}.H \qquad (F, \nu(\mathsf{w}\nu G).G) \to H \tag{52}$$

Then by Proposition 38.8 we can find I such that:

$$H \equiv (I, \nu(\mathsf{w}\nu G).G) \qquad F \to I \tag{53}$$

Thus:

$$\begin{array}{ll}
D & \\
\equiv \nu\vec{x}.F & \text{(Eqn 51)} \\
\to_x \nu\vec{x}.I & \text{(Eqn 53)}
\end{array}$$

$$\begin{array}{rll} & \leftarrow_\gamma \nu\vec{x}.(I, \nu(\mathsf{wv}\,G).G) & (\gamma) \\ \equiv & \nu\vec{x}.H & \text{(Eqn 53)} \\ \equiv & E & \text{(Eqn 53)} \end{array}$$

Thus $D \rightarrow_x \leftarrow_\gamma E$. □

PROPOSITION 47. *For closed D, if $D \rightarrow E$ then $D\Downarrow_x$ iff $E\Downarrow_x$.*

PROOF.

$\Rightarrow$ If $D \rightarrow_c E$, then by Proposition 32, $E\Downarrow_x$.

If $D \rightarrow_\gamma E$, then by Propositions 42 and 23.4:

$$\begin{array}{c} \mathsf{tag}_x D \rightarrow_x \cdots \rightarrow_x F \\ \downarrow^{\leq 1}_{\overline{\gamma}} \\ \mathsf{tag}_x E \end{array}$$

and x is in whnf in F, so by Proposition 46:

$$\begin{array}{ccccc} \mathsf{tag}_x D & \rightarrow_x & \cdots & \rightarrow_x & F \\ \downarrow^{\leq 1}_{\overline{\gamma}} & \downarrow^{\leq 1}_{\overline{\gamma}} & \downarrow^{\leq 1}_{\overline{\gamma}} & & \downarrow^{\leq 1}_{\overline{\gamma}} \\ \mathsf{tag}_x E & \rightarrow_x & \cdots & \rightarrow_x & G \end{array}$$

and by Proposition 28.3 x is in whnf in G, and so $E\Downarrow_x$.

Otherwise $D \rightarrow E$ from ($\vee$UPD) and since $D \not\rightarrow_c E$ and D is closed:

$$D \equiv \nu\vec{x}.(F, w := !y \vee z, y := !\lambda w.M, z := ?N) \quad (54)$$

$$E \equiv \nu\vec{x}.(F, w := !\mathsf{I}, y := !\lambda w.M, z := ?N) \quad (55)$$

and so:

$$\begin{array}{rll} D & & \\ \equiv & \nu\vec{x}.(F, w := !y \vee z, y := !\lambda w.M, z := ?N) & \text{(Eqn 54)} \\ \geq_? & \nu\vec{x}.(F, w := !y \vee z, y := !\lambda w.M, z := !N) & (\text{Defn of } \leq_?) \\ \rightarrow_c & \nu\vec{x}.(F, w := !\mathsf{I}, y := !\lambda w.M, z := !N) & (\vee\text{UPD}a) \\ \leq_? & \nu\vec{x}.(F, w := !\mathsf{I}, y := !\lambda w.M, z := ?N) & (\text{Defn of } \leq_?) \\ \equiv & E & \text{(Eqn 55)} \end{array}$$

Thus by Proposition 32 and Corollary 45, $E\Downarrow_x$.

$\Leftarrow$ Follows from the definition of $D \rightarrow_x$. □

For example, we can use this to show that extending a closed declaration does not affect its convergence.

PROPOSITION 48. *If $D \sqsubseteq E$, $x \in \mathsf{wv}\,D$, and D is closed, then $D\Downarrow_x$ iff $E\Downarrow_x$.*

PROOF. We can show by induction on the proof of $\sqsubseteq$ that:

$$D \equiv \nu\vec{x}.F \qquad E \equiv \nu\vec{x}.(F, G) \qquad \mathsf{wv}\,G \cap \mathsf{fv}\,F = \emptyset$$

Then:

$\Rightarrow$ If $D\Downarrow_x$ then since $\rightarrow_c$ is convergent, $\mathsf{tag}_x(\nu\vec{x}.F) \rightarrow_c^* H$ and x is in whnf in H, so by Proposition 26 $\mathsf{tag}_x F \rightarrow_c^* I$ and $H \equiv \nu\vec{x}.I$, so x is in whnf in I. Then $\mathsf{tag}_x(\nu\vec{x}.(F, G)) \rightarrow_c^* \nu\vec{x}.(I, G)$, and so $E\Downarrow_x$.

$\Leftarrow$ If $E\Downarrow_x$ then by Proposition 29, $\nu(\mathsf{wv}\,G).E\Downarrow_x$, so $\nu(\mathsf{wv}\,G).\nu\vec{x}.(F, G)\Downarrow_x$, so by ($\nu$MIG), $\nu\vec{x}.(F, \nu(\mathsf{wv}\,G).G)\Downarrow_x$, so by Proposition 47 $\nu\vec{x}.F\Downarrow_x$, so $D\Downarrow_x$. □

3.8 Operational properties: referential transparency

Referential transparency was introduced by EVANS (1968) to mean that the semantics of a term should be the same as the semantics of a pointer to a term. In our semantics this is the same as saying:

$$[\![x := !\nabla y, y := !M]\!] = [\![x := !M, y := !M]\!]$$

Denotationally, this is quite simple to prove (although it does require some non-trivial reasoning about fixed points). But to prove this operationally is much harder. We need to show that copying a section of graph is equivalent to making a pointer into a section of graph. Much of the work in showing this turns out to be in showing that if two variables point to the same term, then we can substitute one for the other, that is:

$$[\![(D, x := !M, y := !M)[x/z]]\!] = [\![(D, x := !M, y := !M)[y/z]]\!]$$

In order to prove this operationally, we need to find some property of a declaration $(D, x := !M, y := M)$ which we can use as an operational invariant, so:

- If D satisfies the invariant and $D[x/z] \rightarrow_c E$ then we can find an F such that $E \rightarrow_c^* F[x/z]$, and $D[y/z] \rightarrow_c^* F[y/z]$, and F satisfies the invariant.

We can then use this to show that if $D[x/z]\Downarrow_w$ then $D[y/z]\Downarrow_w$. Unfortunately, we cannot use 'x and y point to syntactically identical terms' as the invariant, since:

$$\begin{array}{l} x := !(\mathsf{rec}\,w := !M \,\mathsf{in}\, \nabla w), y := !(\mathsf{rec}\,w := !M \,\mathsf{in}\, \nabla w) \\ \rightarrow^2 \nu v w.(v := !M[v/w], w := !M, x := !\nabla v, y := !\nabla w) \end{array}$$

and although x and y are syntactically identical in the LHS, they are not syntactically identical in the RHS. However, they *are* identical up to α-conversion, and we can use this as the basis of an invariant: *simulation*, based on MILNER's (1989) definition of bisimulation between processes. Informally, two variables x and y are similar iff x points to M, y points to N, and M and N are identical, up to substitution of similar variables. More formally, we can define a simulation for ν-less declarations as:

DEFINITION. $\mathcal{R} \subseteq \mathsf{wv}\,D \times \mathsf{wv}\,D$ is a ν-less D-simulation iff D is ν-less, and for any $x\,\mathcal{R}\,y$:

- If $D \sqsupseteq (x := !M)$ then $D \sqsupseteq (y := !N[\vec{y}/\vec{z}])$, $M = N[\vec{x}/\vec{z}]$, and $\vec{x}\,\mathcal{R}\,\vec{y}$.

- If $D \sqsupseteq (x := ?M)$ then $D \sqsupseteq (y := ?N[\vec{y}/\vec{z}])$, $M = N[\vec{x}/\vec{z}]$, and $\vec{x}\ \mathcal{R}\ \vec{y}$.

where $\vec{x}\ \mathcal{R}\ \vec{y}$ iff $\forall i \,.\, x_i\ \mathcal{R}\ y_i$. □

For example, if E is ν-less, and D is the declaration:

$$x := !M, y := !M, E$$

then one ν-less D-simulation is:

$$\{(x,y)\}$$

and so x is D-similar to y. If D is the declaration:

$$w := !y@z, x := !z@z, y := !\mathsf{I}, z := \mathsf{I}$$

then one ν-less D-simulation is:

$$\{(w,x),(y,z),(z,z)\}$$

and so w is D-similar to x. If D is:

$$x := !\nabla x, y := !\nabla z, z := !\nabla y$$

then one ν-less D-simulation is:

$$\{(x,y),(x,z)\}$$

and so x is D-similar to y. We can generalize simulation to any declaration D by converting it into the form $\nu\vec{x} \,.\, E$, and finding a ν-less E-simulation:

DEFINITION.

- $\nu x \,.\, \mathcal{R} = \{(y,z) \mid x \neq y\ \mathcal{R}\ z \neq x\}$.
- $\mathcal{R}$ is a D-simulation iff $D \equiv \nu\vec{x}.E$, $\mathcal{R}'$ is a ν-less E-simulation, and $\mathcal{R} = \nu\vec{x} \,.\, \mathcal{R}'$.
- $D \vdash x \sim y$ iff there is a D-simulation $\mathcal{R}$ with $x\ \mathcal{R}\ y$. □

For example, for any E, if D is the declaration:

$$x := !M, y := !M, E$$

then we can find a ν-less F such that $E \equiv \nu\vec{z} \,.\, F$, and we can α-convert E so $D \equiv \nu\vec{z} \,.\, (x := !M, y := !M, F)$, and $\{(x,y)\}$ is a ν-less $(x := !M, y := !M, F)$-simulation, so $\{(x,y)\}$ is a D-simulation, and so:

$$(x := !M, y := !M, E) \vdash x \sim y$$

The rest of this section shows that:

- If $D[x/z] \vdash x \sim y$ then $D[x/z]{\Downarrow_w}$ iff $D[y/z]{\Downarrow_w}$.
- If $D \vdash x \sim y$ then $D{\Downarrow_x}$ iff $D{\Downarrow_y}$.
- We can use this to show referential transparency.

DEFINITION. Define 'D is (un)tagged' as:

- ε is tagged.
- $x := !M$ is tagged.
- $x := ?M$ is untagged.
- D, E is (un)tagged iff D and E are (un)tagged.
- $\nu x \,.\, D$ is (un)tagged iff D is (un)tagged.

Define $\mathsf{tag}\, D$ as:

- $\mathsf{tag}\, \varepsilon = \varepsilon$
- $\mathsf{tag}(x := !M) = (x := !M)$
- $\mathsf{tag}(x := ?M) = (x := !M)$
- $\mathsf{tag}(D, E) = \mathsf{tag}\, D, \mathsf{tag}\, E$
- $\mathsf{tag}(\nu x \,.\, D) = \nu x \,.\, (\mathsf{tag}\, D)$ □

PROPOSITION 49.

1. *If* $D \vdash \vec{x} \sim \vec{y}$ *then* $\mathsf{tag}\, D \vdash \vec{x} \sim \vec{y}$.
2. *If* $\nu\vec{w} \,.\, D \vdash \vec{x} \sim \vec{y}$ *then* $D \vdash \vec{x} \sim \vec{y}$.

PROOF.

1. From the definition, if $\mathcal{R}$ is a ν-less D-simulation, then $\mathcal{R}$ is a ν-less $\mathsf{tag}\, D$-simulation. Thus, if $\mathcal{R}$ is a D-simulation, then $\mathcal{R}$ is a $\mathsf{tag}\, D$-simulation. Thus, if $D \vdash \vec{x} \sim \vec{y}$ then $\mathsf{tag}\, D \vdash \vec{x} \sim \vec{y}$.
2. If $\nu\vec{w} \,.\, D \vdash \vec{x} \sim \vec{y}$ then let $D \equiv \nu\vec{v} \,.\, E$, and $\mathcal{R}$ be a ν-less E-simulation such that $\vec{x}\ \mathcal{R}\ \vec{y}$. Then $\nu\vec{v} \,.\, \mathcal{R}$ is a D-simulation, and so $D \vdash \vec{x}\ \mathcal{R}\ \vec{y}$. □

PROPOSITION 50. *If D is ν-less, closed and tagged, $D[\vec{x}/\vec{z}] \vdash \vec{x} \sim \vec{y}$, and $D[\vec{x}/\vec{z}] \to_c E$, then $E \to_c^* F[\vec{x}/\vec{z}]$, $F[\vec{x}/\vec{z}] \vdash \vec{x} \sim \vec{y}$, and $D[\vec{y}/\vec{z}] \to_c^* F[\vec{y}/\vec{z}]$.*

PROOF. We proceed by analysis on which axiom gave $D[\vec{x}/\vec{z}] \to_c E$. We shall prove the case for (BUILD), since the others are simpler. Since D is ν-less and tagged, we have:

$$D \equiv ((x := !\,\mathsf{rec}\, G \,\mathsf{in}\, M), x_1 := !M_1, \ldots, x_n := !M_n)$$
$$E \equiv ((\mathsf{local}\, G \,\mathsf{in}\, x := !M), x_1 := !M_1, \ldots, x_n := !M_n)[\vec{x}/\vec{z}]$$

and we can α-convert G so that $\mathsf{wv}\, G \cap \mathsf{fv}\, D = \emptyset$. Then let $\vec{w} = \mathsf{wv}\, G$, and define $F \equiv \nu\vec{w}\vec{w}_1 \ldots \vec{w}_n \,.\, (G, x := !M, F_1, \ldots, F_n)$ as:

- If $D[\vec{x}/\vec{z}] \vdash x \sim x_i$ then we can find M_i', $\vec{x}_i$, $\vec{y}_i$ and $\vec{z}_i$ such that:

  $$\mathsf{rec}\, G \,\mathsf{in}\, M[\vec{x}/\vec{z}] = M_i'[\vec{x}_i/\vec{z}_i] \qquad M_i[\vec{x}/\vec{z}] = M_i'[\vec{y}_i/\vec{z}_i] \qquad \vec{x}_i\ \mathcal{R}\ \vec{y}_i$$

  Thus we can find G_i and N_i such that:

  $$(\mathsf{rec}\, G_i \,\mathsf{in}\, N_i)[\vec{x}_i/\vec{z}_i] = \mathsf{rec}\, G \,\mathsf{in}\, M \qquad (\mathsf{rec}\, G_i \,\mathsf{in}\, N_i)[\vec{y}_i/\vec{z}_i] = M_i$$

  Let $\vec{w}_i$ be fresh, and define:

  $$F_i \equiv ([\vec{w}_i/\vec{w}]G_i[\vec{w}_i/\vec{w}], x := !N_i[\vec{w}_i/\vec{w}])$$

- Otherwise $F_i = (x_i := !M_i)$, and $w_i = \varepsilon$.

Then:

- For each i such that $D[\vec{x}/\vec{z}] \vdash x \sim x_i$, $(x_i := !M_i[\vec{x}/\vec{z}]) \to_c \nu\vec{w}_i\,.\,F_i[\vec{x}/\vec{z}]$ and so $E \to_c^* F[\vec{x}/\vec{z}]$.
- Similarly, $D[\vec{y}/\vec{z}] \to_c^* F[\vec{y}/\vec{z}]$.
- Let $\mathcal{R}$ be a ν-less $D[\vec{x}/\vec{z}]$-simulation such that $\vec{x}\,\mathcal{R}\,\vec{y}$. Then let $\mathcal{R}'$ be the smallest relation containing $\mathcal{R}$ such that $\vec{w}\vec{w}\vec{w}_i\vec{w}_i\,\mathcal{R}\,\vec{w}\vec{w}_j\vec{w}_j\vec{w}$. We can show $\mathcal{R}'$ is a ν-less $(G, x := !M, F_1, \ldots, F_n)[\vec{x}/\vec{z}]$-simulation, and so $F[\vec{x}/\vec{z}] \vdash \vec{x} \sim \vec{y}$. □

PROPOSITION 51. *For closed D:*

1. *If $D[\vec{x}/\vec{z}] \vdash \vec{x} \sim \vec{y}$ and $D[\vec{x}/\vec{z}]\Downarrow_x$ then $D[\vec{y}/\vec{z}]\Downarrow_x$.*
2. *If $D \vdash x \sim y$ then $D\Downarrow_x$ iff $D\Downarrow_y$.*

PROOF.

1. By Proposition 28.4 $\mathsf{tag}_x D[\vec{x}/\vec{z}] \to_c^n E$ and x is in whnf in E. We now show by induction on n that if $D[\vec{x}/\vec{z}] \to_c^n E$, x is in whnf in E, and $D[\vec{x}/\vec{z}] \vdash \vec{x} \sim \vec{y}$ then $D[\vec{y}/\vec{z}]\Downarrow_x$.

   By Proposition 20 we can find ν-less D' such that $\mathsf{tag}\,D \equiv \nu\vec{w}\,.\,D'$, and by Propositions 27.4 and 26, $D'[\vec{x}/\vec{z}] \to_c^m E'$, $m \le n$, and x is in whnf in E'. Then:

   - If $m = 0$ then x is in whnf in $D'[\vec{x}/\vec{z}]$, so x is in whnf in $D[\vec{y}/\vec{z}]$, so $D[\vec{y}/\vec{z}]\Downarrow_x$.
   - If $m > 0$ then $D'[\vec{x}/\vec{z}] \to_c F \to_c^{m-1} E'$ so by Proposition 50 $F \to_c^* F'[\vec{x}/\vec{z}]$, $F'[\vec{x}/\vec{z}] \vdash \vec{x} \sim \vec{y}$ and $D'[\vec{y}/\vec{z}] \to_c^* F'[\vec{y}/\vec{z}]$. Thus we have:

$$\begin{array}{lccc} D'[\vec{x}/\vec{z}] \to_c & F & \to_c^{m-1} & E' \\ & \downarrow_c^* & & \\ & F'[\vec{x}/\vec{z}] & & \end{array}$$

   so by confluence:

$$\begin{array}{lccc} D'[\vec{x}/\vec{z}] \to_c & F & \to_c^{m-1} & E' \\ & \downarrow_c^* & & \downarrow_c^* \\ & F'[\vec{x}/\vec{z}] & \to_c^{\le m-1} & E'' \end{array}$$

   Since x is in whnf in E', x is in whnf in E'', so by induction $D'[\vec{y}/\vec{z}]\Downarrow_x$, so by ($\nu$), $\nu\vec{w}\,.\,D'[\vec{y}/\vec{z}]\Downarrow_x$, and so by Corollary 45, $D[\vec{y}/\vec{z}]\Downarrow_x$.

2. Is similar. □

We can use this to show referential transparency:

PROPOSITION 52. *For closed declarations:*

1. $(D, x := !\nabla y)\Downarrow_x$ *iff* $(D, x := !\nabla y)\Downarrow_y$.
2. $(D, x := !M, y := !M)\Downarrow_z$ *iff* $(D, x := !\nabla y, y := !M)\Downarrow_z$.

PROOF.

1. $\Rightarrow$ Show by induction on n that if $\nu\vec{x}\,.\,(D, x := !\nabla y) \to_c^n E$ and x is in whnf in E, then $\nu\vec{x}\,.\,(D, x := !\nabla y)\Downarrow_y$.

   If $n = 0$ then we have a contradiction.

   If $n > 0$ then we have $\nu\vec{x}\,.\,(D, x := !\nabla y) \to_c F \to_c^{n-1} E$, and so by Proposition 30.2 either:

$$F \equiv \nu\vec{y}\,.\,(D', x := !\nabla y)$$

   and so by induction $\nu\vec{x}\,.\,(D, x := !\nabla y)\Downarrow_y$, or:

$$F \equiv \nu\vec{x}\,.\,(D, x := !M, y := !M) \qquad M \equiv \lambda w\,.\,N$$

   and so $\nu\vec{x}\,.\,(D, x := !\nabla y)\Downarrow_y$.

   $\Leftarrow$ Is similar.

2. $\Rightarrow$ Show by induction on n that if:

$$\nu\vec{x}\,.\,(D, x := !M, y := !N) \vdash x \sim y$$
$$\nu\vec{x}\,.\,(D, x := !M, y := !N) \to_c^n E$$
$$z \text{ is in whnf in } E$$

   then $\nu\vec{x}\,.\,(D, x := !\nabla y, y := !N)\Downarrow_z$.

   If $n = 0$ then z is in whnf in $\nu\vec{x}\,.\,(D, x := !M, y := !N)$ and so:

$$\nu\vec{x}\,.\,(D, x := !\nabla y, y := !N)\Downarrow_z$$

   If $n > 0$ then we have:

$$\nu\vec{x}\,.\,(D, x := !M, y := !N) \to_c F \to_c^{<n} E$$

   We proceed by case analysis of the axiom for $\mapsto$ used. Most of the cases are similar, so we shall prove the case for (BUILD) when:

$$\begin{aligned} M &= \mathsf{rec}\,G\,\mathsf{in}\,M' \\ N &= \mathsf{rec}\,H\,\mathsf{in}\,N' \\ F &\equiv \nu\vec{x}\,.\,(D, x := !\,\mathsf{rec}\,G\,\mathsf{in}\,M', \mathsf{local}\,H\,\mathsf{in}\,y := !N') \end{aligned} \tag{56}$$

   We shall α-convert so that $\mathsf{wv}\,D \cup \{x, y\}$, $\mathsf{wv}\,G$ and $\mathsf{wv}\,H$ are disjoint. Then we have:

$$\begin{array}{ccc} \nu\vec{x}\,.\,(D, x := !\,\mathsf{rec}\,G\,\mathsf{in}\,M', \mathsf{local}\,H\,\mathsf{in}\,y := !N') & \to_c^{<n} & E \\ \downarrow_c & & \\ \nu\vec{x}\,.\,(D, \mathsf{local}\,G\,\mathsf{in}\,x := !M', \mathsf{local}\,H\,\mathsf{in}\,y := !N') & & \end{array}$$

   and so by confluence:

$$\begin{array}{ccc} \nu\vec{x}\,.\,(D, x := !\,\mathsf{rec}\,G\,\mathsf{in}\,M', \mathsf{local}\,H\,\mathsf{in}\,y := !N') & \to_c^{<n} & E \\ \downarrow_c & & \downarrow_c^{\le 1} \\ \nu\vec{x}\,.\,(D, \mathsf{local}\,G\,\mathsf{in}\,x := !M', \mathsf{local}\,H\,\mathsf{in}\,y := !N') & \to_c^{<n} & I \end{array}$$

and by (νMIG):

$$\nu\vec{x}\,.\,(D, \mathsf{local}\,G\,\mathsf{in}\,x := !M', \mathsf{local}\,H\,\mathsf{in}\,y := !N')$$
$$\equiv \nu\vec{x}\,.\,\nu(\mathsf{wv}\,G)\,.\,\nu(\mathsf{wv}\,H)\,.\,(D, G, H, x := !M', y := !N')$$

and from the definition of simulation:

$$\nu\vec{x}\,.\,\nu(\mathsf{wv}\,G)\,.\,\nu(\mathsf{wv}\,H)\,.\,(D, G, H, x := !M', y := !N') \vdash x \sim y$$

so by induction:

$$\nu\vec{x}\,.\,\nu(\mathsf{wv}\,G)\,.\,\nu(\mathsf{wv}\,H)\,.\,(D, G, H, x := !\nabla y, y := !N')\Downarrow_z \qquad (57)$$

and so:

$\nu\vec{x}\,.\,(D, x := !\nabla y, y := !N)$
- $\equiv \nu\vec{x}\,.\,(D, x := !\nabla y, y := !\,\mathsf{rec}H\,\mathsf{in}\,N')$ (Eqn 56)
- $\rightarrow \nu\vec{x}\,.\,(D, x := !\nabla y, \mathsf{local}\,H\,\mathsf{in}\,y := !N')$ (BUILD)
- $\leftarrow \nu\vec{x}\,.\,(D, \mathsf{local}\,G\,\mathsf{in}\,\varepsilon, x := !\nabla y, \mathsf{local}\,H\,\mathsf{in}\,y := !N')$ (γ)
- $\equiv \nu\vec{x}\,.\,\nu(\mathsf{wv}\,G)\,.\,\nu(\mathsf{wv}\,H)\,.\,(D, G, H, x := !\nabla y, y := !N')$ (νMIG)

and so by Equation 57 and Proposition 47:

$$\nu\vec{x}\,.\,(D, x := !\nabla y, y := !N)\Downarrow_z$$

The other cases are similar.

⇐ Is similar. □

3.9 Denotational properties

This section looks at some properties of the denotational semantics presented in Section 3.3. In particular, we shall show that:

- $\mathsf{wv}D$ can be determined from $[\![D]\!]$.
- Concatenation is a commutative monoid, so $[\![D,(E,F)]\!] = [\![(D,E),F]\!]$, $[\![D,\varepsilon]\!] = [\![D]\!]$ and $[\![D,E]\!] = [\![E,D]\!]$.
- Syntactic renaming corresponds to semantic renaming, for example $[\![M[y/x]]\!] = [\![M]\!] \circ (x := \mathsf{read}\,y)$.
- α-conversion is sound, so $[\![\nu x\,.\,D]\!] = [\![\nu y\,.\,([y/x]D[y/x])]\!]$ for fresh y.

We can use $\mathsf{read}\,x$ to make reasoning about this semantics easier:

PROPOSITION 53. *$f = g$ iff $\forall x\,.\,\mathsf{read}\,x \circ f = \mathsf{read}\,x \circ g$*

PROOF. Follows from the definition of $\mathsf{read}\,x$. □

For example, this means we could have used the following proposition as the definition of $\mathsf{new}\,x\,f$, $(x := f)$, and $\mathsf{set}\,X\,f\,g$:

PROPOSITION 54. *If $x \in X$ and $y \notin X$ then:*

1. $\mathsf{read}\,x \circ \mathsf{new}\,x\,f = \mathsf{read}\,x$
2. $\mathsf{read}\,y \circ \mathsf{new}\,x\,f = \mathsf{read}\,y \circ f$
3. $\mathsf{read}\,x \circ (x := f) = f$
4. $\mathsf{read}\,y \circ (x := f) = \mathsf{read}\,y$
5. $\mathsf{read}\,x \circ (\mathsf{set}\,X\,f\,g) = \mathsf{read}\,x \circ f \circ g$
6. $\mathsf{read}\,y \circ (\mathsf{set}\,X\,f\,g) = \mathsf{read}\,y$

PROOF. Follows from the definition of $\mathsf{read}\,x$, $\mathsf{new}\,x\,f$, $(x := f)$, and $\mathsf{set}\,X\,f\,g$. □

Another useful property of fix is *uniformity*:

PROPOSITION 55. *If $\forall i\,.\,f(gi) = h(fi)$ and $f\bot = \bot$ then $f(\mathsf{fix}\,g) = \mathsf{fix}\,h$.*

PROOF.

$$\begin{aligned}
&f(\mathsf{fix}\,g)\\
&= f(\bigvee\{g^n\bot \mid n \in \omega\}) && \text{(Defn of fix)}\\
&= \bigvee\{f(g^n\bot) \mid n \in \omega\} && (f \text{ is continuous})\\
&= \bigvee\{h^n(f\bot) \mid n \in \omega\} && \text{(Hypothesis)}\\
&= \bigvee\{h^n\bot \mid n \in \omega\} && \text{(Hypothesis)}\\
&= \mathsf{fix}\,h && \text{(Defn of fix)}
\end{aligned}$$

□

The written variables of a semantic function f can be defined as $\mathsf{wv}\,f$:

DEFINITION. $\mathsf{wv}\,f = \{x \mid \mathsf{read}\,x \circ f \neq \mathsf{read}\,x\}$. □

For example (when $h \neq \mathsf{read}\,x$):

$$\begin{aligned}
\mathsf{wv}(x := h) &= \{x\}\\
\mathsf{wv}(x := \mathsf{read}\,x) &= \emptyset\\
\mathsf{wv}(\mathsf{new}\,x\,f) &= (\mathsf{wv}\,f) \setminus \{x\}\\
\mathsf{wv}(\mathsf{set}\,X\,f\,g) &\subseteq X\\
\mathsf{wv}(f \circ g) &\subseteq \mathsf{wv}\,f \cup \mathsf{wv}\,g
\end{aligned}$$

We can show that the semantic and syntactic definitions of 'written variable' coincide.

PROPOSITION 56.

1. *If $\mathsf{wv}\,f \subseteq \mathsf{wv}D$ then $[\![D]\!] = [\![D]\!] \circ f$.*
2. $\mathsf{wv}D = \mathsf{wv}[\![D]\!]$

PROOF.

1. We can show that if $\mathsf{wv}\,f \subseteq X$ then:

$$\forall i\,.\,(\mathsf{set}\,X\,g\,i) \circ f = \mathsf{set}\,X\,g\,(i \circ f)$$

$$\bot \circ f = \bot$$

and so by uniformity:

$$\mathsf{fix}(\mathsf{set}\,X\,g) \circ f = \mathsf{fix}(\mathsf{set}\,X\,g)$$

From this it is easy to show by induction on D that $[\![D]\!] = [\![D]\!] \circ f$.

2. $(\mathsf{wv}[\![D]\!] \subseteq \mathsf{wv}\,D)$ An induction on D.

   $(\mathsf{wv}[\![D]\!] \supseteq \mathsf{wv}\,D)$ If $\mathsf{wv}[\![D]\!]\,\mathsf{wv}\,D$ then find $x \in \mathsf{wv}\,D$ and $x \notin \mathsf{wv}[\![D]\!]$. Then:

$$\begin{array}{lll} \top & & \\ = \mathsf{read}\,x \circ (x := \top) & & \text{(Propn 54.3)} \\ = \mathsf{read}\,x \circ [\![D]\!] \circ (x := \top) & & (x \notin \mathsf{wv}[\![D]\!]) \\ = \mathsf{read}\,x \circ [\![D]\!] \circ (x := \bot) \circ (x := \top) & & \text{(Part 1)} \\ = \mathsf{read}\,x \circ (x := \bot) \circ (x := \top) & & (x \notin \mathsf{wv}[\![D]\!]) \\ = \bot \circ (x := \top) & & \text{(Propn 54.3)} \\ = \bot & & (\bot \circ f = \bot) \end{array}$$

   This is a contradiction, and so $\mathsf{wv}[\![D]\!] \supseteq \mathsf{wv}\,D$. □

We can now show that concatenation is a commutative monoid.

PROPOSITION 57. *If $\mathsf{wv}\,f \subseteq X$, $\mathsf{wv}\,g \subseteq Y$, $\mathsf{wv}\,h \subseteq X$, and $X \cap Y = \emptyset$ then:*

1. $\mathsf{set}\,X\,f\,h = f \circ h$
2. $\mathsf{fix}(\mathsf{set}\,X\,f) = f \circ \mathsf{fix}(\mathsf{set}\,X\,f)$
3. $\mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g)) = g \circ \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g))$
4. $\mathsf{fix}(\mathsf{set}(X \cup Y)(g \circ f)) = \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g))$
5. $\mathsf{fix}(\mathsf{set}(X \cup Y)(\mathsf{fix}(\mathsf{set}\,X\,f) \circ \mathsf{fix}(\mathsf{set}\,Y\,g))) = \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g))$
6. $[\![D]\!] = \mathsf{fix}(\mathsf{set}(\mathsf{wv}\,D)[\![D]\!])$
7. $[\![D, \varepsilon]\!] = [\![D]\!]$
8. $[\![D, E]\!] = [\![E, D]\!]$
9. $[\![D, (E, F)]\!] = [\![(D, E), F]\!]$

PROOF.

1. For any $x \in X$:

$$\begin{array}{lll} \mathsf{read}\,x \circ \mathsf{set}\,X\,f\,h & & \\ = \mathsf{read}\,x \circ f \circ h & & \text{(Propn 54.5)} \end{array}$$

   For any $x \notin X$:

$$\begin{array}{lll} \mathsf{read}\,x \circ \mathsf{set}\,X\,f\,h & & \\ = \mathsf{read}\,x & & \text{(Propn 54.6)} \\ = \mathsf{read}\,x \circ f \circ h & & (x \notin \mathsf{wv}\,f \cup \mathsf{wv}\,h) \end{array}$$

   So by Proposition 53 $\mathsf{set}\,X\,f\,h = f \circ h$.

2. Follows from part 1.

3. For any $x \in Y$:

$$\begin{array}{lll} \mathsf{read}\,x \circ \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g)) & & \\ = \mathsf{read}\,x \circ f \circ g \circ \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g)) & & \text{(Part 1)} \\ = \mathsf{read}\,x \circ g \circ \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g)) & & (x \notin \mathsf{wv}\,f) \end{array}$$

   For any $x \notin Y$:

$$\begin{array}{lll} \mathsf{read}\,x \circ \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g)) & & \\ = \mathsf{read}\,x \circ g \circ \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g)) & & (x \notin \mathsf{wv}\,g) \end{array}$$

   So by Proposition 53:

$$\mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g)) = g \circ \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g))$$

4. For any $x \in X \cup Y$:

$$\begin{array}{lll} \mathsf{read}\,x \circ \mathsf{set}(X \cup Y)(g \circ f)(\mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g))) & & \\ = \mathsf{read}\,x \circ g \circ f \circ \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g)) & & \text{(Propn 54.5)} \\ = \mathsf{read}\,x \circ g \circ f \circ g\,\mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g)) & & \text{(Part 3)} \\ = \mathsf{read}\,x \circ g \circ \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g)) & & \text{(Propn 57.2)} \\ = \mathsf{read}\,x \circ \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g)) & & \text{(Part 3)} \end{array}$$

   and for any $x \notin X \cup Y$:

$$\begin{array}{lll} \mathsf{read}\,x \circ \mathsf{set}(X \cup Y)(g \circ f)(\mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g))) & & \\ = \mathsf{read}\,x & & \text{(Propn 54.6)} \\ = \mathsf{read}\,x \circ \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g)) & & \text{(Propn 54.6)} \end{array}$$

   and so by Proposition 53:

$$\mathsf{set}(X \cup Y)(g \circ f)(\mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g))) = \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g))$$

   and since $\mathsf{fix}\,f$ is the least fixed point of f:

$$\mathsf{fix}(\mathsf{set}(X \cup Y)(g \circ f)) \leq \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g))$$

   and so by a symmetrical argument:

$$\mathsf{fix}(\mathsf{set}(X \cup Y)(g \circ f)) = \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g))$$

5. If $f = g \circ f$, we can show by induction on n that $(\mathsf{set}\,X\,g)^n \bot \circ f \leq f$:

   - $(\mathsf{set}\,X\,g)^0 \bot \circ f = \bot \circ f \leq f$.
   - If $x \in X$ then:

$$\begin{array}{lll} \mathsf{read}\,x \circ (\mathsf{set}\,X\,g)^{n+1} \bot \circ f & & \\ = \mathsf{read}\,x \circ (\mathsf{set}\,X\,g)((\mathsf{set}\,X\,g)^n \bot) \circ f & & \text{(Defn of } f^n) \\ = \mathsf{read}\,x \circ g \circ (\mathsf{set}\,X\,g)^n \bot \circ f & & \text{(Propn 54.5)} \\ \leq \mathsf{read}\,x \circ g \circ f & & \text{(Indn)} \end{array}$$

$$= \mathsf{read}\, x \circ f \qquad (f = g \circ f)$$

If $x \notin X$ then:

$$\begin{array}{ll} & \mathsf{read}\, x \circ (\mathsf{set}\, X g)^{n+1} \bot \circ f \\ = & \mathsf{read}\, x \circ (\mathsf{set}\, X g)((\mathsf{set}\, X g)^n \bot) \circ f & \text{(Defn of } f^n\text{)} \\ = & \mathsf{read}\, x \circ f & \text{(Propn 54.6)} \end{array}$$

Thus $(\mathsf{set}\, X g)^{n+1} \bot \circ f \leq f$.

Thus:

$$\begin{array}{ll} & f = g \circ f \\ \Rightarrow & \bigvee \{(\mathsf{set}\, X g)^n \bot \circ f \mid n \text{ in } \omega\} \leq f & \text{(Above)} \\ \Rightarrow & \bigvee \{(\mathsf{set}\, X g)^n \bot \mid n \text{ in } \omega\} \circ f \leq f & (\circ \text{ is continuous}) \\ \Rightarrow & \mathsf{fix}(\mathsf{set}\, X g) \circ f \leq f & \text{(Defn of fix)} \end{array}$$

For example, if $\mathsf{wv}\, f = X$, $\mathsf{wv}\, g = Y$ and $X \cap Y = \emptyset$ then we have by part 3:

$$\mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g)) = f \circ \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g))$$

and so by the above:

$$\mathsf{fix}(\mathsf{set}\, X f) \circ \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g)) \leq \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g)) \qquad (58)$$

Similarly:

$$\mathsf{fix}(\mathsf{set}\, Y g) \circ \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g)) \leq \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g)) \qquad (59)$$

Thus:

$$\begin{array}{ll} & \mathsf{set}(X \cup Y)(\mathsf{fix}(\mathsf{set}\, X f) \circ \mathsf{fix}(\mathsf{set}\, Y g))(\mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g))) \\ = & \mathsf{fix}(\mathsf{set}\, X f) \circ \mathsf{fix}(\mathsf{set}\, Y g) \circ \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g)) & \text{(Propn 57.1)} \\ \leq & \mathsf{fix}(\mathsf{set}\, X f) \circ \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g)) & \text{(Eqn 58)} \\ \leq & \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g)) & \text{(Eqn 59)} \end{array}$$

And so since $gf \leq f \Rightarrow \mathsf{fix}\, g \leq f$:

$$\mathsf{fix}(\mathsf{set}(X \cup Y)(\mathsf{fix}(\mathsf{set}\, X f) \circ \mathsf{fix}(\mathsf{set}\, Y g))) \leq \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g)) \qquad (60)$$

Similarly, we can show:

$$\begin{array}{ll} & \mathsf{fix}(\mathsf{set}(X \cup Y)(\mathsf{fix}(\mathsf{set}\, X f) \circ \mathsf{fix}(\mathsf{set}\, Y g))) \\ = & \mathsf{fix}(\mathsf{set}(X \cup Y)(\mathsf{fix}(\mathsf{set}\, Y g) \circ \mathsf{fix}(\mathsf{set}\, X f))) & \text{(Propn 57.4)} \\ = & \mathsf{fix}(\mathsf{set}\, X f) \circ \mathsf{fix}(\mathsf{set}(X \cup Y)(\mathsf{fix}(\mathsf{set}\, Y g) \circ \mathsf{fix}(\mathsf{set}\, X f))) & \text{(Propn 57.3)} \\ = & f \circ \mathsf{fix}(\mathsf{set}\, X f) \circ \mathsf{fix}(\mathsf{set}(X \cup Y)(\mathsf{fix}(\mathsf{set}\, Y g) \circ \mathsf{fix}(\mathsf{set}\, X f))) & \text{(Propn 57.2)} \\ = & f \circ \mathsf{fix}(\mathsf{set}(X \cup Y)(\mathsf{fix}(\mathsf{set}\, Y g) \circ \mathsf{fix}(\mathsf{set}\, X f))) & \text{(Propn 57.3)} \\ = & f \circ g \circ \mathsf{fix}(\mathsf{set}(X \cup Y)(\mathsf{fix}(\mathsf{set}\, X f) \circ \mathsf{fix}(\mathsf{set}\, Y g))) & \text{(Similar)} \\ = & \mathsf{set}(X \cup Y)(f \circ g)(\mathsf{fix}(\mathsf{set}(X \cup Y) \\ & \quad (\mathsf{fix}(\mathsf{set}\, X f) \circ \mathsf{fix}(\mathsf{set}\, Y g)))) & \text{(Propn 57.1)} \end{array}$$

and so, since $f = gf \Rightarrow \mathsf{fix}\, g \leq f$:

$$\mathsf{fix}(\mathsf{set}(X \cup Y)(\mathsf{fix}(\mathsf{set}\, X f) \circ \mathsf{fix}(\mathsf{set}\, Y g))) \geq \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g))$$

which, together with Equation 60 gives:

$$\mathsf{fix}(\mathsf{set}(X \cup Y)(\mathsf{fix}(\mathsf{set}\, X f) \circ \mathsf{fix}(\mathsf{set}\, Y g))) = \mathsf{fix}(\mathsf{set}(X \cup Y)(f \circ g))$$

6. We have:

$$\begin{array}{ll} & \mathsf{fix}(\mathsf{set}\, X f) \\ = & \mathsf{fix}(\mathsf{set}\, X (f \circ \mathsf{id})) & \text{(Identity)} \\ = & \mathsf{fix}(\mathsf{set}\, X (\mathsf{fix}(\mathsf{set}\, X f) \circ \mathsf{fix}(\mathsf{set}\, \emptyset\, \mathsf{id}))) & \text{(Part 5)} \\ = & \mathsf{fix}(\mathsf{set}\, X (\mathsf{fix}(\mathsf{set}\, X f) \circ \mathsf{set}\, \emptyset\, \mathsf{id}(\mathsf{fix}(\mathsf{set}\, \emptyset\, \mathsf{id})))) & \text{(Unfold)} \\ = & \mathsf{fix}(\mathsf{set}\, X (\mathsf{fix}(\mathsf{set}\, X f) \circ \mathsf{id})) & (\mathsf{set}\, \emptyset\, f g = \mathsf{id}) \\ = & \mathsf{fix}(\mathsf{set}\, X (\mathsf{fix}(\mathsf{set}\, X f))) & \text{(Identity)} \end{array}$$

so we can show by induction on D that $[\![D]\!] = \mathsf{fix}(\mathsf{set}(\mathsf{wv}\, D)[\![D]\!])$.

7. Follows immediately from the definition of $[\![D, \varepsilon]\!]$.

8. Follows immediately from part 4.

9. We have:

$$\begin{array}{ll} & [\![D, (E, F)]\!] \\ = & \mathsf{fix}(\mathsf{set}(\mathsf{wv}(D, E, F))([\![D]\!] \\ & \quad \circ \mathsf{fix}(\mathsf{wv}(E, F))([\![E]\!] \circ [\![F]\!]))) & \text{(Defn of } [\![D, E]\!]\text{)} \\ = & \mathsf{fix}(\mathsf{set}(\mathsf{wv}(D, E, F))(\mathsf{fix}(\mathsf{set}(\mathsf{wv}\, D)[\![D]\!]) \\ & \quad \circ \mathsf{fix}(\mathsf{wv}(E, F))([\![E]\!] \circ [\![F]\!]))) & \text{(Part 6)} \\ = & \mathsf{fix}(\mathsf{set}(\mathsf{wv}(D, E, F))([\![D]\!] \circ [\![E]\!] \circ [\![F]\!])) & \text{(Part 5)} \\ = & [\![(D, E), F]\!] & \text{(Similar)} \end{array}$$

Thus concatenation is a commuatative monoid. □

Finally, we can show the relationship between semantic and syntactic relabelling of free variables, and thus show the soundness of α-conversion.

PROPOSITION 58. *If z is fresh and $x \notin \mathsf{fv}\, E$ then:*

1. $[\![M[y/x]]\!] = [\![M]\!] \circ (x := \mathsf{read}\, y)$
2. $(x := \mathsf{read}\, y) \circ [\![D[y/x]]\!] = [\![D]\!] \circ (x := \mathsf{read}\, y)$ *for any* $x, y \notin \mathsf{wv}\, D$
3. $(x := \mathsf{read}\, z) \circ [\![[z/x]D[z/x]]\!] = (z := \mathsf{read}\, x) \circ [\![D]\!] \circ (x := \mathsf{read}\, z)$
4. $[\![\nu x \,.\, D]\!] = [\![\nu z \,.\, ([z/x]D[z/x])]\!]$
5. $[\![\nu x \,.\, \nu y \,.\, D]\!] = [\![\nu y \,.\, \nu x \,.\, D]\!]$
6. $[\![D, \nu x \,.\, E]\!] = [\![\nu x \,.\, (D, E)]\!]$

PROOF. An induction on the size of D and M.

1. The difficult case is to show:

$$[\![(\mathsf{rec}\, D \,\mathsf{in}\, M)[y/x]]\!] = [\![\mathsf{rec}\, D \,\mathsf{in}\, M]\!] \circ (x := \mathsf{read}\, y)$$

If $x \in \mathsf{wv}\,D$ then:

$$\begin{array}{ll}
[\![(\mathsf{rec}D \,\mathsf{in}\, M)[y/x]]\!] & \\
\quad = [\![\mathsf{rec}D \,\mathsf{in}\, M]\!] & \text{(Defn of substitution)} \\
\quad = [\![M]\!] \circ [\![D]\!] & \text{(Defn of } [\![\mathsf{rec}D \,\mathsf{in}\, M]\!]) \\
\quad = [\![M]\!] \circ [\![D]\!] \circ (x := \mathsf{read}\,y) & \text{(Propn 56.1)} \\
\quad = [\![\mathsf{rec}D \,\mathsf{in}\, M]\!] \circ (x := \mathsf{read}\,y) & \text{(Defn of } [\![\mathsf{rec}D \,\mathsf{in}\, M]\!])
\end{array}$$

Otherwise, if $y \in \mathsf{wv}\,D$ then, for fresh z:

$$\begin{array}{ll}
[\![(\mathsf{rec}D \,\mathsf{in}\, M)[y/x]]\!] & \\
\quad = [\![\mathsf{rec}[z/y]D[z/y][y/x] \,\mathsf{in}\, M[z/y][y/x]]\!] & \text{(Defn of substitution)} \\
\quad = [\![M[z/y][y/x]]\!] \circ [\![[z/y]D[z/y][y/x]]\!] & \text{(Defn of } [\![\mathsf{rec}D \,\mathsf{in}\, M]\!]) \\
\quad = [\![M]\!] \circ (y := \mathsf{read}\,z) \circ (x := \mathsf{read}\,y) \circ [\![[z/y]D[z/y][y/x]]\!] & \text{(Indn on 1)} \\
\quad = [\![M]\!] \circ (y := \mathsf{read}\,z) \circ [\![[z/y]D[z/y]]\!] \circ (x := \mathsf{read}\,y) & \text{(Indn on 2)} \\
\quad = [\![M]\!] \circ (z := \mathsf{read}\,y) \circ [\![D]\!] \circ (y := \mathsf{read}\,z) \circ (x := \mathsf{read}\,y) & \text{(Indn on 3)} \\
\quad = [\![M[y/z]]\!] \circ [\![D]\!] \circ (y := \mathsf{read}\,z) \circ (x := \mathsf{read}\,y) & \text{(Indn on 1)} \\
\quad = [\![M]\!] \circ [\![D]\!] \circ (y := \mathsf{read}\,z) \circ (x := \mathsf{read}\,y) & (z \text{ is fresh}) \\
\quad = [\![M]\!] \circ [\![D]\!] \circ (x := \mathsf{read}\,y) & \text{(Propn 56.1)} \\
\quad = [\![\mathsf{rec}D \,\mathsf{in}\, M]\!] \circ (x := \mathsf{read}\,y) & \text{(Defn of } [\![\mathsf{rec}D \,\mathsf{in}\, M]\!])
\end{array}$$

Otherwise:

$$\begin{array}{ll}
[\![(\mathsf{rec}D \,\mathsf{in}\, M)[y/x]]\!] & \\
\quad = [\![\mathsf{rec}D[y/x] \,\mathsf{in}\, M[y/x]]\!] & \text{(Defn of substitution)} \\
\quad = [\![M[y/x]]\!] \circ [\![D[y/x]]\!] & \text{(Defn of } [\![\mathsf{rec}D \,\mathsf{in}\, M]\!]) \\
\quad = [\![M]\!] \circ (x := \mathsf{read}\,y) \circ [\![D[y/x]]\!] & \text{(Indn on 1)} \\
\quad = [\![M]\!] \circ [\![D]\!] \circ (x := \mathsf{read}\,y) & \text{(Indn on 2)} \\
\quad = [\![\mathsf{rec}D \,\mathsf{in}\, M]\!] \circ (x := \mathsf{read}\,y) & \text{(Defn of } [\![\mathsf{rec}D \,\mathsf{in}\, M]\!])
\end{array}$$

The other cases are simpler.

2. The difficult cases are those involving fix, for example to show:

$$(x := \mathsf{read}\,y) \circ [\![z := !M[y/x]]\!] = [\![z := !M]\!] \circ (x := \mathsf{read}\,y)$$

We first show:

$$\begin{array}{c}
\forall i \,.\, (x := \mathsf{read}\,y) \circ \mathsf{set}\{z\}((z := [\![M]\!]) \circ (x := \mathsf{read}\,y))i \\
= (\mathsf{set}\{x,z\}(x := \mathsf{read}\,y) \circ \{z\}(z := [\![M]\!]))((x := \mathsf{read}\,y) \circ i) \qquad (61) \\
(x := \mathsf{read}\,y) \circ \bot = \bot
\end{array}$$

and:

$$\begin{array}{c}
\forall i \,.\, \mathsf{set}\{z\}(z := [\![M]\!])i \circ (x := \mathsf{read}\,y) \\
= (\mathsf{set}\{x,z\}(x := \mathsf{read}\,y) \circ \{z\}(z := [\![M]\!]))(i \circ (x := \mathsf{read}\,y)) \qquad (62) \\
\bot \circ (x := \mathsf{read}\,y) = \bot
\end{array}$$

Then we can use uniformity twice to show:

$$\begin{array}{ll}
(x := \mathsf{read}\,y) \circ [\![z := !M[y/x]]\!] & \\
\quad = (x := \mathsf{read}\,y) \circ \mathsf{fix}(\mathsf{set}\{z\}(z := [\![M[y/x]]\!])) & \text{(Defn of } [\![x := !M]\!]) \\
\quad = (x := \mathsf{read}\,y) \circ \mathsf{fix}(\mathsf{set}\{z\}((z := [\![M]\!]) \circ (x := \mathsf{read}\,y))) & \text{(Indn on 1)} \\
\quad = \mathsf{fix}(\mathsf{set}\{x,z\}(x := \mathsf{read}\,y) \circ \mathsf{set}\{z\}(z := [\![M]\!])) & \text{(Unif on Eqn 61)} \\
\quad = \mathsf{fix}(\mathsf{set}\{z\}(z := [\![M]\!])) \circ (x := \mathsf{read}\,y) & \text{(Unif on Eqn 62)} \\
\quad = [\![z := !M]\!] \circ (x := \mathsf{read}\,y) & \text{(Defn of } [\![x := !M]\!])
\end{array}$$

The other cases are similar.

3. This has a similar proof to part 2.

4. If $y \notin \mathsf{wv}(\nu x \,.\, D)$ then:

$$\begin{array}{ll}
\mathsf{read}\,y \circ [\![\nu x \,.\, D]\!] & \\
\quad = \mathsf{read}\,y & (y \notin \mathsf{wv}[\![\nu x \,.\, D]\!]) \\
\quad = \mathsf{read}\,y \circ [\![\nu z \,.\, ([z/x]D[z/x])]\!] & (y \notin \mathsf{wv}[\![\nu z \,.\, ([z/x]D[z/x])]\!])
\end{array}$$

Otherwise:

$$\begin{array}{ll}
\mathsf{read}\,y \circ [\![\nu x \,.\, D]\!] & \\
\quad = \mathsf{read}\,y \circ \mathsf{new}\,x[\![D]\!] & \text{(Defn of } [\![\nu x \,.\, D]\!]) \\
\quad = \mathsf{read}\,y \circ [\![D]\!] & \text{(Propn 54.2)} \\
\quad = \mathsf{read}\,y \circ (z := \mathsf{read}\,x) \circ [\![D]\!] & \text{(Propn 54.4)} \\
\quad = \mathsf{read}\,y \circ (z := \mathsf{read}\,x) \circ [\![D]\!] \circ (x := \mathsf{read}\,z) & \text{(Propn 56.1)} \\
\quad = \mathsf{read}\,y \circ (x := \mathsf{read}\,z) \circ [\![[z/x]D[z/x]]\!] & \text{(Indn on 3)} \\
\quad = \mathsf{read}\,y \circ [\![[z/x]D[z/x]]\!] & \text{(Propn 54.4)} \\
\quad = \mathsf{read}\,y \circ \mathsf{new}\,z[\![[z/x]D[z/x]]\!] & \text{(Propn 54.2)} \\
\quad = \mathsf{read}\,y \circ [\![\nu z \,.\, [z/x]D[z/x]]\!] & \text{(Defn of } [\![\nu x \,.\, D]\!])
\end{array}$$

Thus for any y:

$$\mathsf{read}\,y \circ [\![\nu x \,.\, D]\!] = \mathsf{read}\,y \circ [\![\nu z \,.\, [z/x]D[z/x]]\!]$$

and so $[\![\nu x \,.\, D]\!] = [\![\nu z \,.\, [z/x]D[z/x]]\!]$.

5. Follows from the definition of new.

6. To begin with, we can show by induction on D that if $x \notin \mathsf{fv}\,D$ then:

$$(x := \bot \circ [\![D]\!]) = ([\![D]\!] \circ x := \bot) \qquad (63)$$

We can also show that:

$$(x := \bot \circ f) = (x := \bot \circ \mathsf{new}\,x\,f) \qquad (64)$$

Then for any i:

$$\begin{array}{ll}
x := \bot \circ (\mathsf{set}(\mathsf{wv}(D,E))([\![D]\!] \circ [\![E]\!])i) & \\
\quad = x := \bot \circ [\![D]\!] \circ [\![E]\!] \circ i & \text{(Propn 57.1)}
\end{array}$$

$$
\begin{aligned}
&= [\![D]\!] \circ x := \bot \circ [\![E]\!] \circ i && \text{(Eqn 63)}\\
&= [\![D]\!] \circ x := \bot \circ [\![E]\!] \circ x := \bot \circ i && \text{(Propn 56.1)}\\
&= [\![D]\!] \circ x := \bot \circ \mathsf{new}\, x [\![E]\!] \circ x := \bot \circ i && \text{(Eqn 64)}\\
&= x := \bot \circ [\![D]\!] \circ \mathsf{new}\, x [\![E]\!] \circ x := \bot \circ i && \text{(Eqn 63)}\\
&= \mathsf{set}(\mathsf{wv}(D,E))(x := \bot \circ [\![D]\!] \circ \mathsf{new}\, x [\![E]\!])(x := \bot \circ i) && \text{(Eqn 57.1)}
\end{aligned}
$$

Thus, since $x := \bot \circ \bot = \bot$, we can apply unification and get:

$$
\begin{aligned}
x := \bot \circ \mathsf{fix}(\mathsf{set}(\mathsf{wv}(D,E))([\![D]\!] \circ [\![E]\!])) \\
= \mathsf{fix}(\mathsf{set}(\mathsf{wv}(D,E))(x := \bot \circ [\![D]\!] \circ \mathsf{new}\, x [\![E]\!]))
\end{aligned}
\qquad (65)
$$

Similarly, we can show:

$$
\begin{aligned}
x := \bot \circ \mathsf{fix}(\mathsf{set}(\mathsf{wv}(D, \nu x\,.\,E))([\![D]\!] \circ \mathsf{new}\, x [\![E]\!])) \\
= \mathsf{fix}(\mathsf{set}(\mathsf{wv}(D,E))(x := \bot \circ [\![D]\!] \circ \mathsf{new}\, x [\![E]\!]))
\end{aligned}
\qquad (66)
$$

And so:

$$
\begin{aligned}
& x := \bot \circ [\![\nu x\,.\,(D,E)]\!] \\
&\quad = x := \bot \circ \mathsf{new}\, x (\mathsf{fix}(\mathsf{set}(\mathsf{wv}(D,E))([\![D]\!] \circ [\![E]\!]))) && \text{(Defn of } [\![\nu x\,.\,(D,E)]\!]) \\
&\quad = x := \bot \circ \mathsf{fix}(\mathsf{set}(\mathsf{wv}(D,E))([\![D]\!] \circ [\![E]\!])) && \text{(Eqn 64)} \\
&\quad = \mathsf{fix}(\mathsf{set}(\mathsf{wv}(D,E))(x := \bot \circ [\![D]\!] \circ \mathsf{new}\, x [\![E]\!])) && \text{(Eqn 65)} \\
&\quad = x := \bot \circ \mathsf{fix}(\mathsf{set}(\mathsf{wv}(D, \nu x s t E))([\![D]\!] \circ \mathsf{new}\, x [\![E]\!])) && \text{(Eqn 66)} \\
&\quad = x := \bot \circ [\![D, \nu x\,.\,E]\!] && \text{(Defn of } [\![D, \nu x\,.\,E]\!])
\end{aligned}
$$

From this, it is easy to show that $[\![\nu x\,.\,(D,E)]\!] = [\![D, \nu x\,.\,E]\!]$. □

3.10 Logical properites

This section looks at some properties of the operational characterization of the program logic given in Section 3.4. In particular, we shall show that:

- The logic respects duplication, so $\models (D, x := !M, y := !M)[x/z] : \Delta$ iff $\models (D, x := !M, y := !M)[y/z] : \Delta$.
- The logic is referentially transparent, so $\models (D, x := !\nabla y)[x/z] : \Delta$ iff $\models (D, x := !\nabla y)[y/z] : \Delta$.
- The logic is unaffected by local variables, so if $x \notin \mathsf{wv}\,\Delta$ then $\models D : \Delta$ iff $\models \nu x\,.\,D : \nu x\,.\,\Delta$.
- The logic respects reduction, so if $D \to E$ then $\models D : \Delta$ iff $\models E : \Delta$.

PROPOSITION 59.

1. *If y is fresh and $\models D : \Delta$ then $\models [y/x]D[y/x] : [y/x]\Delta$.*
2. *If y is fresh and $\Gamma \models D : \Delta$ then $[y/x]\Gamma \models [y/x]D[y/x] : [y/x]\Delta$.*
3. *If y is fresh and $\Gamma \models M : \phi$ then $[y/x]\Gamma \models M[y/x] : \phi$.*

PROOF. Follows from Proposition 24. □

PROPOSITION 60. *If D is closed and $\mathsf{wv}\,E \cap \mathsf{wv}\,\Delta = \emptyset$ then $\models D : \Delta$ iff $\models (D,E) : \Delta$.*

PROOF. We show by induction on ϕ that $\models D : (x : \phi)$ iff $\models (D,E) : (x : \phi)$. The only difficult case is when $\phi = \psi \to \chi$:

⇒ Assume $\models D : (x : \psi \to \chi)$. Then $D \Downarrow_x$ so by Proposition 48 we have $(D,E) \Downarrow_x$. Then for any $(z := !x@y) \sqsubseteq F \sqsupseteq (D,E) \sqsupseteq D$, by induction:

$$\models F : (y : \psi) \Rightarrow \models F : (z : \chi)$$

and so $\models (D,E) : (x : \psi \to \chi)$.

⇐ Assume $\models (D,E) : (x : \psi \to \chi)$. Then $(D,E) \Downarrow_x$ so by Proposition 48 we have $D \Downarrow_x$. Then for any $(z := !x@y) \sqsubseteq F \sqsupseteq D$, let $\vec{w} = \mathsf{wv}\,E$, and $\vec{v}$ be fresh, so:

$$
\begin{aligned}
&\models (F, z := !x@y) : (y : \psi) \\
&\quad \Rightarrow \models [\vec{v}/\vec{w}](F, z := !x@y)[\vec{v}/\vec{w}] : [\vec{v}/\vec{w}](y : \psi) && \text{(Propn 59)} \\
&\quad \Rightarrow \models ([\vec{v}/\vec{w}](F, z := !x@y)[\vec{v}/\vec{w}], E) : [\vec{v}/\vec{w}](y : \psi) && \text{(Indn)} \\
&\quad \Rightarrow \models ([\vec{v}/\vec{w}](F, z := !x@y)[\vec{v}/\vec{w}], E) : [\vec{v}/\vec{w}](z : \chi) && \text{(Hypothesis)} \\
&\quad \Rightarrow \models [\vec{v}/\vec{w}](F, z := !x@y)[\vec{v}/\vec{w}] : [\vec{v}/\vec{w}](z : \chi) && \text{(Indn)} \\
&\quad \Rightarrow \models (F, z := !x@y) : (z : \chi) && \text{(Propn 59)}
\end{aligned}
$$

and so $\models D : (x : \psi \to \chi)$.

Then by induction on Δ, $\models D : \Delta$ iff $\models (D, w := !M) : \Delta$. □

PROPOSITION 61. *For closed declarations:*

1. *If $\models (D, w := !M, x := !M) : (w : \phi)$ then $\models (D, w := !M, x := !M) : (x : \phi)$.*
2. *If $\models (D, w := !M, x := !M, z := !w@y) : \Delta$ then $\models (D, w := !M, x := !M, z := !x@y) : \Delta$.*
3. *$\models (D, x := !\nabla w) : (x : \phi)$ iff $\models (D, x := !\nabla w) : (w : \phi)$.*
4. *$\models (D, x := !\nabla w, z := !x@y) : \Delta$ iff $\models (D, x := !\nabla w, z := !w@y) : \Delta$.*
5. *$\models (D, x := !M, y := !M)[x/z] : \Delta$ iff $\models (D, x := !M, y := !M)[y/z] : \Delta$.*
6. *$\models (D, x := !\nabla y)[x/z] : \Delta$ iff $\models (D, x := !\nabla y)[y/z] : \Delta$.*
7. *$\models (D, x := ?M) : \Delta$ iff $\models (D, x := !M) : \Delta$*

PROOF. These all have similar proofs, so we shall show parts 1 and 2 as an example. We shall show by induction on ϕ that:

1. If $\models (D, w := !M, x := !M) : (w : \phi)$ then $\models (D, w := !M, x := !M) : (x : \phi)$.
2. If $\models (D, w := !M, x := !M, z := !w@y) : (v : \phi)$ then $\models (D, w := !M, x := !M, z := !x@y) : (v : \phi)$.

From this it is easy to show parts 1 and 2. The only difficult case is when $\phi = \psi \to \chi$.

1. Assume:

$$\models (D, w := !M, x := !M) : (w : \psi \to \chi) \tag{67}$$

Then $(D, w := !M, x := !M)\Downarrow_w$ so by Proposition 51.2:

$$(D, w := !M, x := !M)\Downarrow_x$$

For any $(z := !x@y) \sqsubseteq E \sqsupseteq (D, w := !M, x := !M)$ either:

- $z = w$ or $z = x$, so $M = x@y$, so $(D, w := !M, x := !M)\Uparrow_x$, which is a contradiction.
- $w \neq z \neq x$, so by Proposition 22, we can find F such that:

$$(F, w := !M, x := !M, z := !x@y) \equiv E \qquad (F, z := !x@y) \sqsupseteq D \tag{68}$$

Then, for fresh v:

$\models E : (y : \psi)$
$\Rightarrow \models (F, w := !M, x := !M, z := !x@y) : (y : \psi)$ (Eqn 68)
$\Rightarrow \models (F, w := !M, x := !M, z := !x@y, v := !x@y) : (y : \psi)$ (Propn 60)
$\Rightarrow \models (F, w := !M, x := !M, z := !x@y, v := !w@y) : (y : \psi)$ (Indn on 2)
$\Rightarrow \models (F, w := !M, x := !M, z := !x@y, v := !w@y) : (v : \chi)$ (Eqn 67)
$\Rightarrow \models (F, w := !M, x := !M, z := !x@y, v := !x@y) : (v : \chi)$ (Indn on 2)
$\Rightarrow \models (F, w := !M, x := !M, z := !x@y, v := !x@y) : (z : \chi)$ (Indn on 1)
$\Rightarrow \models (F, w := !M, x := !M, z := !x@y) : (z : \chi)$ (Propn 60)
$\Rightarrow E : (z : \chi)$ (Above)

Thus for any $(z := !x@y) \sqsubseteq E \sqsupseteq (D, w := !M, x := !M)$:

$$\models E : (y : \psi) \Rightarrow \models E : (z : \chi)$$

and so $\models (D, w := !M, x := !M) : (x : \psi \to \chi)$.

2. Assume:

$$\models (D, w := !M, x := !M, z := !w@y) : (v : \psi \to \chi) \tag{69}$$

Then $(D, w := !M, x := !M, z := !w@y)\Downarrow_v$, and so by Proposition 51.1, we have $(D, w := !M, x := !M, z := !x@y)\Downarrow_v$.

For any $(t := !v@u) \sqsubseteq E \sqsupseteq (D, w := !M, x := !M, z := !x@y)$ either:

- $t = z$, so $v = w$ and $u = y$. Then:

$\models (D, w := !M, x := !M, z := !w@y) : (v : \psi \to \chi)$
$\Rightarrow \models (D, w := !M, x := !M, z := !w@y) : (w : \psi \to \chi)$ $(v = w)$
$\Rightarrow \models (D, w := !M, x := !M, z := !w@y) : (x : \psi \to \chi)$ (Part 1)

and so, since $(z := !w@y) \sqsubseteq E \sqsupseteq (D, w := !M, x := !M, z := !w@y)$:

$$\models E : (u : \psi)$$

$\Rightarrow \models E : (y : \psi)$ $(u = y)$
$\Rightarrow \models E : (z : \chi)$ (Above)
$\Rightarrow \models E : (t : \chi)$ $(t = z)$

- $t \neq z$ so by Proposition 22, we can find F such that:

$$E \equiv (F, x := !M, y := !M, z := !x@y) \quad F \sqsupseteq D \quad F \sqsupseteq (t := !v@u) \tag{70}$$

and so:

$\models E : (u : \psi)$
$\Rightarrow \models (F, x := !M, y := !M, z := !x@y) : (u : \psi)$ (Eqn 70)
$\Rightarrow \models (F, w := !M, x := !M, z := !w@y) : (u : \psi)$ (Indn on 2)
$\Rightarrow \models (F, w := !M, x := !M, z := !w@y) : (t : \chi)$ (Eqn 69)
$\Rightarrow \models (F, w := !M, x := !M, z := !x@y) : (t : \chi)$ (Indn on 2)
$\Rightarrow \models E : (t : \chi)$ (Indn on 2)

Thus for any $(t := !v@u) \sqsubseteq E \sqsupseteq (D, w := !M, x := !M, z := !x@y)$:

$$\models E : (u : \psi) \Rightarrow \models E : (t : \chi)$$

and so $\models (D, w := !M, x := !M, z := !x@y) : (v : \psi \to \chi)$. □

PROPOSITION 62. *If* $(\Gamma, w : \phi) \models M : \psi$ *then* $(\Gamma \wedge y : \phi) \models M[y/w] : \psi$.

PROOF. For any D and x, find a fresh z, and so by Proposition 59 we have:

$$(\Gamma, z : \phi) \models M[z/w] : \psi \tag{71}$$

and:

$\models (D, x := !M[y/w]) : \Gamma \wedge (y : \phi)$
$\Rightarrow (D, x := !M[z/w][y/z]) : \Gamma \wedge (y : \phi)$ (Substitution)
$\Rightarrow (D, x := !M[z/w][y/z], z := !\nabla y) : \Gamma \wedge (y : \phi)$ (Propn 60)
$\Rightarrow (D, x := !M[z/w], z := !\nabla y) : \Gamma \wedge (y : \phi)$ (Propn 61.6)
$\Rightarrow (D, x := !M[z/w], z := !\nabla y) : \Gamma \wedge (z : \phi)$ (Propn 61.3)
$\Rightarrow (D, x := !M[z/w], z := !\nabla y) : (\Gamma, z : \phi)$ (z is fresh)
$\Rightarrow (D, x := !M[z/w], z := !\nabla y) : (x : \psi)$ (Eqn 71)
$\Rightarrow (D, x := !M[z/w][y/z], z := !\nabla y) : (x : \psi)$ (Propn 61.6)
$\Rightarrow (D, x := !M[z/w][y/z]) : (x : \psi)$ (Propn 60)
$\Rightarrow (D, x := !M[y/w]) : (x : \psi)$ (Substitution)

Thus $\Gamma \wedge (y : \phi) \models M[y/w] : \psi$. □

PROPOSITION 63. *For closed* D:

1. *If* $w \neq x$ *then* $\models D : (x : \phi)$ *iff* $\models \nu w.\, D : (x : \phi)$.
2. *If* $\models D : \Delta$ *then* $\models \nu w.\, D : \nu w.\, \Delta$.
3. *If* $\models \nu w.\, D : \Delta$ *then* $\models D : \Delta$.

PROOF.

1. An induction on ϕ. The only difficult case is when $\phi = \psi \to \chi$.

   $\Rightarrow$ If $\models D : (x : \psi \to \chi)$ then $D{\Downarrow_x}$ so by Proposition 29 $\nu w.\, D{\Downarrow_x}$. For any $(z := !x@y) \sqsubseteq E \sqsupseteq (\nu w.\, D)$, let v be fresh, by Proposition 22, we can find $F \sqsupseteq (z := !x@y)$ such that:

   $$E \equiv \nu v.\, F \qquad F \sqsupseteq [v/w]D[v/w] \tag{72}$$

   so by Proposition 59:

   $$\models [v/w]D[v/w] : (x : \psi \to \chi) \tag{73}$$

   and so:

   $$\begin{array}{lr} \models E : (y : \psi) & \\ \quad \Rightarrow \models \nu v.\, F : (y : \psi) & \text{(Eqn 72)} \\ \quad \Rightarrow \models F : (y : \psi) & \text{(Indn)} \\ \quad \Rightarrow \models F : (z : \chi) & \text{(Eqn 73)} \\ \quad \Rightarrow \models \nu v.\, F : (z : \chi) & \text{(Indn)} \\ \quad \Rightarrow \models E : (z : \chi) & \text{(Eqn 72)} \end{array}$$

   Thus $\models \nu w.\, D : (x : \psi \to \chi)$.

   $\Leftarrow$ If:

   $$\models \nu w.\, D : (x : \psi \to \chi) \tag{74}$$

   then $\nu w.\, D{\Downarrow_x}$ so by Proposition 29 $D{\Downarrow_x}$.

   For any $(z := !x@y) \sqsubseteq E \sqsupseteq D$, we can find F such that:

   $$E \equiv (F, z := !x@y) \tag{75}$$

   Then $\nu w.\, (F, z := !x@y) \sqsupseteq \nu w.\, D$, so for fresh u and v:

   $$\begin{array}{lr} \models E : (y : \psi) & \\ \quad \Rightarrow (F, z := !x@y) : (y : \psi) & \text{(Eqn 75)} \\ \quad \Rightarrow \models (F, z := !x@y, u := !\nabla y, v := !x@u) : (y : \psi) & \text{(Propn 60)} \\ \quad \Rightarrow \models (F, z := !x@y, u := !\nabla y, v := !x@u) : (u : \psi) & \text{(Propn 61.3)} \\ \quad \Rightarrow \models \nu w.\, (F, z := !x@y, u := !\nabla y, v := !x@u) : (u : \psi) & \text{(Indn)} \\ \quad \Rightarrow \models (\nu w.\, (F, z := !x@y), u := !\nabla y, v := !x@u) : (u : \psi) & (\nu\text{MIG}) \\ \quad \Rightarrow \models (\nu w.\, (F, z := !x@y), u := !\nabla y, v := !x@u) : (v : \chi) & \text{(Eqn 74)} \\ \quad \Rightarrow \models \nu w.\, (F, z := !x@y, u := !\nabla y, v := !x@u) : (v : \chi) & (\nu\text{MIG}) \\ \quad \Rightarrow \models (F, z := !x@y, u := !\nabla y, v := !x@u) : (v : \chi) & \text{(Indn)} \\ \quad \Rightarrow \models (F, z := !x@y, u := !\nabla y, v := !x@y) : (v : \chi) & \text{(Propn 61.6)} \\ \quad \Rightarrow \models (F, z := !x@y, u := !\nabla y, v := !x@y) : (z : \chi) & \text{(Propn 61.1)} \\ \quad \Rightarrow \models (F, z := !x@y) : (z : \chi) & \text{(Propn 60)} \\ \quad \Rightarrow \models E : (z : \chi) & \text{(Eqn 75)} \end{array}$$

   Thus $\models D : (x : \psi \to \chi)$.

2. An induction on Δ, using part 1.

3. An induction on Δ, using part 1. □

PROPOSITION 64.

1. *If* $\vdash \phi \leq \psi$ *and* $\models D : (x : \phi)$ *then* $\models D : (x : \psi)$.
2. *If* $\vdash \Gamma \leq \Delta$ *and* $\models D : \Gamma$ *then* $\models D : \Delta$.
3. *If* $\Gamma \leq \Gamma'$, $\Gamma' \models D : \Delta'$ *and* $\vdash \Delta' \leq \Delta$ *then* $\Gamma \models D : \Delta$.
4. *If* $\Gamma \leq \Delta$, $\Delta \models M : \phi$ *and* $\vdash \phi \leq \psi$ *then* $\Gamma \models M : \psi$.

PROOF.

1. An induction on the proof of $\vdash \phi \leq \psi$.
2. An induction on the proof of $\models D : \Gamma$.
3. Follows from part 2.
4. Follows from part 2. □

PROPOSITION 65. *For closed D:*

1. *If* $D \equiv E$ *then* $\models D : \Delta$ *iff* $\models E : \Delta$.
2. *If* $D \sqsubseteq E$ *and* $\models D : \Delta$ *then* $\models E : \Delta$.
3. *If* $D \to_c E$ *then* $\models D : \Delta$ *iff* $\models E : \Delta$.

Thus, if $E \to_c^* F$ *then* $\Gamma \models E : \Delta$ *iff* $\Gamma \models F : \Delta$.

PROOF.

1. A simple induction on Δ.

2. Follows from Propositions 60 and 63.

3. Show by induction on ϕ that $D : (x : \phi)$ iff $E : (x : \phi)$. The only difficult case is when $\phi = \psi \to \chi$. We then have two cases, depending on the axiom used in proving $D \to_c E$:

   (BUILD). We shall show that if $D : (x : \psi \to \chi)$ then $E : (x : \psi \to \chi)$ since the other direction is easier.

   By Proposition 25.2 we can find ν-less F such that:

   $$\begin{aligned} D &\equiv \nu \vec{x}.\, (F, w := !\,\mathsf{rec}\, G\, \mathsf{in}\, M) \\ E &\equiv \nu \vec{x}\vec{y}.\, (F, G, w := !M) \\ \vec{y} &= \mathsf{wv}\, G \end{aligned}$$

   For simplicity, we assume G is ν-less, and that the declaration (F, G) is well-formed, as the general case is no more interesting. Then for any $(z := !x@y) \sqsubseteq H \sqsupseteq E$:

   - If $w = z$ then $D{\Uparrow_x}$, which is a contradiction.

- If $w = x$ then we can find fresh $\vec{y}$ and I such that:

$$H \equiv \nu\vec{x}\vec{y}.(F, G, I, w := !M, z := !w@y) \tag{76}$$

so let $\vec{w} = \mathsf{wv}\, G$, and let v and $\vec{v}$ be fresh. Then since $\models D : (x : \psi \to \chi)$, by Proposition 59:

$$\nu\vec{x}.(F, v := !\,\mathsf{rec}\, G \,\mathsf{in}\, M)[v/w] : (v : \psi \to \chi) \tag{77}$$

and, from the definition of $\sqsubseteq$:

$$\begin{array}{l}(z := !v@y) \\ \quad \sqsubseteq \nu\vec{x}.(F[v/w], G, I, v := !(\mathsf{rec}\, G \,\mathsf{in}\, M)[v/w], \\ \qquad w := !M[v/w], z := !v@y) \\ \quad \sqsupseteq \nu\vec{x}.(F, v := !\,\mathsf{rec}\, G \,\mathsf{in}\, M)[v/w]\end{array} \tag{78}$$

Then:

$\models H : (y : \psi)$
$\Rightarrow \models \nu\vec{x}\vec{y}.(F, G, I, w := !M, z := !w@y) : (y : \psi)$ (Eqn 76)
$\Rightarrow \models (F, G, I, w := !M, z := !w@y) : (y : \psi)$ (Propn 63)
$\Rightarrow \models (F, G, I, [v\vec{v}/w\vec{w}]G[v\vec{v}/w\vec{w}],$
$v := !M, w := !M, z := !w@y) : (y : \psi)$ (Propn 60)
$\Rightarrow \models (F[v/w], G, I, [v\vec{v}/w\vec{w}]G[v\vec{v}/w\vec{w}],$
$v := !M[v/w], w := !M[v/w], z := !v@y) : (y : \psi)$ (Propn 61.5)
$\Rightarrow \models (F[v/w], G, I, v := !(\mathsf{rec}\, G \,\mathsf{in}\, M)[v/w],$
$w := !M[v/w], z := !v@y) : (y : \psi)$ (Indn)
$\Rightarrow \models (F[v/w], G, I, v := !(\mathsf{rec}\, G \,\mathsf{in}\, M)[v/w],$
$w := !M[v/w], z := !v@y) : (z : \chi)$ (Eqns 77 and 78)
$\Rightarrow \models H : (z : \chi)$ (Similarly)

Thus $\models E : (x : \psi \to \chi)$.

- If $x \neq w \neq z$ then the proof is similar.

(OTHER) If $D \to_c E$ is proved without (BUILD) then we can show that:

$$D \sqsubseteq D' \text{ implies } D' \to_c E' \sqsupseteq E$$
$$E \sqsubseteq E' \text{ implies } D \sqsubseteq D' \to_c E'$$

Then if $\models D : (x : \psi \to \chi)$ then $D{\Downarrow_x}$ so by Proposition 32, $E{\Downarrow_x}$. Then for any $(z := !x@y) \sqsubseteq F \sqsupseteq E$, we can find G such that:

$$F \equiv (G, z := !x@y) \tag{79}$$

Then let w be fresh, so:

$$(w := !x@y) \sqsubseteq (G, w := !x@y, z := !x@y) \sqsupseteq E$$

and we can find $H \sqsupseteq D$ such that:

$$H \to_c F$$

Then:

$\models F : (y : \psi)$
$\Rightarrow \models (G, z := !x@y) : (y : \psi)$ (Eqn 79)
$\Rightarrow \models (G, z := !x@y, w := !x@y) : (y : \psi)$ (Propn 60)
$\Rightarrow \models (H, w := !x@y) : (y : \psi)$ (Indn)
$\Rightarrow \models (H, w := !x@y) : (w : \chi)$ $(\models D : (x : \psi \to \chi))$
$\Rightarrow \models (G, z := !x@y, w := !x@y) : (w : \chi)$ (Indn)
$\Rightarrow \models (G, z := !x@y, w := !x@y) : (z : \chi)$ (Propn 61.1)
$\Rightarrow \models (G, z := !x@y) : (z : \chi)$ (Propn 60)
$\Rightarrow \models F : (z : \chi)$ (Eqn 79)

Thus for any $(z := !x@y) \sqsubseteq F \sqsupseteq E$:

$$\models F : (y : \psi) \Rightarrow \models F : (z : \chi)$$

so $\models E : (x : \psi \to \chi)$.

The other direction is shown similarly. □

3.11 Full abstraction

In this section, we show that the model **D** is fully abstract for concurrent graph reduction. This means that concurrent graph reduction has the same fully abstract model as leftmost-outermost reduction, and so concurrent graph reduction has exactly the same computational power as leftmost-outermost reduction.

This proof follows the same structure as Section 2.7

- We show that $\Gamma \vdash D : \Delta$ iff $[\![\Delta]\!] \leq [\![D]\!][\![\Gamma]\!]$, thus showing that the proof system is sound and complete for the denotational semantics. This is Proposition 66, the graph reduction equivalent of Proposition 15.
- We then show that if $\Gamma \vdash D : \Delta$ then $\Gamma \models D : \Delta$, and that if $\Gamma \models D : \Delta$ then $[\![\Delta]\!] \leq [\![D]\!][\![\Gamma]\!]$. Thus the three presentations of the logic are equivalent. This is Proposition 69, the graph reduction equivalent of Proposition 18.
- Finally, we show that full abstraction is gained by proving the three logical presentations to be equivalent. This is Proposition 70, the graph reduction equivalent of Proposition 19.

Thus, ABRAMSKY and ONG's techniques can be adapted to graph reduction.

PROPOSITION 66.

1. $\Gamma \vdash M : \phi$ *iff* $[\![\phi]\!] \leq [\![M]\!][\![\Gamma]\!]$.
2. $\Gamma \vdash D : \Delta$ *iff* $[\![\Delta]\!] \leq [\![D]\!][\![\Gamma]\!]$.

PROOF.

SOUNDNESS ($\Rightarrow$) We have to prove the rules of $\Gamma \vdash M : \phi$ and $\Gamma \vdash D : \Delta$ be sound. For example, to prove (!), if $[\![\Delta]\!] \leq [\![x := !M]\!][\![\Gamma]\!]$ and $[\![\phi]\!] \leq [\![M]\!][\![\Delta]\!]$ then:

$$\begin{array}{ll}
[\![x : \phi]\!] & \\
\quad \leq (x := [\![M]\!])[\![\Delta]\!] & \text{(Hypothesis)} \\
\quad \leq (x := [\![M]\!])([\![x := !M]\!][\![\Gamma]\!]) & \text{(Hypothesis)} \\
\quad = [\![x := !M]\!][\![\Gamma]\!] & \text{(Propn 57.2)}
\end{array}$$

The other cases are similar.

COMPLETENESS ($\Leftarrow$) An induction on M and D. For example, if $x \neq y$ and:

$$[\![\phi]\!] \leq [\![x@y]\!][\![\Gamma]\!]$$

then either $[\![\phi]\!] = \bot$, so $\vdash \phi = \omega$ and so $\Gamma \vdash x@y : \phi$, or:

$$\begin{array}{ll}
[\![\phi]\!] \leq [\![x@y]\!][\![\Gamma]\!] & \\
\quad \Rightarrow [\![\phi]\!] \leq \mathsf{apply}[\![\Gamma(x)]\!][\![\Gamma(y)]\!] & \text{(Defn of } [\![x@y]\!]) \\
\quad \Rightarrow [\![\Gamma(y) \to \phi]\!] \leq [\![\Gamma(x)]\!] & \text{(Propn 9.1)} \\
\quad \Rightarrow \vdash \Gamma(x) \leq \Gamma(y) \to \phi & \text{(Propn 12)} \\
\quad \Rightarrow \vdash \Gamma \leq x : \Gamma(y) \to \phi, y : \Gamma(y) & \text{(Defn of } \leq) \\
\quad \Rightarrow \Gamma \vdash x@y : \phi & ((\leq) \text{ and } (@a))
\end{array}$$

The proofs for $x@x$, ∇x, $x \vee y$ and $x \vee x$ are similar. The proof for $\lambda x \,.\, M$ is similar to that of Proposition 15. If:

$$[\![\phi]\!] \leq [\![\mathsf{rec} D \,\mathsf{in}\, M]\!][\![\Gamma]\!] \tag{80}$$

then:

$$\begin{array}{ll}
[\![\phi]\!] & \\
\quad \leq [\![\mathsf{rec} D \,\mathsf{in}\, M]\!][\![\Gamma]\!] & \text{(Eqn 80)} \\
\quad = [\![M]\!]([\![D]\!][\![\Gamma]\!]) & \text{(Defn of } [\![\mathsf{rec} D \,\mathsf{in}\, M]\!]) \\
\quad = [\![M]\!](\bigvee\{[\![\Delta]\!] \mid [\![\Delta]\!] \leq [\![D]\!][\![\Gamma]\!]\}) & \text{(Propn 14)} \\
\quad = \bigvee\{[\![M]\!][\![\Delta]\!] \mid [\![\Delta]\!] \leq [\![D]\!][\![\Gamma]\!]\}) & \text{(Continuity)}
\end{array}$$

so since $[\![\phi]\!]$ is compact we can find a Δ such that:

$$[\![\phi]\!] \leq [\![M]\!][\![\Delta]\!] \qquad [\![\Delta]\!] \leq [\![D]\!][\![\Gamma]\!]$$

so by induction:

$$\Delta \vdash M : \phi \qquad \Gamma \vdash D : \Delta$$

and so by (rec):

$$\Gamma \vdash \mathsf{rec} D \,\mathsf{in}\, M : \phi$$

If $[\![\Delta]\!] \leq [\![(x := !M)]\!][\![\Gamma]\!]$ then:

$$\begin{array}{ll}
[\![\Delta]\!] & \\
\quad \leq [\![(x := !M)]\!][\![\Gamma]\!] & \text{(Hypothesis)} \\
\quad = \mathsf{fix}(\mathsf{set}\{x\}(x := [\![M]\!]))[\![\Gamma]\!] & \text{(Defn of } [\![x := !M]\!]) \\
\quad = \bigvee\{(\mathsf{set}\{x\}(x := [\![M]\!]))^n \bot \mid n \text{ in } \omega\}[\![\Gamma]\!] & \text{(Defn of fix)} \\
\quad = \bigvee\{(\mathsf{set}\{x\}(x := [\![M]\!]))^n \bot [\![\Gamma]\!] \mid n \text{ in } \omega\} & \text{(Continuity)}
\end{array}$$

so since $[\![\Delta]\!]$ is compact, we can find an n such that:

$$[\![\Delta]\!] \leq (\mathsf{set}\{x\}(x := [\![M]\!]))^n \bot [\![\Gamma]\!]$$

then we can show by induction on n that $\Gamma \vdash (x := !M) : \Delta$:

- If $[\![\Delta]\!] \leq (\mathsf{set}\{x\}(x := [\![M]\!]))^0 \bot [\![\Gamma]\!]$ then $[\![\Delta]\!] = \bot$ so by Proposition 12 $\vdash \Delta = \varepsilon$, so by $(\leq)$ and $(\bot)$ $\Gamma \vdash (x := !M) : \Delta$.
- If $[\![\Delta]\!] \leq (\mathsf{set}\{x\}(x := [\![M]\!]))^{n+1} \bot [\![\Gamma]\!]$ then:

  $$\begin{array}{ll}
  [\![\Delta(x)]\!] & \\
  \quad \leq (\mathsf{set}\{x\}(x := [\![M]\!]))^{n+1} \bot [\![\Gamma]\!]x & \text{(Hypothesis)} \\
  \quad = [\![M]\!]((\mathsf{set}\{x\}(x := [\![M]\!]))^n \bot [\![\Gamma]\!]) & \text{(Defn of set)} \\
  \quad = [\![M]\!](\bigvee\{[\![\Theta]\!] \mid [\![\Theta]\!] \leq (\mathsf{set}\{x\}(x := [\![M]\!]))^n \bot [\![\Gamma]\!]\}) & \text{(Propn 14)} \\
  \quad = \bigvee\{[\![M]\!][\![\Theta]\!] \mid [\![\Theta]\!] \leq (\mathsf{set}\{x\}(x := [\![M]\!]))^n \bot [\![\Gamma]\!]\} & \text{(Continuity)}
  \end{array}$$

  so since $[\![\Delta(x)]\!]$ is compact we can find a Θ such that:

  $$[\![\Delta(x)]\!] \leq [\![M]\!][\![\Theta]\!] \qquad [\![\Theta]\!] \leq (\mathsf{set}\{x\}(x := [\![M]\!]))^n \bot [\![\Gamma]\!]$$

  so by induction:

  $$\Theta \vdash M : \Delta(x) \qquad \Gamma \vdash (x := !M) : \Theta$$

  and so by (!):

  $$\Gamma \vdash (x := !M) : (x : \Delta(x))$$

  For any $y \neq x$:

  $$\begin{array}{ll}
  \mathsf{true} & \\
  \quad \Rightarrow [\![\Delta(y)]\!] = [\![\Delta]\!]y & \text{(Defn of } [\![\Delta]\!]) \\
  \quad \Rightarrow [\![\Delta(y)]\!] \leq (\mathsf{set}\{x\}(x := [\![M]\!]))^{n+1} \bot [\![\Gamma]\!]y & \text{(Hypothesis)} \\
  \quad \Rightarrow [\![\Delta(y)]\!] \leq [\![\Gamma]\!]y & \text{(Defn of set)} \\
  \quad \Rightarrow [\![\Delta(y)]\!] \leq [\![\Gamma(y)]\!] & \text{(Defn of } [\![\Gamma]\!]) \\
  \quad \Rightarrow \vdash \Gamma(y) \leq \Delta(y) & \text{(Propn 12)} \\
  \quad \Rightarrow \vdash \nu x \,.\, \Gamma \leq y : \Delta(y) & \text{(Defn of } \leq) \\
  \quad \Rightarrow \Gamma \vdash (x := !M) : (y : \Delta(y)) & ((\bot) \text{ and } (\leq))
  \end{array}$$

  Thus $\forall y \,.\, \Gamma \vdash (x := !M) : (y : \Delta(y))$ and so $\Gamma \vdash (x := !M) : \Delta$.

The proofs for $x := ?M$ and D, E are similar, and the proof for $\nu x \,.\, D$ is much simpler. $\square$

PROPOSITION 67. *If $D \to E$ then $[\![D]\!] = [\![E]\!]$.*

PROOF. This is a matter of proving each of the axioms for $\equiv$ and $\mapsto$ to be sound. The axioms for $\equiv$ are covered in Propositions 57 and 58.

Of the axioms for $\mapsto$, the axiom for graph building can be shown from Proposition 58. The axioms for spine traversal are simple since $[\![y := ?M]\!] = [\![y := !M]\!]$. Garbage collection is equally simple, from the definition of new. This leaves the axioms for updating which all have similar proofs, so we shall consider the case of an indirection node. For any $z \neq x$:

$$\begin{aligned}
&\mathsf{read}\, z \circ (x := \mathsf{read}\, y) \circ [\![y := !M]\!] \\
&\quad = \mathsf{read}\, z \circ [\![y := !M]\!] && \text{(Propn 54.4)} \\
&\quad = \mathsf{read}\, z \circ (x := [\![M]\!]) \circ [\![y := !M]\!] && \text{(Propn 54.4)}
\end{aligned}$$

and:

$$\begin{aligned}
&\mathsf{read}\, x \circ (x := \mathsf{read}\, y) \circ [\![y := !M]\!] \\
&\quad = \mathsf{read}\, y \circ [\![y := !M]\!] && \text{(Propn 54.3)} \\
&\quad = \mathsf{read}\, y \circ (y := [\![M]\!]) \circ [\![y := !M]\!] && \text{(Propn 57.2)} \\
&\quad = [\![M]\!] \circ [\![y := !M]\!] && \text{(Propn 54.3)} \\
&\quad = \mathsf{read}\, x \circ (x := [\![M]\!]) \circ [\![y := !M]\!] && \text{(Propn 54.3)}
\end{aligned}$$

Thus for any z:

$$\mathsf{read}\, z \circ (x := \mathsf{read}\, y) \circ [\![y := !M]\!] = \mathsf{read}\, z \circ (x := [\![M]\!]) \circ [\![y := !M]\!]$$

and so:

$$(x := \mathsf{read}\, y) \circ [\![y := !M]\!] = (x := [\![M]\!]) \circ [\![y := !M]\!] \qquad (81)$$

Then:

$$\begin{aligned}
&[\![x := !\nabla y, y := !\lambda w.\, M]\!] \\
&\quad = \mathsf{fix}(\mathsf{set}\{x,y\}(\mathsf{fix}(\mathsf{set}\{x\}(x := \mathsf{read}\, y)) \circ [\![y := !\lambda w.\, M]\!])) && \text{(Defn of } [\![M]\!]) \\
&\quad = \mathsf{fix}(\mathsf{set}\{x,y\}(\mathsf{fix}(\mathsf{set}\{x\}(x := \mathsf{read}\, y)) \\
&\qquad \circ \mathsf{fix}(\mathsf{set}\{y\}[\![y := !\lambda w.\, M]\!]))) && \text{(Propn 57.6)} \\
&\quad = \mathsf{fix}(\mathsf{set}\{x,y\}(x := \mathsf{read}\, y \circ [\![y := !\lambda w.\, M]\!])) && \text{(Propn 57.5)} \\
&\quad = \mathsf{fix}(\mathsf{set}\{x,y\}(x := [\![\lambda w.\, M]\!] \circ [\![y := !\lambda w.\, M]\!])) && \text{(Eqn 81)} \\
&\quad = \mathsf{fix}(\mathsf{set}\{x,y\}(\mathsf{fix}(\mathsf{set}\{x\}(x := [\![\lambda w.\, M]\!])) \\
&\qquad \circ \mathsf{fix}(\mathsf{set}\{y\}[\![y := !\lambda w.\, M]\!]))) && \text{(Propn 57.5)} \\
&\quad = \mathsf{fix}(\mathsf{set}\{x,y\}(\mathsf{fix}(\mathsf{set}\{x\}(x := [\![\lambda w.\, M]\!])) \circ [\![y := !\lambda w.\, M]\!])) && \text{(Propn 57.6)} \\
&\quad = [\![x := !\lambda w.\, M, y := !\lambda w.\, M]\!] && \text{(Defn of } [\![M]\!])
\end{aligned}$$

The other cases are similar. □

COROLLARY 68. *If $D \Downarrow_x$ then $[\![D]\!]\sigma x \neq \bot$.*

PROPOSITION 69.

- $(\Gamma \vdash M : \phi) \Rightarrow (\Gamma \models M : \phi) \Rightarrow ([\![\phi]\!] \leq [\![M]\!][\![\Gamma]\!])$.

- $(\Gamma \vdash D : \Delta) \Rightarrow (\Gamma \models D : \Delta) \Rightarrow ([\![\Delta]\!] \leq [\![D]\!][\![\Gamma]\!])$.

PROOF.

SOUNDNESS ($1 \Rightarrow 2$) This is a matter of proving each of the rules of $\Gamma \vdash M : \phi$ and $\Gamma \vdash D : \Delta$ sound.

($\bot$) For any D and z, if $\models (D, z := !M) : \varepsilon$ then by Proposition 64 we know that $\models (D, z := !M) : (z : \omega)$. Thus $\models M : \omega$.

(ID) For any D and z, if $\models (D, z := !\nabla x) : (x : \phi)$ then by Proposition 61.4, $\models (D, z := !\nabla x) : (z : \phi)$. Thus $x : \phi \models \nabla x : \phi$.

($\to$E) For any D and z, if $\models (D, z := !x@y) : (x : \phi \to \psi \wedge y : \phi)$ then $\models (D, z := !x@y) : (x : \phi \to \psi)$ and $\models (D, z := !x@y) : (y : \phi)$, so $\models (D, z := !x@y) : (z : \psi)$. Thus $x : \phi \to \psi \wedge y : \phi \models x@y : \psi$.

($\vee a$) For any D and z, assume $(D, z := !x \vee y) : (x : \gamma)$, so:

$$\mathsf{tag}_x(D, z := !x \vee y) \to_c^* E \qquad (82)$$

and x is in whnf in E, so:

$$\begin{aligned}
&(D, z := !x \vee y) \\
&\quad \to_c^{\leq 1} \mathsf{tag}_x(D, z := !x \vee y) && (\vee\text{TRAV}) \\
&\quad \to_c^* E && \text{(Eqn 82)}
\end{aligned}$$

and by Propositions 30.4 and 30.5 either $E \equiv (F, z := !\,\mathsf{I})$, or $E \equiv (F, z := !x \vee y)$ and since x is in whnf in E, $E \to_c (F, z := !\,\mathsf{I})$. Thus:

$$(D, z := !x \vee y) \to_c^* (F, z := !\,\mathsf{I})$$

and:

$$\models (F, z := !\,\mathsf{I}) : (z : \phi \to \phi)$$

so by Proposition 65:

$$\models (D, z := !x \vee y) : (z : \phi \to \phi)$$

Thus $(x : \gamma) \models (x \vee y : \phi \to \phi)$.

($\vee b$) $y : \gamma \models x \vee y : \phi \to \phi$ is proved similarly.

($\wedge$) Assume $\Gamma \models M : \phi$ and $\Gamma \models M : \psi$.
Then for any D and z, if $\models (D, z := !M) : \Gamma$
then $\models (D, z := !M) : (z : \phi)$ and $\models (D, z := !M) : (z : \psi)$
so $\models (D, z := !M) : (z : \phi \wedge \psi)$. Thus $\Gamma \models M : (\phi \wedge \psi)$.

($\leq$) Follows from Proposition 64.

($\to$) Assume $\Gamma, w : \phi \models M : \psi$. Then for any D and x, assume $\models (D, x := !\lambda w.\, M) : \Gamma$. Then $(D, x := !\lambda w.\, M) \Downarrow_x$, and for any

$(z := !x@y) \sqsubseteq E \sqsupseteq (D, x := !\lambda w.\, M)$, by Proposition 22 we can find F such that:

$$(F, x := !\lambda w.\, M, z := !x@y) \equiv E \tag{83}$$

so:

$$\begin{array}{lll} \models (D, x := !\lambda w.\, M) : \Gamma & & \\ \quad \Rightarrow \models E : \Gamma & & \text{(Propn 60)} \\ \quad \Rightarrow \models (F, x := !\lambda w.\, M, z := !x@y) : \Gamma & & \text{(Eqn 83)} \\ \quad \Rightarrow \models (F, x := !\lambda w.\, M, z := !M[y/w]) : \Gamma & & (@\text{UPD}) \end{array}$$

Thus:

$$\begin{array}{lll} \models E : (y : \phi) & & \\ \quad \Rightarrow \models (F, x := !\lambda w.\, M, z := !x@y) : (y : \phi) & & \text{(Eqn 83)} \\ \quad \Rightarrow \models (F, x := !\lambda w.\, M, z := !M[y/w]) : (y : \phi) & & (@\text{UPD}) \\ \quad \Rightarrow \models (F, x := !\lambda w.\, M, z := !M[y/w]) : \Gamma \wedge (y : \phi) & & (\wedge) \\ \quad \Rightarrow \models (F, x := !\lambda w.\, M, z := !M[y/w]) : (z : \psi) & & \text{(Propn 62)} \\ \quad \Rightarrow \models (F, x := !\lambda w.\, M, z := !x@y) : (z : \psi) & & (@\text{UPD}) \\ \quad \Rightarrow \models E : (z : \psi) & & \text{(Eqn 83)} \end{array}$$

Thus $(D, x := !\lambda w.\, M) : (x : \phi \to \psi)$, and so $\Gamma \models \lambda w.\, M : \phi \to \psi$.

(rec) Assume $\Gamma \models D : \Delta$ and $\Delta \models M : \phi$. Let $\vec{x} = \mathsf{wv}\, D$ and let $\vec{y}$ be fresh. Then by Proposition 59:

$$\Gamma \models [\vec{y}/\vec{x}]D[\vec{y}/\vec{x}] : [\vec{y}/\vec{x}]\Delta \tag{84}$$

$$[\vec{y}/\vec{x}]\Delta \models M[\vec{y}/\vec{x}] : \phi \tag{85}$$

so for any E and z:

$$\begin{array}{lll} \models E, z := !(\mathsf{rec}\, D \,\mathsf{in}\, M) : \Gamma & & \\ \quad \Rightarrow \models E, \mathsf{local}\, D \,\mathsf{in}\, z := !M : \Gamma & & (\text{BUILD}) \\ \quad \Rightarrow \models E, \nu\vec{y}.\, (z := !M[\vec{y}/\vec{x}], [\vec{y}/\vec{x}]D[\vec{y}/\vec{x}]) : \Gamma & & \text{(Defn of local)} \\ \quad \Rightarrow \models \nu\vec{y}.\, (E, z := !M[\vec{y}/\vec{x}], [\vec{y}/\vec{x}]D[\vec{y}/\vec{x}]) : \Gamma & & (\nu\text{MIG}) \\ \quad \Rightarrow \models (E, z := !M[\vec{y}/\vec{x}], [\vec{y}/\vec{x}]D[\vec{y}/\vec{x}]) : \Gamma & & \text{(Propn 63)} \\ \quad \Rightarrow \models (E, z := !M[\vec{y}/\vec{x}], [\vec{y}/\vec{x}]D[\vec{y}/\vec{x}]) : [\vec{y}/\vec{x}]\Delta & & \text{(Eqn 84)} \\ \quad \Rightarrow \models (E, z := !M[\vec{y}/\vec{x}], [\vec{y}/\vec{x}]D[\vec{y}/\vec{x}]) : (z : \phi) & & \text{(Eqn 85)} \\ \quad \Rightarrow \models \nu\vec{y}.\, (E, z := !M[\vec{y}/\vec{x}], [\vec{y}/\vec{x}]D[\vec{y}/\vec{x}]) : (z : \phi) & & \text{(Propn 63)} \\ \quad \Rightarrow \models E, \nu\vec{y}.\, (z := !M[\vec{y}/\vec{x}], [\vec{y}/\vec{x}]D[\vec{y}/\vec{x}]) : (z : \phi) & & (\nu\text{MIG}) \\ \quad \Rightarrow \models E, \mathsf{local}\, D \,\mathsf{in}\, z := !M : (z : \phi) & & \text{(Defn of local)} \\ \quad \Rightarrow \models E, z := !\,\mathsf{rec}\, D \,\mathsf{in}\, M : (z : \phi) & & (\text{BUILD}) \end{array}$$

Thus $\Gamma \models \mathsf{rec}\, D \,\mathsf{in}\, M : \phi$.

($\perp$) For any E, if $\models D, E : \nu(\mathsf{wv}\, D)\,.\,\Gamma$ then $\models D, E : \nu(\mathsf{wv}\, D)\,.\,\Gamma$. Thus $\Gamma \models D : \nu(\mathsf{wv}\, D)\,.\,\Gamma$.

($\wedge$) Assume $\Gamma \models D : \Delta$ and $\Gamma \models D : \Theta$. Then for any E, if $\models D, E : \nu(\mathsf{wv}\, D)\,.\,\Gamma$ then $\models D, E : \Delta$ and $\models D, E : \Theta$, so $\models D, E : \Delta \wedge \Theta$. Thus $\Gamma \models D : \Delta \wedge \Theta$.

($\leq$) Follows from Proposition 64.

(!) Assume $\Gamma \models (x := !M) : \Delta$ and $\Delta \models M : \phi$. Then for any E, if $\models (x := !M), E : \nu x\,.\,\Gamma$ then $\models (x := !M), E : \Delta$ so $\models (x := !M), E : \Theta$. Thus $\Gamma \models (x := !M) : \Theta$.

(?) Follows from (!) and Proposition 61.7.

(L) Assume $\Gamma \models D, E : \Delta$ and $\Delta \models D : \Theta$. Then for any F, if $\models D, E, F : \nu(\mathsf{wv}(D, E))\,.\,\Gamma$ then $\models D, E, F : \Delta$ so $\models D, E, F : \Theta$. Thus $\Gamma \models D, E : \Theta$.

(R) Assume $\Gamma \models D, E : \Delta$ and $\Delta \models D : \Theta$. Then $\Gamma \models D, E : \Theta$ follows similarly.

(ν) Assume $\nu x\,.\,\Gamma \models D : \Delta$. Then for any fresh y, by Proposition 59:

$$\nu x\,.\,\Gamma \models [y/x]D[y/x] : [y/x]\Delta \tag{86}$$

Then for any E and fresh y:

$$\begin{array}{lll} \models (\nu x\,.\,D), E : \nu(\mathsf{wv}(\nu x\,.\,D))\,.\,\Gamma & & \\ \quad \Rightarrow \models (\nu y\,.\,[y/x]D[y/x]), E : \nu(\mathsf{wv}(\nu x\,.\,D))\,.\,\Gamma & & (\alpha) \\ \quad \Rightarrow \models \nu y\,.\,([y/x]D[y/x], E) : \nu(\mathsf{wv}(\nu x\,.\,D))\,.\,\Gamma & & (\nu\text{MIG}) \\ \quad \Rightarrow \models [y/x]D[y/x], E : \nu(\mathsf{wv}(\nu x\,.\,D))\,.\,\Gamma & & \text{(Propn 63)} \\ \quad \Rightarrow \models [y/x]D[y/x], E : \nu(\mathsf{wv}([y/x]D[y/x]))\,.\,\nu x\,.\,\Gamma & & \text{(Defn of } \nu x\,.\,\Gamma) \\ \quad \Rightarrow \models [y/x]D[y/x], E : [y/x]\Delta & & \text{(Eqn 86)} \\ \quad \Rightarrow \models \nu y\,.\,([y/x]D[y/x], E) : \nu y\,.\,[y/x]\Delta & & \text{(Propn 63)} \\ \quad \Rightarrow \models (\nu y\,.\,[y/x]D[y/x]), E : \nu y\,.\,[y/x]\Delta & & (\nu\text{MIG}) \\ \quad \Rightarrow \models (\nu x\,.\,D), E : \nu x\,.\,\Delta & & (\alpha) \end{array}$$

Thus $\Gamma \models \nu x\,.\,D : \nu x\,.\,\Delta$.

Thus the proof system is operationally consistent.

COMPLETENESS (2 $\Rightarrow$ 3) We first show by induction on Δ that if D is closed and $\models D : \Delta$ then $[\![\Delta]\!] \leq [\![D]\!]\sigma$.

- If $\models D : \varepsilon$ then $[\![\varepsilon]\!] = \perp \leq [\![D]\!]\sigma$.
- If $\models D : \Delta, \Gamma$ then $\models D : \Delta$ and $\models D : \Gamma$, so by induction $[\![\Gamma]\!] \leq [\![D]\!]\sigma$ and $[\![\Delta]\!] \leq [\![D]\!]\sigma$, so $[\![\Gamma, \Delta]\!] = [\![\Gamma]\!] \vee [\![\Delta]\!] \leq [\![D]\!]\sigma$.
- If $\models D : (x : \omega)$ then $[\![x : \omega]\!] = \perp \leq [\![D]\!]\sigma$.
- If $\models D : (x : \phi \wedge \psi)$ then $\models D : (x : \phi)$ and $\models D : (x : \psi)$ so by induction $[\![x : \phi]\!] \leq [\![D]\!]\sigma$ and $[\![x : \psi]\!] \leq [\![D]\!]\sigma$, so $[\![x : \phi \wedge \psi]\!] \leq [\![D]\!]\sigma$.

- If $\models D : (x : \phi \to \psi)$ then $D{\Downarrow_x}$ so by Corollary 68 $[\![D]\!]\sigma x \neq \bot$. Also, for fresh y and z:

$$
\begin{array}{ll}
\mathsf{true} & \\
\quad \Rightarrow [\![y : \phi]\!] = [\![D_{y:\phi}]\!]\bot & \text{(Defn of } D_\Delta\text{)} \\
\quad \Rightarrow \vdash D_{y:\phi} : (y : \phi) & \text{(Propn 66)} \\
\quad \Rightarrow \vdash D, D_{y:\phi}, z := !x@y : (y : \phi) & (\text{(L) and (R)}) \\
\quad \Rightarrow \models D, D_{y:\phi}, z := !x@y : (y : \phi) & \text{(Soundness)} \\
\quad \Rightarrow \models D, D_{y:\phi}, z := !x@y : (z : \psi) & \text{(Defn of } \models\text{)} \\
\quad \Rightarrow [\![z : \psi]\!] \leq [\![D, D_{y:\phi}, z := !x@y]\!]\sigma & \text{(Induction)} \\
\quad \Rightarrow [\![\psi]\!] \leq [\![D, D_{y:\phi}, z := !x@y]\!]\sigma z & \text{(Application)} \\
\quad \Rightarrow [\![\psi]\!] \leq \mathsf{apply}([\![D]\!]\sigma x)([\![D_{y:\phi}]\!]\sigma y) & \text{(Defn of } [\![x@y]\!]\text{)} \\
\quad \Rightarrow [\![\psi]\!] \leq \mathsf{apply}([\![D]\!]\sigma x)[\![\phi]\!] & \text{(Defn of } D_\Delta\text{)} \\
\quad \Rightarrow [\![\phi \to \psi]\!] \leq [\![D]\!]\sigma x & \text{(Propn 9.1)} \\
\quad \Rightarrow [\![x : \phi \to \psi]\!] \leq [\![D]\!]\sigma & \text{(Defn of } [\![x : \phi \to \psi]\!]\text{)}
\end{array}
$$

  Thus $[\![\phi \to \psi]\!] \leq [\![D]\!]\sigma$.

Thus we have:

$$\models D : \Delta \Rightarrow [\![\Delta]\!] \leq [\![D]\!]\sigma \qquad (87)$$

If $\Gamma \models D : \Delta$ then:

$$
\begin{array}{ll}
\mathsf{true} & \\
\quad \Rightarrow [\![\nu(\mathsf{w} \nu D) \,.\, \Gamma]\!] = [\![D_{\nu(\mathsf{w}\nu D).\Gamma}]\!]\bot & \text{(Defn of } D_\Delta\text{)} \\
\quad \Rightarrow \vdash D_{\nu(\mathsf{w}\nu D).\Gamma} : \nu(\mathsf{w} \nu D) \,.\, \Gamma & \text{(Propn 66)} \\
\quad \Rightarrow \vdash D, D_{\nu(\mathsf{w}\nu D).\Gamma} : \nu(\mathsf{w} \nu D) \,.\, \Gamma & (\text{($\bot$) and (R)}) \\
\quad \Rightarrow \models D, D_{\nu(\mathsf{w}\nu D).\Gamma} : \nu(\mathsf{w} \nu D) \,.\, \Gamma & \text{(Soundness)} \\
\quad \Rightarrow \models D, D_{\nu(\mathsf{w}\nu D).\Gamma} : \Delta & \text{(Defn of } \models\text{)} \\
\quad \Rightarrow [\![\Delta]\!] \leq [\![D, D_{\nu(\mathsf{w}\nu D).\Gamma}]\!]\bot & \text{(Eqn 87)} \\
\quad \Rightarrow [\![\Delta]\!] \leq [\![D]\!][\![\Gamma]\!] & \text{(Defn of } D_\Delta\text{)}
\end{array}
$$

If $\Gamma \models M : \phi$ then for fresh z:

$$
\begin{array}{ll}
\mathsf{true} & \\
\quad \Rightarrow [\![\Gamma]\!] = [\![D_\Gamma]\!]\bot & \text{(Defn of } D_\Delta\text{)} \\
\quad \Rightarrow \vdash D_\Gamma : \Gamma & \text{(Propn 66)} \\
\quad \Rightarrow \vdash D_\Gamma, z := !M : \Gamma & \text{(L)} \\
\quad \Rightarrow \models D_\Gamma, z := !M : \Gamma & \text{(Soundness)} \\
\quad \Rightarrow \models D_\Gamma, z := !M : (z : \phi) & \text{(Defn of } \models\text{)} \\
\quad \Rightarrow [\![z : \phi]\!] \leq [\![D_\Gamma, z := !M]\!]\bot & \text{(Eqn 87)} \\
\quad \Rightarrow [\![z : \phi]\!] \leq [\![z := !M]\!][\![\Gamma]\!] & \text{(Defn of } D_\Delta\text{)} \\
\quad \Rightarrow [\![\phi]\!] \leq [\![M]\!][\![\Gamma]\!] & \text{(Application)}
\end{array}
$$

Thus the semantics is operationally complete. □

PROPOSITION 70.

1. $D \sqsubseteq_O E$ *iff* $D \sqsubseteq_S E$ *iff* $D \sqsubseteq_D E$.
2. $M \sqsubseteq_O N$ *iff* $M \sqsubseteq_S N$ *iff* $M \sqsubseteq_D N$.

PROOF. Similar to Proposition 19. □

Thus we have shown that **D** is fully abstract for concurrent graph

4 Conclusions

In this paper, we have investigated the relationship between the semantic notion of *full abstraction* and the implementation technique of *concurrent graph reduction*. We have shown that:

- Concurrent graph reduction can be given a simple operational presentation in the style of BERRY and BOUDOL's (1990) *chemical abstract machine*, and MILNER's (1991) *polyadic π-calculus*.
- The techniques of ABRAMSKY (1989) and ONG's (1988) *lazy λ-calculus* can be used to show that the fully abstract model for leftmost-outermost reduction is also fully abstract for concurrent graph reduction.
- To show full abstraction, we discussed a *confluent* reduction strategy, the relationship between *concurrent* and *sequential* reduction, and *referential transparency*. These properties are also important in implementations, and it is reassuring that showing full abstraction and writing compilers have so many issues in common.

This Chapter will discuss related work in the semantics of graph reduction, and possible future work.

4.1 Related work

In this section, we discuss some related work on the relationship between denotational or operational models of the λ-calculus. The papers described here are only those most directly related to models of concurrent graph reduction. For a discussion of models of tree reduction for the λ-calculus, see (BARENDREGT, 1984); for a discussion of implementation of graph reduction, see (PEYTON JONES, 1987).

ABRAMSKY AND ONG. This paper is based on ABRAMSKY (1989) and ONG's (1988) *Lazy λ-calculus*, which is summarized in Chapter 2.

The main difference between their approach and that outlined here is that their operational semantics is for tree reduction rather than graph reduction, and so models β-reduction by substitution rather than sharing.

In addition, ABRAMSKY and ONG investigate *applicative simulation* as an alternative characterization of the operational order. For closed terms from the λ-calculus, $M \sqsubseteq_A N$ iff:

- If $M{\Downarrow}$ then $N{\Downarrow}$.
- For any closed O, $MO \sqsubseteq_A NO$.

This can be adapted to the λ-calculus with rec as $D \models x \sqsubseteq_A y$ iff:

- If $D{\Downarrow}_x$ then $D{\Downarrow}_y$.
- For any closed E, if $(x' := !x@z, y' := !y@z) \sqsubseteq E \sqsupseteq D$ then $E \models x' \sqsubseteq_A y'$.

However, this definition does not relate directly to the proof of full abstraction, in the way that ABRAMSKY and ONG's definition does, and so was not used in Chapter 3.

BOUDOL. Another paper based on leftmost-outermost reduction of the untyped λ-calculus is BOUDOL's (1992) *λ-calculi for (strict) parallel functions*. This presents an operational and denotational semantics for the λ-calculus extended with call-by-value abstraction ($\lambda^v x\,.\,M$) and concurrency ($M \parallel N$, which we wrote as $\mathsf{M}MN$). The decision to extend the λ-calculus with with P or with $\lambda^v\,.\,M$ and $M \parallel N$ is somewhat arbitrary, since both are inter-definable:

$$\begin{aligned}\lambda^v x\,.\,M &= \lambda x\,.\,\mathsf{P}xxM\\ M \parallel N &= \mathsf{Y}(\lambda x\,.\,\lambda y\,.\,\lambda z\,.\,(\mathsf{P}yz(\lambda w\,.\,x(yw)(zw))))MN\\ \mathsf{P}MN &= ((\lambda^v x\,.\,\mathsf{I})M) \parallel ((\lambda^v x\,.\,\mathsf{I})M)\end{aligned}$$

In this paper we used P, since it has a simpler graph-reduction semantics, and corresponds very closely to AUGUSTSSON's *oracular choice* discussed below.

BOUDOL's syntax allows for declarations, $\mathsf{let}D\ \mathsf{in}\ \lambda x\,.\,M$, but his reduction rule for declarations is by substitution rather than sharing, and so he models tree reduction rather than graph reduction. Indeed, the main result of his paper is to find a fully abstract model for the strict λ-calculus with parallelism, and it is difficult to see how such a result could be applied to graph reduction, since graph reduction is usually used to evaluate non-strict languages.

ROSE. Another approach to cyclic declarations of the form $\mathsf{rec}D\ \mathsf{in}\ M$ is taken by ROSE (1993), who defines an operational semantics for the λ-calculus extended with rec. He then shows that the λ-calculus with rec is a model for the λ-calculus.

However, his semantics for declarations allows non-whnf declarations to be copied, for example (in our syntax):

$$(\mathsf{rec}\,x := !M\ \mathsf{in}\ x) \Rightarrow (\mathsf{rec}\,x := !M\ \mathsf{in}\ M)$$

Thus his operational semantics does not correspond to graph reduction. However, his techniques are useful for showing that the λ-calculus with rec is a model for the λ-calculus, and it would be interesting to see if they could be applied to a semantics with sharing. This is mentioned as being 'current work'.

WADSWORTH. The study of graph reduction began with WADSWORTHS' (1971) thesis. He presents the notion of graph reduction, and shows that graph reduction is correct for tree reduction of the untyped λ-calculus. His graphs are similar to ours, but are rooted, do not include tagging information, and do not contain re-

cursive declarations, local variables or cycles. WADSWORTH also investigates the relationship between graph reduction and the $\mathbf{D}_\infty$ model of the untyped λ-calculus (see (BARENDREGT, 1984) for more details), a topic which was later picked up by LESTER (1989) and ABRAMSKY (1989) and ONG (1988).

BARANDREGT *et al.* There is a large body of work on *term graph rewriting*, introduced by BARANDREGT *et al.* (1987), and surveyed by KENNAWAY *et al.* (1993b) and the other papers in SLEEP *et al.*'s (SLEEP *et al.*, 1993) book. Term graphs are very similar to declarations, but are rooted, and do not include tagging information, recursive declarations or local variables.

Term graphs are parameterized by a *signature* of combinators, and so model *combinator graph reduction*, that is graph reduction with a fixed set of combinators, such as S, K and I. Combinator graph reduction was used by TURNER (1979, 1985) in the implementation of SASL and Miranda.

Since term graphs do not have a fixed signature, they allow for more general reduction strategies than ours. In particular they allow for a natural presentation of type constructors and deconstructors.

However, since term graphs are so general, it is difficult to find denotational models for term graph reduction. BARANDREGT *et al.* (1987) and KENNAWAY *et al.* (1993a) show that term graph reduction is adequate for term tree reduction, but it is not obvious whether more abstract models for term graph reduction be developed.

LESTER. After WADSWORTH's thesis, one of the first papers to investigate denotational semantics for graph reduction was LESTER's (1989). He presents a typed λ-calculus, and gives it three semantics:

- A denotational semantics based on STOY's (1977) semantics for a typed λ-calculus.
- An abstract operational semantics for graph reduction using *digraphs*, which are very similar to our declarations.
- A concrete operational semantics for graph reduction based on JOHNSSON's (1984) *G*-machine.

He then shows that the denotational semantics is correct for the abstract operational semantics, which is in turn correct for the concrete operational semantics. It is possible that the same techniques could be applied to our work, to find a fully abstract semantics for the *G*-machine.

LAUNCHBURY. The approach most like ours is LAUNCHBURY's (1993) *natural semantics for lazy evaluation*. The differences between his operational semantics and ours are:

- He presents a 'large-step' operational semantics '$M \Downarrow N$' rather than a 'small-step' semantics '$M \to N$'.
- He presents sequential rather than concurrent reduction, so at each stage there is one node where reduction can take place. This allows him to give his reductions between terms of the form '$\mathsf{rec}\, D \,\mathsf{in}\, M$', where M is the term currently being reduced.
- His syntax does not include local variables '$\nu x \,.\, D$', does not distinguish between tagged and untagged nodes, and does not include fork nodes '$y \vee z$'. It does allow applications of the form 'Mx' rather than just '$x@y$'.

His semantics is (rewritten in our syntax):

$$\frac{}{(\mathsf{rec}\, D \,\mathsf{in}\, \lambda x \,.\, M) \Downarrow (\mathsf{rec}\, D \,\mathsf{in}\, \lambda x \,.\, M)}$$

$$\frac{(\mathsf{rec}\, D \,\mathsf{in}\, x) \Downarrow (\mathsf{rec}\, E \,\mathsf{in}\, \lambda z \,.\, N) \qquad (\mathsf{rec}\, E \,\mathsf{in}\, N[y/z]) \Downarrow (\mathsf{rec}\, F \,\mathsf{in}\, O)}{(\mathsf{rec}\, D \,\mathsf{in}\, x@y) \Downarrow (\mathsf{rec}\, F \,\mathsf{in}\, O)}$$

$$\frac{(\mathsf{rec}\, D \,\mathsf{in}\, M) \Downarrow (\mathsf{rec}\, E \,\mathsf{in}\, N)}{(\mathsf{rec}\, D, x := !M \,\mathsf{in}\, x) \Downarrow (\mathsf{rec}\, E, x := !N \,\mathsf{in}\, N)}$$

$$\frac{(\mathsf{rec}\, D, E \,\mathsf{in}\, M) \Downarrow (\mathsf{rec}\, F \,\mathsf{in}\, N)}{(\mathsf{rec}\, D \,\mathsf{in}\, \mathsf{rec}\, E \,\mathsf{in}\, M) \Downarrow (\mathsf{rec}\, F \,\mathsf{in}\, N)} [\mathsf{fv}\, D \cap \mathsf{wv}\, E = \emptyset]$$

Then it is easy to see that LAUNCHBURY's semantics is a subset of ours, in that if:

$$(\mathsf{rec}\, D \,\mathsf{in}\, M) \Downarrow (\mathsf{rec}\, E \,\mathsf{in}\, N)$$

then:

$$(\mathsf{local}\, D \,\mathsf{in}\, x := !M) \to^* (\mathsf{local}\, E \,\mathsf{in}\, x := !N)$$

However, since LAUNCHBURY's semantics is designed to model sequential rather than concurrent reduction, our semantics has some reductions which cannot be matched by his, for example:

$$(\mathsf{local}\, y := !\mathsf{I}\, \mathsf{I} \,\mathsf{in}\, x := !\mathsf{I}) \to (\mathsf{local}\, y := !\mathsf{I} \,\mathsf{in}\, x := !\mathsf{I})$$

But since the main result of LAUNCHBURY's paper is to show that **D** is an adequate model for his semantics, we have for free that our semantics is adequate for his. Thus by showing that **D** is fully abstract for concurrent graph reduction, we have also shown that concurrent graph reduction is adequate for LAUNCHBURY's model of sequential graph reduction.

LAUNCHBURY has investigated a number of properties of his semantics which have not been covered here, such as space- and time-complexity, update analysis (LAUNCHBURY *et al.*, 1992), and combinators. It is an open problem as to whether these approaches can be directly translated into our semantics.

PURUSHOTHAMAN AND SEAMAN. Another approach to the operational semantics of graph reduction is PURUSHOTHAMAN and SEAMAN's (1992) LAZY-PCF+SHAR, which extends PLOTKIN's (1977) PCF with let declarations. This is given a big-step operational semantics of the form (in our syntax):

$$(\mathsf{let}\, D\, \mathsf{in}\, M) \Downarrow (\mathsf{let}\, E\, \mathsf{in}\, N)$$

This semantics is similar to ours and LAUNCHBURY's, except that:

- LAZY-PCF+SHAR is a typed language, and has constructors and deconstructors for booleans and natural numbers.
- Since let-expressions are being used rather than rec-expressions, the semantics for fixed points lose some sharing information:

$$\frac{(\mathsf{let}\, D\, \mathsf{in}\, \mathsf{let}\, x := !(\mu x.\, M)\, \mathsf{in}\, M) \Downarrow (\mathsf{let}\, E\, \mathsf{in}\, N)}{(\mathsf{let}\, D\, \mathsf{in}\, \mu x.\, M) \Downarrow (\mathsf{let}\, E\, \mathsf{in}\, N)}$$

  Extending the semantics to deal with rec-expressions is ongoing work.
- Garbage collection is not modelled, except in the case when an expression is of ground type (Bool or Int). If M is a function, then their reduction rule for let is:

$$\frac{(\mathsf{let}\, D, x := !N\, \mathsf{in}\, M) \Downarrow (\mathsf{let}\, D', x := !N'\, \mathsf{in}\, M')}{(\mathsf{let}\, D\, \mathsf{in}\, \mathsf{let}\, x := !N\, \mathsf{in}\, M) \Downarrow (\mathsf{let}\, D'\, \mathsf{in}\, \mathsf{let}\, x := !N'\, \mathsf{in}\, M')}$$

  But if M is of ground type, then it has no free variables, and so garbage collection can be performed:

$$\frac{(\mathsf{let}\, D, x := !N\, \mathsf{in}\, M) \Downarrow (\mathsf{let}\, D', x := !N'\, \mathsf{in}\, M')}{(\mathsf{let}\, D\, \mathsf{in}\, \mathsf{let}\, x := !N\, \mathsf{in}\, M) \Downarrow (\mathsf{let}\, D'\, \mathsf{in}\, M')}$$

  This is the only form of garbage collection given for LAZY-PCF+SHAR.

PURUSHOTHAMAN and SEAMAN show that LAZY-PCF+SHAR can be given an adequate semantics in the same domain as PCF. It is an open problem as to whether LAZY-PCF+SHAR with parallel conditionals can be given a fully abstract semantics in the same domain as PCF with parallel conditionals.

ARIOLA AND ARVIND. The *Graph Rewriting Systems* (GRS's) of ARIOLA and ARVIND (1993) are very similar to the declarations introduced in this paper. The only differences are:

- GRS's are not explicitly designed for parallel evaluation, and so do not distinguish between tagged and untagged nodes.
- Local variables are provided by a declaration $\mathsf{local}\, D\, \mathsf{in}\, E$ rather than by $\forall x.\, D$.
- GRS's allow for arbitrary term rewriting, rather than being specific to the untyped λ-calculus.

- GRS's have a term '$\circ$' to denote black holes such as $\mathsf{rec}\, x := !x\, \mathsf{in}\, x$.

The operational semantics for GRS's is very similar to ours and LAUNCHBURY's, for example one of their rules is (in our syntax):

$$(\mathsf{rec}\, D\, \mathsf{in}\, (\mathsf{rec}\, E\, \mathsf{in}\, M)) \rightarrow (\mathsf{rec}\, D, E\, \mathsf{in}\, M)$$

This is the same as our (BUILD) except that reduction is between terms rather than declarations.

ARIOLA and ARVIND present an adequate model for GRS's in terms of sets of normal forms. Since GRS's are independent of the term reductions, it is not obvious whether a fully abstract semantics could be found.

THE AUTHOR. In a previous paper, the author (1993) presents an operational semantics for concurrent graph reduction, and shows that graph reduction is correct for tree reduction. The improvements given in this paper are:

- The proof that the denotational model **D** is fully abstract for concurrent graph reduction.
- The use of ABRAMSKY's (1991) domain theory in logical form to structure the proof of full abstraction.
- The presentation of the operational semantics directly in terms of declarations, rather than introducing a new type of chemical solutions.
- The graphical presentation of graphs has been improved.

The translation of graph reduction into MILNER's (1991) polyadic π-calculus has been omitted, and is further work.

4.2 Future work

There are a number of open problems raised by this work.

SIMPLIFICATION. The operational proofs in Sections 3.6–3.8 are long and rather tedious case analysis. To a degree, this is to be expected, since any verification of a practical implementation technique is likely to involve extended case analysis. However, it would be useful if a presentation could be found which simplified and generalized the proofs given here.

An analogy can be drawn between the presentation of graph reduction by WADSWORTH (1971) and BARENDREGT *et al.* (1987). The former makes explicit mention of application and abstraction nodes, where the latter is a general theory of graph and tree reduction.

Unfortunately, full abstraction results are often very syntax-dependent—for example PLOTKIN's (1977) proof of full abstraction for PCF with parallel conditionals is very dependent on the syntax and operational semantics of PCF.

Finding a proof technique that is powerful enough to show full abstraction for concurrent graph reduction, but does not rely on long case analysis is likely to be quite difficult.

TYPED λ-CALCULI. The proofs given in this paper are only for the untyped λ-calculus with recursive declarations. The non-strict functional languages which are used in practice are typed, and have type constructors and deconstructors (usually in the form of pattern-matching).

Such constructors and deconstructors could be added to the λ-calculus with recursive declarations. For example, the product type $T \times U$ with constructors and deconstructors:

$$\mathsf{pair} : T \to U \to (T \times U) \qquad \mathsf{fst} : (T \times U) \to T \qquad \mathsf{snd} : (T \times U) \to U$$

could be added to the λ-calculus with recursive declarations as:

$$M ::= \cdots \mid \mathsf{pair}\, xy \mid \mathsf{fst}\, x \mid \mathsf{snd}\, x$$

with the operational semantics for fst given:

$$w := !\,\mathsf{fst}\, x, x := ?M \mapsto w := !\,\mathsf{fst}\, x, x := !M$$
$$w := !\,\mathsf{fst}\, x, x := !\,\mathsf{pair}\, yz \mapsto w := !\nabla y, x := !\,\mathsf{pair}\, yz$$

Unfortunately, this leaves the problem of giving a semantics for when x is a function:

$$x := !\,\mathsf{fst}\, y, y := !\lambda w.\, M \mapsto x := !\,\mathsf{Something}, y := !\lambda w.\, M$$

One possibility would be to use a type system to bar such declarations, but this would make the proof dependent on a choice of type system.

Another problem is that the proofs in Chapter 3 rely on the fact that **D** is a lattice. If the boolean type were to be allowed, the model would no longer be a lattice, and so the proofs would require the techniques of ABRAMSKY's (1991) domain theory in logical form, rather than the simpler logic of Chapter 2.

OTHER PARALLEL COMBINATORS. The parallel mechanisms given in this paper are:

- *Parallel evaluation* of the form $\mathsf{rec}\, x := !M \,\mathsf{in}\, N$.
- *Parallel convergence* of the form $x \vee y$.

Neither of these introduce any nondeterminism, which is one reason why concurrent graph reduction has the same fully abstract model as leftmost-outermost reduction. In practice, languages often require a more powerful form of parallel convergence, which returns the value of the term which reached whnf first. For example, if we have a term:

$$\mathsf{pick} : \alpha \to \alpha \to \mathsf{Bool}$$

which says which of its arguments reached whnf, then we can write a function which merges two lists as:

$$\begin{aligned} \mathsf{merge}\, xs\, ys &= \mathsf{if}(\mathsf{pick}\, xs\, ys) \\ &\quad\ \mathsf{then}(\mathsf{merge}'\, xs\, ys) \\ &\quad\ \mathsf{else}(\mathsf{merge}'\, ys\, xs) \\ \mathsf{merge}'\, []\, ys &= ys \\ \mathsf{merge}'\, (x :: xs)\, ys &= x :: (\mathsf{merge}\, xs\, ys) \end{aligned}$$

The merge function can then be used, for example, in I/O routines which need to merge two event streams.

The pick function could be added to the λ-calculus with rec as:

$$M ::= \cdots \mid \mathsf{pick}\, xy$$

with the operational semantics given:

$$x := !\,\mathsf{pick}\, yz, y := ?M \mapsto x := !\,\mathsf{pick}\, yz, y := !M$$
$$x := !\,\mathsf{pick}\, yz, z := ?M \mapsto x := !\,\mathsf{pick}\, yz, z := !M$$
$$x := !\,\mathsf{pick}\, yz, y := !\lambda w.\, M \mapsto x := !\,\mathsf{true}, y := !\lambda w.\, M$$
$$x := !\,\mathsf{pick}\, yz, z := !\lambda w.\, M \mapsto x := !\,\mathsf{false}, z := !\lambda w.\, M$$

This is the same operational semantics as $y \vee z$, except that we return true ($\lambda xy.\, x$) or false ($\lambda xy.\, y$) rather than I.

Although it is simple to give an operational semantics for the λ-calculus with recursive declarations and pick, finding a fully abstract model is non-trivial. For example, pick is not referentially transparent, since:

$$[\![\mathsf{rec}(x := ?\,\mathsf{pick\ I\ I})\ \mathsf{in}\ (x \Leftrightarrow x)]\!] = [\![\mathsf{true}]\!]$$
$$[\![\mathsf{pick\ I\ I} \Leftrightarrow \mathsf{pick\ I\ I}]\!] = [\![\mathsf{pick\ I\ I}]\!]$$

This lack of referential transparency is caused by non-determinism, so one would expect to need a model based on powersets. One open problem is to show that any fully abstract model for leftmost-outermost reduction of the λ-calculus with nondeterminism is also fully abstract for concurrent graph reduction with nondeterminism. One would then be able to apply the techniques of ONG (1993) or DE 'LIGUORO and PIPERNO (1992) to concurrent graph reduction.

Another approach, which is still nondeterministic, but retains referential transparency, is AUGUSTSSON's (1989) *oracles*, which have been implemented in Lazy ML. This replaces pick with a choose function, that as a side-effect sets an oracle variable recording the result.

In AUGUSTSSON's implementation there are an unbounded number of oracles, but we can simply model the case where there is only one oracle variable called o. This is initially declared as $\mathsf{o} := !\bot$, and can have one of three states:

$\bot$, l or r. The choose function could be added to the λ-calculus with recursive declarations as:

$$M ::= \cdots \mid \mathsf{choose}\, xy$$
$$D ::= \cdots \mid \mathsf{o} := !\bot \mid \mathsf{o} := !\mathsf{l} \mid \mathsf{o} := !\mathsf{r}$$

with the operational semantics given:

$$x := !\,\mathsf{choose}\, yz, y := ?M \mapsto x := !\,\mathsf{choose}\, yz, y := !M$$
$$x := !\,\mathsf{choose}\, yz, z := ?M \mapsto x := !\,\mathsf{choose}\, yz, z := !M$$
$$\mathsf{o} := !\bot, x := !\,\mathsf{choose}\, yz, y := !\lambda w.\, M \mapsto \mathsf{o} := !\mathsf{l}, x := !\,\mathsf{choose}\, yz, y := !\lambda w.\, M$$
$$\mathsf{o} := !\bot, x := !\,\mathsf{choose}\, yz, z := !\lambda w.\, M \mapsto \mathsf{o} := !\mathsf{r}, x := !\,\mathsf{choose}\, yz, z := !\lambda w.\, M$$
$$\mathsf{o} := !\mathsf{l}, x := !\,\mathsf{choose}\, yz \mapsto \mathsf{o} := !\mathsf{l}, x := !\,\mathsf{true}$$
$$\mathsf{o} := !\mathsf{r}, x := !\,\mathsf{choose}\, yz \mapsto \mathsf{o} := !\mathsf{r}, x := !\,\mathsf{false}$$

This is the same operational semantics as pick as long as the o variable is $\bot$. Once the o variable has been set, choose xy always returns the same result. Thus, choose is nondeterministic, but is referentially transparent, for example:

$$[\![\mathsf{rec}(x := ?\,\mathsf{choose}\,\mathsf{l}\,\mathsf{l})\ \mathsf{in}\ (x \Leftrightarrow x)]\!] = [\![\mathsf{true}]\!]$$
$$[\![\mathsf{choose}\,\mathsf{l}\,\mathsf{l} \Leftrightarrow \mathsf{choose}\,\mathsf{l}\,\mathsf{l}]\!] = [\![\mathsf{true}]\!]$$

It is an open problem to find a model for such an operator, which is referentially transparent, but which may have side-effects.

References

ABRAMSKY, S. (1989). The lazy lambda calculus. In TURNER, D., editor, *Declarative Programming*. Addison-Wesley.

ABRAMSKY, S. (1991). Domain theory in logical form. *Ann. Pure Appl. Logic*, 51:1–77.

ARIOLA, Z. M. and ARVIND (1993). Graph rewriting systems for efficient compliation. In SLEEP, M. R., PLASMEIJER, M. J., and VAN EEKELEN, M. C. J. D., editors, *Term Graph Rewriting: Theory and Practice*, chapter 6. John Wiley and Sons.

AUGUSTSSON, L. (1984). A compiler for lazy ML. In *Proc. ACM Symp. Lisp and Functional Programming*, pages 218–227.

AUGUSTSSON, L. (1989). Non-deterministic programming in a deterministic functional language. Memo 66, Programming Methodology Group, Chalmers University.

BARANDREGT, H. P., COPPO, M., and DEZANI-CIANCAGLINI, M. (1983). A filter lambda model and the completeness of type assignment. *J. Symbolic Logic*, 48(4):931–940.

BARENDREGT, H. P. (1984). *The Lambda Calculus*. North-Holland. Studies in logic 103.

BARENDREGT, H. P., VAN EEKELEN, M. C. J. D., GLAUERT, J. R. W., KENNAWAY, J. R., PLASMEIJER, M. J., and SLEEP, M. R. (1987). Term graph rewriting. In *Proc. PARLE 87*, volume 2, pages 141–158. Springer-Verlag. LNCS 259.

BERRY, G. and BOUDOL, G. (1990). The chemical abstract machine. In *Proc. 17th Ann. Symp. Principles of Programming Languages*.

BOUDOL, G. (1992). Lambda-calculi for (strict) parallel functions. Technical report 1387, INRIA Sophia-Antipolis.

BROOKES, S. D., HOARE, C. A. R., and ROSCOE, A. W. (1984). A theory of communicating sequential processes. *J. Assoc. Comput. Mach.*, 31(3):560–599.

CLEAVELAND, R., PARROW, J., and STEFFEN, B. (1989). The concurrency workbench. In *Proc. Workshop Automatic Verification Methods for Finite-State Systems*. Springer-Verlag. LNCS.

DAVEY, B. A. and PRIESTLEY, H. A. (1990). *Introduction to Lattices and Order*. Cambridge University Press.

DE 'LIGUORO, U. and PIPERNO, A. (1992). Non deterministic extensions of untyped λ-calculus. In RAOULT, J.-C., editor, *Proc. CAAP 92*. Springer-Verlag. LNCS 581.

DE NICOLA, R. (1985). *Testing Equivalences and Fully Abstract Models for Communicating Processes*. Ph.D. thesis, University of Edinburgh.

EVANS JR, A. (1968). PAL—a language for teaching programming linguistics. In *Proc. ACM 23rd Natl. Conf.* Brandon/Systems Press.

FAIRBURN, J. (1982). Ponder and its type system. Technical report 31, Cambridge University Computer Lab.

HENNESSY, M. (1988). *Algebraic Theory of Processes*. MIT Press.

HENNESSY, M. and MILNER, R. (1980). On observing nondeterminism and concurrency. In DE BAKKER, J. W. and VAN LEEUWEN, J., editors, *Proc. ICALP 80*. Springer-Verlag. LNCS 85.

HOARE, C. A. R. (1985). *Communicating Sequential Processes*. Prentice-Hall.

HUDAK, P., PEYTON JONES, S. L., WADLER, P., *et al.*(1992). A report on the functional language Haskell. *SIGPLAN Notices*.

HUGHES, J. (1984). *The Design and Implementation of Programming Languages*. D.Phil thesis, Oxford University.

JEFFREY, A. (1993). A chemical abstract machine for graph reduction. In *Proc. MFPS 93*. Springer-Verlag. LNCS.

JOHNNSON, T. (1984). Effecient compilation of lazy evaluation. In *Proc. Sigplan 84 Symp. Compiler Construction*, pages 58–69.

JONES, M. (1992). The Gofer technical manual. Part of the Gofer distribution.

KENNAWAY, J. R., KLOP, J. W., SLEEP, M. R., and DE VRIES, F. J. (1993a). The adequacy of term graph rewriting for simulating term rewriting. In SLEEP, M. R., PLASMEIJER, M. J., and VAN EEKELEN, M. C. J. D., editors, *Term Graph Rewriting: Theory and Practice*, chapter 12. John Wiley and Sons.

KENNAWAY, J. R., KLOP, J. W., SLEEP, M. R., and DE VRIES, F. J. (1993b). An introduction to term graph rewriting. In SLEEP, M. R., PLASMEIJER, M. J., and VAN EEKELEN, M. C. J. D., editors, *Term Graph Rewriting: Theory and Practice*, chapter 1. John Wiley and Sons.

LARSEN, K. G., GODSKESEN, J. C., and ZEEBERG, M. (1989). TAV, tools for automatic verification, user manual. Technical report R 89–19, Dept Math. and Comp. Sci., Ålborg University.

LASSEZ, J. L., NGUYEN, V. L., and SONENBERG, E. A. (1982). Fixed point theorems and semantics: A folk tale. *Information Processing Letters*, 14(3):112–116.

LAUNCHBURY, J. (1993). A natural semantics for lazy evaluation. In *Proc. ACM Sigplan–Sigact POPL*.

LAUNCHBURY, J., GILL, A., HUGHES, J., MARLOW, S., PEYTON JONES, S. L., and WADLER, P. (1992). Avoiding unnecessary updates. In *Proc. Glasgow Functional Programming Workshop*, Dept. Comp. Sci., Glasgow University.

LESTER, D. (1989). *Combinator Graph Reduction: A Congruence and its Applications*. D.Phil thesis, Oxford University.

MAC LANE, S. (1971). *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer-Verlag.

MILNER, R. (1977). Fully abstract semantics of typed λ-calculi. *Theoret. Comput. Sci.*, 4:1–22.

MILNER, R. (1989). *Communication and Concurrency*. Prentice-Hall.

MILNER, R. (1991). The polyadic π-calculus: a tutorial. In *Proc. International Summer School on Logic and Algebra of Specification*, Marktoberdorf.

MORRIS, J.-H. (1968). Lambda calculus models of programming languages. Dissertation, M.I.T.

MYCROFT, A. (1981). *Abstract Interpretation and Optimising Transformations for Applicative Programs*. PhD thesis, Edinburgh University Dept. Computer Science.

ONG, C.-H. L. (1988). *The Lazy Lambda Calculus: An Investigation into the Foundations of Functional Programming*. PhD thesis, Imperial College, London University.

ONG, C.-H. L. (1993). Non-determinism in a functional setting. In *Proc. LICS 93*, pages 275–286. IEEE Computer Soc. Press.

PEYTON JONES, S. L. (1987). *The Implementation of Functional Programming Languages*. Prentice-Hall.

PIERCE, B. C. (1991). *Basic Category Theory for Computer Scientists*. MIT press.

PLOTKIN, G. (1977). LCF considered as a programming language. *Theoret. Comput. Sci.*, 5:223–256.

PLOTKIN, G. (1983). Domains. available by anonymous ftp.

PURUSHOTHAMAN, S. and SEAMAN, J. (1992). An adequate operational semantics of sharing in lazy evaluation. In *Proc. ESOP 92*.

ROSE, K. H. (1993). Explicit cyclic substitution. Semantics note D–166, DIKU, University of Copenhagen.

SCOTT, D. S. (1982). Domains for denotational semantics. In NEILSEN, M. and SCHMIDT, E. M., editors, *Proc. ICALP 82*, pages 577–613. Springer-Verlag. LNCS 140.

SLEEP, M. R., PLASMEIJER, M. J., and VAN EEKELEN, M. C. J. D., editors (1993). *Term Graph Rewriting: Theory and Practice*. John Wiley and Sons.

STOY, J. E. (1977). *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press.

TURNER, D. (1979). A new implementation technique for applicative languages. *Software: Practice and Experience*, 9:31–49.

TURNER, D. (1985). Miranda: A non-strict functional language with polymorphic types. In *Proc. IFIP Conf. Functional Programming Languages and Computer Architecture*. Springer-Verlag. LNCS 201.

WADSWORTH, C. P. (1971). *Semantics and Pragmatics of the Lambda Calculus*. D.Phil thesis, Oxford University.

Index of authors

Index of definitions