From SOS Rules to Proof Principles:
An Operational Metatheory for Functional Languages

David Sands

Chalmers, Sweden[1]

Abstract

Structural Operational Semantics (SOS) is a widely used formalism for specifying the computational meaning of programs, and is commonly used in specifying the semantics of functional languages. Despite this widespread use there has been relatively little work on the "metatheory" for such semantics. As a consequence the operational approach to reasoning is considered *ad hoc* since the same basic proof techniques and reasoning tools are reestablished over and over, once for each operational semantics specification. This paper develops some metatheory for a certain class of SOS language specifications for functional languages. We define a *rule format*, *Globally Deterministic SOS* (GDSOS), and establish some proof principles for reasoning about equivalence which are sound for all languages which can be expressed in this format. More specifically, if the SOS rules for the operators of a language conform to the syntax of the GDSOS format, then

- a syntactic analogy of continuity holds, which relates a recursive function to its finite unwindings, and forms the basis of a Scott-style fixed-point induction technique;
- a powerful induction principle called *improvement induction* holds for a certain class of *instrumented* GDSOS *semantics*; the *Improvement Theorem* from [Sands, POPL'95] is a simple corollary;
- a useful bisimulation-based coinductive proof technique for operational approximation (and its "instrumented" variants) is sound.

To Appear: 24th Annual SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Paris, France, January 15–17, 1997.

1 Introduction

Methods for reasoning about equivalence in functional languages are often based upon denotational semantics. One characteristic feature of such semantics is that recursion in the language is modelled by the construction of a least fixed point in some suitable ordered domain of interpretations. The meta-theory of denotational semantics—domain theory—guarantees the existence of fixed points, and provides certain reasoning principles for recursion based on this theory. Operational semantics can also be used as the basis for reasoning about equivalence in functional languages, and has recently enjoyed something of a revival. A notable operationally-based technique is the use of "applicative bisimulation" [Abr90, How89, Gor95] to reason about equivalence.

Comparing the denotational and operational based approaches, the operational approach has the advantage that it is built up from rather modest mathematical tools; it also appears that the operationally-based techniques are fairly robust with respect to changes or extensions to the underlying language. The disadvantage of operationally-based reasoning is that there is no analogy of the domain theory that underpins the denotational approach. Consequently, any reasoning techniques or tools that are built up from operational semantics must be reestablished, and the same theorems must be reproved for each operational semantics specification.

In this article we consider some "metatheory" for operational semantics, by studying a *rule-format*, GDSOS, — a syntactic schema for SOS rules suitable for specifying small-step evaluation in functional languages. A number of proof techniques are established for *any* language whose operational semantics specification matches the format.

[1]Department of Computing Science,
Chalmers University of Technology and Göteborg University,
S-412 96 Göteborg, Sweden
dave@cs.chalmers.se http://www.cs.chalmers.se/~dave

The contributions of the paper are firstly to lift some standard proof techniques to the level of a rule format, and secondly, also at the rule-format level, to introduce a new operational proof technique based on instrumented semantics. Specifically, the proof techniques we establish for GDSOS languages are

- *least fixed-point properties* of the style usually associated with the standard denotational approach;
- *improvement induction* — a new proof technique based on an instrumented semantics and a corresponding definition of a contextual *improvement* relation between terms, and
- *coinductive proof techniques* of the standard bisimulation variety usually associated with the operational approach.

Recursion as a Least Fixed Point The properties of recursive definitions usually associated with the trappings of a denotational semantics can be established at the syntactic level from operational semantics, with very little mathematical overhead. We establish a syntactic analogy of continuity which forms the basis of Scott-style fixed point induction. Syntactic continuity says that a recursive constant $\mathbf{f}$ can be characterised in terms of its syntactic finite unwindings, $\{\mathbf{f^i}\}_{i\geq 0}$. More specifically, $\mathbf{f}$ is the least term (up to operational equivalence) which is operationally greater than the $\mathbf{f}^i$; moreover, this property is preserved by all syntactic contexts $C[\,]$:

$$C[\mathbf{f}] \sqsubseteq_{\sim} M \iff \forall i.\, C[\mathbf{f}^i] \sqsubseteq_{\sim} M$$

where $\sqsubseteq_{\sim}$ is the operational approximation ordering: $M \sqsubseteq_{\sim} N$ iff for all contexts C', $C'[N]$ terminates whenever $C'[M]$ terminates.

Improvement Induction The following operational variant of the least pre-fixed-point property of a recursive definition is easily established from the syntactic continuity: for all recursive $\mathbf{f}$ defined by $\mathbf{f} \triangleq C[\mathbf{f}]$

$$M \sqsupseteq_{\sim} C[M] \Rightarrow M \sqsupseteq_{\sim} \mathbf{f}$$

This is sometimes known as *Park Induction.* Unfortunately, the "dual" principle, that $M \sqsubseteq_{\sim} C[M]$ (or $M \cong C[M]$) implies $M \sqsubseteq_{\sim} \mathbf{f}$, is not sound in general.

The idea of the *improvement induction principle* introduced in this article is to establish $M \sqsubseteq_{\sim} \mathbf{f}$ by establishing some stronger relation between M and $C[M]$ than operational approximation. The improvement induction principle shows that there are many possible (nontrivial) relations, each one obtained by instrumenting the GDSOS rules with additional "resource" information, and by deriving strengthened definitions of operational approximation, called *improvement*, which also take into account resource use.

A fairly immediate corollary of the improvement induction principle is a generalisation of the Improvement Theorem [San96b]. The Improvement Theorem for a specific higher-order functional language has been used to develop a correctness preserving variant of unfold-fold transformations (in *loc. cit.*), as well as to give the first correctness proofs for some well known transformation methods [San96a].

Functional (bi)simulations and Coinduction
Operationally-based proof principles of a more standard nature are also established for GDSOS languages. We show that coinductive reasoning techniques based on bisimulation (*à la* [Abr90, How89, Pit99, Gor95]) are sound for reasoning about the various preorders and equivalences discussed in this article.

1.1 Related Work

Proofs of "standard" fixed-point properties based on operational semantics, for specific languages, can be found in [Tal85, Smi92, Dam94, MT91, MST96, Pit99]. We have taken the term "syntactic continuity" from [Pit99] although our proof is somewhat more direct, and much more in the spirit of proofs found in [Smi92, MST96, MT91].[1] The main contribution of this paper for this kind of theorem is to lift these proofs from a particular (functional) language to any functional language whose structural operational semantics fit a certain *rule format*; the rules of [MST96] (a call-by-value functional language) fit the rule format, as do the call-by-name rules of [Smi92] and [Pit99].

Rule formats are well-known in process algebra (e.g., see [GV92], [BIM95]) where the typical theorem established is that bisimulation is a congruence. Aceto, Bloom, and Vaandrager [ABV94] study axiomatisations of strong bisimulation for the GSOS format of [BIM95]; to handle potential recursion include an induction principle known as *approximation induction.*

The use of rule formats in the setting of functional programs is rather less well-known. Bloom [Blo90] defines a rule format for operators extending the LCF language, and establishes that all such extensions satisfy a certain operational extensionality property. The rule format is rather restrictive, does not permit variable-binding constructs, and the arguments are assumed to

[1]It should be noted that this is a continuity property of a particular chain; in general, continuity fails for the operational ordering (see [Smi92]).

evaluate to simple constants.

Howe [How91, How96] defines a rule format for the evaluation relation for functional languages, and proves that a functional analogy of bisimulation is a congruence. Our meta-syntax for defining rules is inspired by Howe's. Our rule format is based on a single step reduction relation, and appears to be slightly less general, since it seems possible to obtain, from our rule format for reduction, a natural semantics format which is subsumed by Howe's. Howe does not, however, consider fixed-point properties, or other induction principles other than (bi)simulation. We establish a similar result to Howe's for GDSOS (using the same methods), but also for instrumented variations of the semantics. This is a fairly easy adaptation of our earlier work on operational theories of improvement [San91], which gave more abstract conditions (not based on SOS rules *per se*) which guarantee that a certain coinductively defined class of improvement relations are congruences. We give some consideration to the special case of purely call-by-value languages (languages in which all variables get bound to values) via the notion of value metaterms in the SOS rules. Howe achieves the same ends by using call-by-value variables and explicit abstraction over values as part of the ordinary syntax. A yet more elaborate term-syntax (much more elaborate than needed in our setting) in which variables are sorted, is described in [FV95]. They study very basic conservativity properties of SOS rules, but covering a very general class of languages.

Independently, Kristian Nielsen [Nie96] has recently introduced a kind of rule format (expressing what he calls "simple functional languages") in both "large-step" and equivalent "small-step" forms. The aims are to develop some generalised theory for partial evaluation and deforestation. Although notationally rather different, GDSOS and Nielsen's format are very close. The most recent refinement of GDSOS was to include non-strict data constructors; the technical development of this aspect (mostly omitted from this version of the paper) was influenced by Nielsen's approach. Independently of Howe's work, Nielsen establishes that applicative bisimulation is a congruence.

2 Preliminaries

2.1 Operational Orderings

To define what it means for two programs to be equivalent, or for one program to "approximate" another, we use an evaluation relation as our basic building block. An evaluation relation, $\Downarrow$, is a binary relation between closed terms and values; values are themselves just terms of a particular syntactic form. If $M{\Downarrow}V$ then computation beginning with M terminates with value V.

Given such a relation, we can build, in a standard way, definitions of approximation and equivalence. The operational approximation we use is the standard Morris-style contextual ordering. The notion of "observation" we take is just the fact of convergence, as in the lazy lambda calculus [Abr90]. This particular choice is not central to the development, and could be generalised by a definition parameterised by a suitable notion of an "observable" value.

Operational equivalence equates two expressions if and only if in all closing contexts they give rise to the same observation – i.e. either they both converge, or they both diverge.

DEFINITION 2.1

(i) M *operationally approximates* N, $M \precsim N$, if for all contexts C such that $C[M]$, $C[N]$ are closed, if $C[M]{\Downarrow}V$ then there exists a W such that $C[N]{\Downarrow}W$.

(ii) M is *operationally equivalent* to N, $M \cong N$, if $M \precsim N$ and $N \precsim M$.

We could also parameterise the theory by a static typing discipline, which would only allow the comparison of certain well-typed terms, and then only in appropriately typed contexts. This adds noise but relatively little effect to the developments in this paper, since most of the theorems are proved directly from the operational semantics, and this, by definition, is independent of static typing. Adoption of a *dynamic* typing discipline would require more fundamental changes to the framework.

2.2 Structural Operational Semantics

In this article we will consider evaluation relations which are defined in terms of a finer one-step evaluation relation, $\mapsto$; we define $M{\Downarrow}V$ for some value V if $M \mapsto^* V$. The mechanism we consider for defining $\mapsto$ is the well-known *Structural Operational Semantics* (SOS) [Plo81]. SOS is a syntactic style for presenting an inductively defined transition relation between "machine configurations", given by cases according to their syntactic structure. In our setting, the machine configurations will be simply closed terms. As an example we present the one-step evaluation relation for the lazy lambda calculus [Abr90], together with booleans and conditionals, in Figure 1. For this language the values are deemed to be ***true***, ***false***, and any closed lambda-term. It is easy to see that $\mapsto$, and hence $\Downarrow$ are partial functions.

3 Second Order Abstract Syntax

We introduce an abstract syntax for specifying functional programs, and for specifying the GDSOS rule for-

$$\frac{M \mapsto M'}{M\,N \mapsto M'\,N} \qquad \frac{}{(\lambda x.M)\,N \mapsto M\{{}^{N}\!/_{x}\}}$$

$$\frac{M \mapsto N}{\textbf{if } M \textbf{ then } N_1 \textbf{ else } N_2 \;\mapsto \textbf{if } N \textbf{ then } N_1 \textbf{ else } N_2}$$

$$\frac{}{\textbf{if } \boldsymbol{true} \textbf{ then } N_1 \textbf{ else } N_2 \;\mapsto N_1} \qquad \frac{}{\textbf{if } \boldsymbol{false} \textbf{ then } N_1 \textbf{ else } N_2 \;\mapsto N_2}$$

Figure 1: Example SOS Rules

mat given in the next section. Our syntax follows [How91] very closely, but is fairly standard from the point of view of formal specification of syntax which includes variable binding operators such as lambda abstraction (e.g., see [NPS90, Klo80, PE88]).

First we fix a countably infinite set *Var* of ordinary variables. A language L is specified by (amongst other things) a set of operators O of a fixed arity α. As usual, the arity specifies the number of *operands* for each operator, but it specifies more than just this, since we wish to specify the syntax of operators with binding. Each operand is possibly an abstraction, i.e., a list of zero or more distinct variables followed by a term, where the variables are considered bound in the term. The arity of an operator is therefore given by a sequence of natural numbers; the length of the sequence is the number of operands, and the natural numbers are the number of bound variables associated with the corresponding operand. For example, the language implicit in the rules of Figure 1 would be represented in this syntax by the set of operators $\{\lambda, \boldsymbol{apply}, \boldsymbol{if}, \boldsymbol{true}, \boldsymbol{false}\}$ with arities $\alpha(\lambda) = (1)$, $\alpha(\boldsymbol{apply}) = (0,0)$, $\alpha(\boldsymbol{if}) = (0,0,0)$, $\alpha(\boldsymbol{true}) = \alpha(\boldsymbol{false}) = ()$.

Let x, y, etc., range over *Var*, and let $\boldsymbol{p}$, $\boldsymbol{q}$ range over O.

The terms of L, T, ranged over by M, N are defined inductively as follows:

$$\frac{}{x \in T} \qquad \frac{M_1 \in T \cdots M_n \in T}{\boldsymbol{p}(\,(\vec{x}_1)M_1, \ldots, (\vec{x}_n)M_n) \in T}$$

where $\alpha(\boldsymbol{p}) = (k_1, \ldots k_n)$
and each $\vec{x}_i$ is a list of k_i distinct variables.

For example, the term $(\lambda x.y)z$ would be written in this abstract syntax as $\boldsymbol{apply}(\boldsymbol{\lambda}((x)y), z)$. Where we need to distinguish between the terms from different languages we will write $T(L)$ for the terms of L.

3.1 Meta Terms

In order to formalise the rules of a structural operational semantics for such a language we introduce a syntax for *meta terms*. We begin by fixing a a countable set of *metavariables Mvar* ranged over by $\mathcal{X}$, $\mathcal{Y}$, etc., and a disjoint set of *value metavariables*, *ValMvar*, ranged over by $\mathcal{V}$, $\mathcal{W}$. Metavariables will range over both terms and operands (abstractions); value metavariables will range over only those terms which are deemed to be values. With each metavariable $\mathcal{X}$, we associate an arity $\alpha(\mathcal{X})$ which is a natural number. The idea is that metavariables of arity 0 will range over terms, while meta variables of higher arity will range over abstractions. Value metavariables always have arity 0.

Definition 3.1 To define the metaterms for a given language L, we define an indexed set of meta-abstractions $\{MT_i\}_{i\geq 0}$, (ranged over by $\mathcal{M}$, $\mathcal{N}$, etc.) inductively according to the following rules:

$$\frac{}{x \in MT_0} \qquad \frac{}{\mathcal{X} \in MT_n}\ \alpha(\mathcal{X}) = n$$

$$\frac{\mathcal{M} \in MT_0}{(x_1, \ldots, x_n)\mathcal{M} \in MT_n}$$

$$\frac{\mathcal{M}_1 \in MT_0 \cdots \mathcal{M}_n \in MT_0}{\mathcal{X} \cdot (\mathcal{M}_1, \ldots, \mathcal{M}_n) \in MT_0}\ \alpha(\mathcal{X}) = n$$

$$\frac{\mathcal{M}_1 \in MT_{k_1} \ldots \mathcal{M}_n \in MT_{k_n}}{\boldsymbol{p}(\mathcal{M}_1, \ldots, \mathcal{M}_n) \in MT_0}\ \alpha(\boldsymbol{p}) = (k_1 \ldots k_n)$$

Then we define the metaterms to be MT_0.

The construct $\mathcal{X}\cdot(\mathcal{M}_1, \ldots, \mathcal{M}_n)$ denotes meta-application. Metavariable $\mathcal{X}$, of arity n, ranges over abstractions of n variables. When $\mathcal{X}$ is replaced by such an abstraction, the meta-terms $\mathcal{M}_1, \ldots, \mathcal{M}_n$ will be substituted for the respective abstracted variables. Let σ range over second-order substitutions—finite mappings from metavariables to abstractions of corresponding arity (and such that value meta-variables can only be bound to values). Thus if $\sigma(\mathcal{X})$ is defined then it has the form $(x_1, \ldots, x_n)M$. The operation of second order substitution is defined as for normal substitution, except that if $\sigma(\mathcal{X}) = (x_1, \ldots, x_n)M$ then

$$\sigma(\mathcal{X}\cdot(\mathcal{M}_1, \ldots, \mathcal{M}_n)) = M\{{}^{\sigma(\mathcal{M}_1), \ldots, \sigma(\mathcal{M}_n)}\!/_{x_1, \ldots, x_n}\}.$$

3.2 Values and Metavalues

Before we introduce the GDSOS rule format we need to specify the set of terms deemed to be values. In addition to the arity, a language L comes equipped with a specification of a syntactic *strictness* property for each operator. For each operator $\boldsymbol{p}$ of arity $\alpha(\boldsymbol{p}) = (k_1 \dots k_n)$ we must specify a set $\mathit{strict}(\boldsymbol{p}) \subseteq \{1 \dots n\}$ — which we call the set of *strict arguments* of $\boldsymbol{p}$. This set satisfies the property that $i \in \mathit{strict}(\boldsymbol{p}) \Rightarrow k_i = 0$ — in other words strict arguments cannot be abstractions.

Furthermore, each operator of the language is classified as either a *constructor* or a *non-constructor*. We use $\boldsymbol{c}$, $\boldsymbol{c}'$ etc. to range over constructors. Values are built inductively from the constructors; meta-values are a particular set of metaterms which only match values.

DEFINITION 3.2 The set of *values*, *Val*, ranged over by V, W etc. is the subset of the terms of a given language defined inductively as follows:

$$\frac{\{M_j \in \mathit{Val} \mid j \in \mathit{strict}(\boldsymbol{c})\}}{\boldsymbol{c}(\,(\vec{x}_1)M_1, \dots, (\vec{x}_n)M_n) \in \mathit{Val}}$$

The set of *simple metavalues* is a subset of the metaterms defined inductively by the following rules:

$$\frac{}{\mathit{ValMvar} \subseteq \mathit{Mval}}$$

$$\frac{\begin{array}{ll}\{\mathcal{M}_j \in \mathit{Mval} & \mid\ j \in \mathit{strict}(\boldsymbol{c})\} \\ \{\mathcal{M}_k \in \mathit{Mvar} & \mid\ k \in \{1 \dots n\} \backslash \mathit{strict}(\boldsymbol{c})\}\end{array}}{\boldsymbol{c}(\mathcal{M}_1, \dots, \mathcal{M}_n) \in \mathit{Mval}}$$

Example If $\boldsymbol{stream}$ is a constructor of arity $(0, 0)$ which is strict in its first argument, then $\boldsymbol{stream}(\boldsymbol{true}, \mathcal{X})$ and $\boldsymbol{stream}(\mathcal{V}, \mathcal{X})$ are simple metavalues, but $\boldsymbol{stream}(\mathcal{X}, \mathcal{Y})$ is not (it matches some non-values), and neither are $\boldsymbol{stream}(\mathcal{V}, \boldsymbol{stream}(\mathcal{X}, \mathcal{Y}))$ or $\boldsymbol{stream}(\boldsymbol{true}, \mathcal{V})$ (they are not "simple").

4 The GDSOS Rule Format

The purpose of introducing metaterms is to describe the rules of a structural operational semantics. As an example, an axiom for a call-by-value application operator, @ might be written: $\frac{}{@(\lambda\mathcal{X}, \mathcal{V}) \mapsto \mathcal{X} \cdot \mathcal{V}}$.
Such a rule schema defines transitions on terms as follows: for all second order substitutions σ which map $\mathcal{X}$ to some closed abstraction $(x)M$ and $\mathcal{V}$ to some closed value, we have that $\sigma(@(\lambda\mathcal{X}, \mathcal{V})) \mapsto \sigma(\mathcal{X} \cdot \mathcal{V}) = M\{{}^{\sigma(\mathcal{V})}/_{x}\}$. Here we use the metaterms to define a particular rule format, the GDSOS format.

DEFINITION 4.1 (GDSOS) The operational semantics for a language L is in *Globally deterministic SOS format* (GDSOS format) if the one step evaluation relation can be expressed by rules of the following forms.

The SOS rules for each operator $\boldsymbol{p}$ of arity $(k_1, \dots k_n)$ consist of $|I|$ inference rules, where $I = \mathit{strict}(\boldsymbol{p})$, together with possibly infinitely many axioms of the following forms:

$$\boxed{\begin{array}{c}\left\{\dfrac{\mathcal{X}_i \mapsto \mathcal{Y}_i}{\boldsymbol{p}(\mathcal{X}_1 \dots \mathcal{X}_i \dots \mathcal{X}_n) \mapsto \boldsymbol{p}(\mathcal{X}_1 \dots \mathcal{Y}_i \dots \mathcal{X}_n)}\right\}_{i \in I} \\ \\ \left\{\dfrac{}{\boldsymbol{p}(\mathcal{M}_{1j}, \dots, \mathcal{M}_{nj}) \mapsto \mathcal{M}_j}\right\}_{j \in J}\end{array}}$$

for some indexing set J such that if $\boldsymbol{p}$ is a constructor then $J = \emptyset$, and such that for all $i \in I$, $j \in J$,

(i) $\mathcal{M}_{ij}$ is a simple metavalue; other metaterms $\mathcal{M}_{hj}$ $h \notin I$ are metavariables;

(ii) metavariables $\mathcal{X}_i$ and $\mathcal{Y}_i$ are distinct, and all metavariables and value metavariables occur at most once in any metaterm except $\mathcal{M}_j$;

(iii) the set of left-hand sides of the axioms are non-overlapping (that is, there is no second-order substitution which unifies any two left hand sides);

(iv) there are no free (ordinary) variables in $\mathcal{M}_j$.

The operational semantics for a language L is defined to be in *call-by-value* GDSOS *format* if it is in GDSOS format, and additionally all meta-applications $\mathcal{X} \cdot (\mathcal{N}_1 \dots \mathcal{N}_n)$ occuring in the $\mathcal{M}_j$ are such that all closed instances of the $\mathcal{N}_i$ are necessarily *values*.

Before giving some examples of rules in this format we give some intuitions for the requirements of the GDSOS format. The collection of $|\mathit{strict}(\boldsymbol{p})|$ inference rules for each operator in GDSOS format reflect that evaluation steps may occur in any of the strict arguments. The side conditions can be explained respectively as follows:

(i) implies that the axioms can only be applied once all the strict arguments have been evaluated to a value. The definition of simple metavalues ensures that the evaluation step can only depend on the constructors of these values;

(ii) prevents the rules from depending on syntactic equivalence of arbitrary terms;

(iii) is needed to ensure that the evaluation relation derived from the system is deterministic;

(iv) ensures that free variables are not introduced by reduction (and hence is also needed for determinacy).

The definition of call-by-value GDSOS will be used to refine certain proof principles to the special case of call-by-value languages.

Examples Here we give some examples of operators which can be expressed in GDSOS format. The rules of Fig. 1 can all be written in this format. Consider a slight variation: a call-by-value application operator. This operator is strict in both arguments. The associated rules are:

$$\boxed{\begin{array}{cc} \dfrac{\mathcal{Y} \mapsto \mathcal{Y}'}{@(\mathcal{Y},\mathcal{Z}) \mapsto @(\mathcal{Y}',\mathcal{Z})} & \dfrac{\mathcal{Z} \mapsto \mathcal{Z}'}{@(\mathcal{Y},\mathcal{Z}) \mapsto @(\mathcal{Y},\mathcal{Z}')} \\ \multicolumn{2}{c}{\overline{@(\lambda\mathcal{X},\mathcal{V}) \mapsto \mathcal{X}\cdot\mathcal{V}}} \end{array}}$$

As an example of a operator which is strict in no arguments, consider a local recursion construct with concrete syntax **letrec** $x = M$ **in** N, and with a computation rule:

$$\mathbf{letrec}\ x = M\ \mathbf{in}\ N \mapsto N\{^{\mathbf{letrec}\ x = M\ \mathbf{in}\ M}/_{x}\}$$

The ***letrec*** operator has arity $(1,1)$, and **letrec** $x = M$ **in** N has abstract syntax $\boldsymbol{letrec}((x)M,(x)N)$. The rule is

$$\boxed{\overline{\boldsymbol{letrec}(\mathcal{X},\mathcal{Y}) \mapsto \mathcal{Y}\cdot\boldsymbol{letrec}(\mathcal{X},\mathcal{X})}}$$

As a more involved example, consider a simple Miranda or Haskell-like list comprehension with concrete syntax $[M \mid x \in N, P]$, representing "the list of M's, for each element x in the list N which satisfy P". The variable x is bound in M and P, so in abstract syntax this is represented by an operator ***lcomp*** of arity $(1,0,1)$. The operator is strict in its second argument, and the associated rules (where ***cons*** and ***nil*** are constructors) are:

$$\boxed{\begin{array}{c} \dfrac{\mathcal{Y} \mapsto \mathcal{Y}'}{\boldsymbol{lcomp}(\mathcal{X},\mathcal{Y},\mathcal{Z}) \mapsto \boldsymbol{lcomp}(\mathcal{X},\mathcal{Y}',\mathcal{Z})} \\[2ex] \overline{\begin{array}{l} \boldsymbol{lcomp}(\mathcal{X},\boldsymbol{cons}(\mathcal{Y}_1,\mathcal{Y}_2),\mathcal{Z}) \mapsto \\ \quad \boldsymbol{if}(\mathcal{Z}\cdot\mathcal{Y}_1, \boldsymbol{cons}(\mathcal{X}\cdot\mathcal{Y}_1, \boldsymbol{lcomp}(\mathcal{X},\mathcal{Y}_2,\mathcal{Z}), \\ \qquad\qquad\qquad\qquad \boldsymbol{lcomp}(\mathcal{X},\mathcal{Y}_2,\mathcal{Z}))) \end{array}} \\[2ex] \overline{\boldsymbol{lcomp}(\mathcal{X},\boldsymbol{nil},\mathcal{Z}) \mapsto \boldsymbol{nil}} \end{array}}$$

Here the rules assume that ***cons*** is lazy (i.e. $strict(\boldsymbol{cons}) = \emptyset$). The rules for a strict version of ***cons*** differ only in the use of value metavariables in place of metavariables $\mathcal{Y}_1$ and $\mathcal{Y}_2$.

To represent a primitive function like addition, or a length function for strict lists one can give "direct" definitions using the fact that an infinite number of axioms are permitted by the rule format.

Curried operators cannot be directly represented, but can be encoded in terms of lambda abstractions. An example of an operator that cannot be represented by GDSOS operators is *parallel-or* [Plo77], which returns ***true*** if either of its operands can be evaluated to ***true***, and diverges iff both operands diverge. For more on the operational representability of parallel-or for "PCF-like" languages, see [JM91]. Parallel-or is interesting (only) because its addition to the language makes the usual denotational semantics fully-abstract. Since we work directly with the operational orderings, full abstraction is not an issue.

4.1 The Evaluation Relation for GDSOS

The rules of a GDSOS induce a one step transition relation on closed terms of the underlying language L in the obvious way. Let $\mapsto^*$ denote the reflexive and transitive closure of the one step evaluation relation. Convergence is defined as follows:

DEFINITION 4.2 Closed term M converges to value V, written $M{\Downarrow}V$ if and only if $M \mapsto^* V$.

PROPOSITION 4.3 Relation $\Downarrow$ is deterministic (up to $\equiv$).

PROOF. By showing that the conditions on the rules guarantee that $\mapsto$ is strongly confluent the result follows by a simple diagram-chase. $\square$

5 Fixed-Point Properties

In this section we present the "syntactic continuity" property for GDSOS languages with recursion. For the reader interested in an outline of the proofs we include further details in the appendix. The main point of interest in the proofs is that we sketch how GDSOS languages can be reduced to a simpler DSOS format for which $\mapsto$ is deterministic.

Recursion There are many equivalent forms of recursive definition. In what follows we will consider GDSOS languages containing constants as specified by recursion equations. The addition of such constants does not change the expressive power of a GDSOS language[2] but eases the job of proving fixed point properties—in particular since the recursive constants are not variables, so cannot be captured.

A recursion equation $\mathbf{h} \triangleq M$ is taken to be synonymous with a non-constructor $\mathbf{h}$ of arity $()$, with GDSOS rule $\mathbf{h} \mapsto M$. In particular we consider languages in

[2]More precisely, under the assumption that the language already contains some nonterminating term, then the addition of any recursive constants conservatively extends the theory of operational approximation. Interestingly, to prove this conservativity property we first need the syntactic continuity result.

GDSOS format which contain some recursively defined constant

$$\mathbf{f} \triangleq C_{\mathbf{f}}[\mathbf{f}]$$

where we assume that the (closed) context $C_{\mathbf{f}}$ does not contain occurrences of $\mathbf{f}$. Now we also assume a set of constants $\{\mathbf{f}^i\}_{i\geq 0}$ defined inductively by

$$\begin{aligned}\mathbf{f}^0 &\triangleq \mathbf{f}^0\\ \mathbf{f}^{i+1} &\triangleq C_{\mathbf{f}}[\mathbf{f}^i].\end{aligned}$$

We will call such a language a GDSOS *language with recursion.* The functions $\{\mathbf{f}^i\}_{i\geq 0}$ form a chain, bounded above by $\mathbf{f}$:

PROPOSITION 5.1 For any GDSOS language with recursion,

(i) $\mathbf{f} \cong C_{\mathbf{f}}[\mathbf{f}]$.

(ii) For all $i \geq 0$, $\mathbf{f}^i \precsim \mathbf{f}^{i+1} \precsim \mathbf{f}$.

The point of the functions $\{\mathbf{f}^i\}_{i\geq 0}$ is that they completely characterise the behaviour of the function $\mathbf{f}$. This is the essence of the fixed-point induction principle of Scott-style denotational semantics. The main property which justifies this is the following syntactic notion of continuity for the chain $\{\mathbf{f}^i\}_{i\geq 0}$; $\mathbf{f}$ the least upper bound of this chain, and contexts preserve this property:

THEOREM 5.2 (**Syntactic Continuity**)
For any GDSOS language with recursion, for all contexts $C[\,]$, and expressions M we have

$$C[\mathbf{f}] \precsim M \iff \forall i.\, C[\mathbf{f}^i] \precsim M$$

PROOF. Details in the appendix. □

From this we can establish some other standard fixed point properties. We already know that $\mathbf{f}$ satisfies $\mathbf{f} \cong C_{\mathbf{f}}[\mathbf{f}]$. The least-fixed-point property asserts that $\mathbf{f}$ is the least expression (up to $\cong$) satisfying this equation. The following is a slight strengthening:

THEOREM 5.3 (**Least Pre Fixed-Point**)
For any GDSOS language with recursion, if M is a closed expression satisfying $C_{\mathbf{f}}[M] \precsim M$ then $\mathbf{f} \precsim M$.

PROOF. By Theorem 5.2, it is sufficient to show that $\forall i. \mathbf{f}^i \precsim M$. This follows by an easy induction on i; we know that $\mathbf{f}^0 \precsim M$ (Prop 5.1). Suppose $\mathbf{f}^k \precsim M$. Now $\mathbf{f}^{k+1} \cong C_{\mathbf{f}}[\mathbf{f}^k]$, and so by the induction hypothesis and congruence we have $\mathbf{f}^{k+1} \cong C_{\mathbf{f}}[\mathbf{f}^k] \precsim C_{\mathbf{f}}[M]$. □

It is also possible to build up various fixed-point induction principles. The following is a good starting point and is an easy corollary of Theorem 5.2:

COROLLARY 5.4 If for all $n \geq 0$ we have $C[\mathbf{f}^n] \precsim C'[\mathbf{f}^n]$, then $C[\mathbf{f}] \precsim C'[\mathbf{f}]$.

6 Instrumented GDSOS

We introduce *instrumented* versions of GDSOS rules, obtained by labelling the axioms with some *resource* information. This induces a resource-labelled evaluation relation. Resources are partially ordered, and from this we define a resource-based strengthening of operational approximation called *improvement.* M is improved by N (with respect to some instrumented semantics) if in all closing contexts if $C[M]$ terminates then $C[N]$ terminates using fewer resources.

DEFINITION 6.1 (**Resource Structure**)
A *resource structure* is a quadruple $\langle R, \cdot, \mathbf{0}, \geq\rangle$ where $\langle R, \geq\rangle$ is a partially ordered set of resources, and $\langle R, \cdot, \mathbf{0}\rangle$ is a commutative monoid—that is to say, '$\cdot$' (composition of resources) is an associative, commutative operation on R with $\mathbf{0}$ as an identity.

A resource structure is defined to be *monotonic* if $r \geq \mathbf{0}$ for all $r \in R$, and if composition is $\geq$-monotonic, so that $r \geq s \Rightarrow \forall t.\, r \cdot t \geq s \cdot t$.

It is defined to be *well-founded monotonic* (or simply, *well-founded*) if, additionally, $>$ is well-founded (where $r > s \iff r \geq s\ \&\ r \neq s$) and composition is $>$-monotonic.

As an example of a monotonic resource we might take R to be the powerset of non-constructor names, with composition given by set union, zero by the empty set and ordered by subset inclusion. As an example of a strictly monotonic resource we can take $\langle I\!N, +, 0, \geq\rangle$.

We will instrument GDSOS by resource information from a resource structure. The key features of the definition, and how they will be used in the remainder of the article are as follows:

- We use '$\cdot$' to combine the resources of each reduction step to obtain a resource-labelled evaluation relation; the monoid structure in the definition of a resource is just sufficient to make the resulting instrumented evaluation relation deterministic;
- the partial ordering will provide us with a notion of improvement which is a preordering;
- *monotonic* resource structures are those for which we provide certain coinductive proof techniques in Section 8;
- *well-founded monotonic* resources will form the basis of the improvement induction proof technique given in Section 7.

The use of a monotonic resource to instrumenting semantic definitions is fairly natural, and is anticipated by Gurr [Gur91] (there called a *commutative ordered monoid*) in the context of monadic semantics.

DEFINITION 6.2 For a given resource structure, an instrumented GDSOS is defined by labelling each axiom with some resource $r \in R$, and by labelling the transition of each inference rule with a resource (meta)variable ρ thus:

$$\frac{\mathcal{X}_i \overset{\rho}{\mapsto} \mathcal{Y}_i}{p(\mathcal{X}_1, \ldots \mathcal{X}_i, \ldots \mathcal{X}_n) \overset{\rho}{\mapsto} p(\mathcal{X}_1, \ldots \mathcal{Y}_i, \ldots \mathcal{X}_n)}$$

This induces a resource-labelled transition system on terms in the obvious way. Now we define the multiple-step labelled transition relation inductively as follows:

- $M \overset{\mathbf{0}}{\mapsto\!\!\!\twoheadrightarrow} M$ for all M
- $M \overset{s}{\mapsto} N'$ and $N' \overset{t}{\mapsto\!\!\!\twoheadrightarrow} N$ then $M \overset{s \cdot t}{\mapsto\!\!\!\twoheadrightarrow} N$.

Finally, define the resource-labelled convergence relation between closed expressions by:
For all closed M, V, define $M {\Downarrow^r} V$ iff $M \overset{r}{\mapsto\!\!\!\twoheadrightarrow} V$.

It follows easily from the transitivity and commutativity properties of resource composition that for each M there is at most one resource r and value V such that $M {\Downarrow^r} V$. Now for each instrumented GDSOS we define an improvement relation by strengthening the requirements of the operational orderings:

DEFINITION 6.3 With respect to a particular instrumented GDSOS define

(i) M is *improved by* N, $M \underset{\sim}{\trianglerighteq} N$, if for all closing contexts C, if $C[M]{\Downarrow^r} V$ then $C[N]{\Downarrow^s} W$ for some $s \leq r$;

(ii) M is *cost equivalent* to N, $M \underset{\sim}{\lhd\!\rhd} N$, if $M \underset{\sim}{\trianglerighteq} N$ and $N \underset{\sim}{\trianglerighteq} M$.

Certain properties of improvement follow easily from the definition; it is a preorder which is closed under syntactic contexts —in other words, it is a precongruence. Note that for the trivial monotonic resource structure in which R is just a singleton set, the improvement relation degenerates to operational approximation. Note also that improvement implies operational approximation, and cost equivalence implies operational equivalence.

7 Improvement Induction

In this section we present the improvement induction principle which holds for any well-founded instrumented GDSOS. The basic idea of the theorem can be motivated as follows. The least fixed-point principle says that for $\mathbf{f}$ defined by $\mathbf{f} \triangleq C[\mathbf{f}]$ we have

$$M \underset{\sim}{\sqsupseteq} C[M] \Rightarrow M \underset{\sim}{\sqsupseteq} \mathbf{f}$$

We want to find an analogy to this that allows us to establish that $M \underset{\sim}{\sqsubseteq} \mathbf{f}$. What we seek is a relation $\mathcal{A}$ such that

$$M \mathrel{\mathcal{A}} C[M] \Rightarrow M \underset{\sim}{\sqsubseteq} \mathbf{f} \qquad (*)$$

As mentioned in the introduction, taking $\mathcal{A}$ to be simply operational approximation or equivalence is not sufficient[3] — for example if C is the trivial context $[\,]$ and $\mathbf{f}$ therefore never terminates, then this would allow us to show $M \underset{\sim}{\sqsubseteq} \mathbf{f}$ for all M!

Rather than focussing on syntactic properties of C, the improvement induction principle provides a relation $\mathcal{A}$ (or rather, a set of nontrivial such relations) which makes $(*)$ hold. The relation in question is based on the improvement relation of some well-founded instrumented GDSOS. Assume that we have an instrumented semantics over some well-founded resource structure for which $\mathbf{f} \overset{r}{\mapsto} C[\mathbf{f}]$ for some $r > \mathbf{0}$. Then the following property holds:

$$\begin{aligned} M \underset{\sim}{\trianglerighteq} \overset{r}{\mapsto\!\!\!\twoheadrightarrow} \underset{\sim}{\trianglerighteq} C[M] \quad &\Rightarrow \quad M \underset{\sim}{\trianglerighteq} \mathbf{f} \\ &\Rightarrow \quad M \underset{\sim}{\sqsubseteq} \mathbf{f} \end{aligned}$$

where $\underset{\sim}{\trianglerighteq} \overset{r}{\mapsto\!\!\!\twoheadrightarrow} \underset{\sim}{\trianglerighteq}$ is just composition of the relations, so $M \underset{\sim}{\trianglerighteq} \overset{r}{\mapsto\!\!\!\twoheadrightarrow} \underset{\sim}{\trianglerighteq} N$ iff there exist N_1, N_2 such that

$$M \underset{\sim}{\trianglerighteq} N_1,\ N_1 \overset{r}{\mapsto\!\!\!\twoheadrightarrow} N_2, \text{ and } N_2 \underset{\sim}{\trianglerighteq} N.$$

The improvement induction theorem gives a slightly more general variant of the above property, by removing the dependence on the recursive constants $\mathbf{f}$. First we introduce some new notation.

Define the following relations on expressions:

$$\begin{aligned} \underset{\sim}{\trianglerighteq}^{r} &\overset{\text{def}}{=} \underset{\sim}{\trianglerighteq} \overset{r}{\mapsto\!\!\!\twoheadrightarrow} \underset{\sim}{\trianglerighteq} \\ \underset{\sim}{\lhd\!\rhd}^{r} &\overset{\text{def}}{=} \underset{\sim}{\lhd\!\rhd} \overset{r}{\mapsto\!\!\!\twoheadrightarrow} \underset{\sim}{\lhd\!\rhd}. \end{aligned}$$

Intuitively, $M \underset{\sim}{\trianglerighteq}^{r} N$ says that M is improved by N and that any evaluation of N would be "faster" than M by at least r. If $r \geq s$ then we have that

$$\underset{\sim}{\trianglerighteq} = \underset{\sim}{\trianglerighteq}^{0} \supseteq \underset{\sim}{\trianglerighteq}^{s} \supseteq \underset{\sim}{\trianglerighteq}^{r}$$

Using this terminology we state the main theorem:

THEOREM 7.1 (**Improvement Induction**) For any well-founded instrumented GDSOS, for all closed C, M, N and all resources $t > 0$,

$$\left.\begin{array}{rll} M & \underset{\sim}{\trianglerighteq}^{t} & C[M] \\ \text{and } N & \underset{\sim}{\lhd\!\rhd}^{t} & C[N] \end{array}\right\} \Rightarrow M \underset{\sim}{\trianglerighteq} N.$$

[3] One way around this problem is to restrict attention to only certain classes of context C for which this *does* hold; this technique is used extensively in process algebra (e.g., see [BW90]) but is rarely used in the functional setting — although a few related techniques have been described — see e.g., [Car84] and [Cou79].

EXAMPLE 7.2 Consider an instrumented version of the lazy lambda calculus over the well-founded resource structure $\langle I\!N, +, 0, \geq \rangle$, in which each axiom is labelled with 1 (i.e., each reduction step costs 1). We will show that

$$\mathbf{fix} \cong (\lambda x.\lambda f.f\,(x\,x\,f))(\lambda x.\lambda f.f\,(x\,x\,f))$$

where $\mathbf{fix} \triangleq \lambda f.f(\mathbf{fix}\,f)$.

Let P be the expression $\lambda x.\lambda f.f\,(x\,x\,f)$. Note that $P\,P \overset{1}{\mapsto} \lambda f.f\,(P\,P\,f)$, and hence that

$$P\,P \cong \lambda f.f\,(P\,P\,f).$$

It follows by the least fixed-point property (Theorem 5.3) that $\mathbf{fix} \mathrel{\underset{\sim}{\sqsubset}} P\,P$. To prove the reverse inclusion we use improvement induction. From the above reduction we see that $P\,P \mathrel{\underset{\sim}{\triangleright}}^1 \lambda f.f\,(P\,P\,f)$, and from the definition of $\mathbf{fix}$ we know that $\mathbf{fix} \mathrel{\underset{\sim}{\triangleleft\!\triangleright}}^1 \lambda f.f(\mathbf{fix}\,f)$. By improvement induction we conclude that $P\,P \mathrel{\underset{\sim}{\triangleright}} \mathbf{fix}$, and hence that $P\,P \cong \mathbf{fix}$.

In the above example the $\mathrel{\underset{\sim}{\triangleright}}^r$ -property needed to apply improvement induction is established very simply by the fact that $\overset{r}{\mapsto\!\!\!\rightarrow} \subseteq \mathrel{\underset{\sim}{\triangleright}}^r$. In our experience, only relatively simple properties need be established to apply improvement induction to good effect. One factor that aids reasoning is if there exist closed contexts C_r with the property that for all N, $C_r[N] \overset{r}{\mapsto\!\!\!\rightarrow} N$. In this case we can reduce reasoning about $\mathrel{\underset{\sim}{\triangleright}}^r$ to reasoning about $\mathrel{\underset{\sim}{\triangleright}}$, since $M \mathrel{\underset{\sim}{\triangleright}}^r N \iff M \mathrel{\underset{\sim}{\triangleright}} C_r[N]$. In the case of applications of the Improvement Theorem (below), the C_r are just r-fold compositions of an identity function, and the specific improvement laws for C_r are what are known as the "tick algebra" of [San96b].

The Improvement Theorem The Improvement Theorem from [San96b] is a simple corollary of improvement induction:

COROLLARY 7.3 (**Improvement Theorem**) For any instrumented GDSOS over the well-founded resource structure $\langle I\!N, 0, +, \geq \rangle$, if all the axioms for recursive constants are labelled by some resource $n > 0$ then we have the following:

If $\mathbf{g} \triangleq M$ and $M \mathrel{\underset{\sim}{\triangleright}} C[\mathbf{g}]$ then $\mathbf{g} \mathrel{\underset{\sim}{\triangleright}} \mathbf{h}$, where $\mathbf{h} \triangleq C[\mathbf{h}]$.

PROOF. Since $\mathbf{g} \overset{n}{\mapsto} M$ (by definition) and $M \mathrel{\underset{\sim}{\triangleright}} C[\mathbf{g}]$, we have that $\mathbf{g} \mathrel{\underset{\sim}{\triangleright}}^n C[\mathbf{g}]$. Also by definition, $\mathbf{h} \overset{n}{\mapsto} C[\mathbf{h}]$, and since $\mathrel{\underset{\sim}{\triangleleft\!\triangleright}}$ is reflexive, we have $\mathbf{h} \mathrel{\underset{\sim}{\triangleleft\!\triangleright}}^n C[\mathbf{h}]$. Now we have, directly from Theorem 7.1 that $\mathbf{g} \mathrel{\underset{\sim}{\triangleright}} \mathbf{h}$. □

The significant advance of Corollary 7.3 is that here it is established for any GDSOS language. A minor variation allows n to be different for each recursive constant; if $n_{\mathbf{g}} \geq n_{\mathbf{h}} > 0$ then the Improvement Theorem still holds. This verifies a conjecture in [San95a] about so-called *weighted improvement relations*, which are used to establish the correctness of a higher-order variant of Scherlis' *expression procedure* transformation framework. The Improvement Theorem for this version of improvement provides a much simpler correctness proof than that presented in [San95a].

Space does not permit an illustration of the utility of the Improvement Theorem in establishing the correctness of program transformations; we refer the reader to [San96b, San96a] for a number of substantial applications.

7.1 Open Improvement Induction

The improvement induction principle is stated for closed expressions and contexts. Our proof extends to open expressions in two important special cases, according to how improvement behaves under the application of substitutions.

THEOREM 7.4 The Improvement Induction Principle (as stated in Theorem 7.1) holds for open C, M and N in the following case:

(i) if $\mathrel{\underset{\sim}{\triangleright}}$ is closed under substitution (i.e. $M \mathrel{\underset{\sim}{\triangleright}} N$ implies $M\sigma \mathrel{\underset{\sim}{\triangleright}} N\sigma$ for all σ), or

(ii) if $\mathrel{\underset{\sim}{\triangleright}}$ is closed under value-substitutions (substitutions of values for variables) and the language is in call-by-value GDSOS format.

A sufficient condition for improvement to be closed under substitution is if a call-by-name let-expression is expressible (up to operational equivalence) in the language; $\mathrel{\underset{\sim}{\triangleright}}$ is closed under value substitutions providing a call-by-value let-expression is expressible.

8 Coinductive Proof Techniques for GDSOS

We have established a number of proof techniques for operational approximation and equivalence which hold for languages whose operators are in GDSOS format. The proofs of these theorems work directly from the definition of operational approximation (although we do simplify the GDSOS format somewhat, as outlined in the appendix); however, in applications of the theorems we need to prove (hopefully much simpler) properties about operational approximation and improvement in order to make use of the theorems. For example, in Example 7.2 we tacitly assumed that $\mapsto \subseteq \cong$, but we did not actually prove it. Proving such "simple" properties directly from the definitions is certainly possible, but somewhat tedious. Here we establish a simpler (but not necessarily complete) method for reasoning about improvement (and hence also operational approximation) for any monotonic instrumented GDSOS language. The basic idea is now well-established (see e.g., [Abr90, How89, Gor95, Pit99]): to establish $M \gtrsim N$ it is sufficient to find a certain coinductive relation which relates M and N.

DEFINITION 8.1 A binary relation $\mathcal{I}$ on expressions is an *open improvement simulation* (respectively, an *open value improvement simulation*) if whenever $M \mathcal{I} N$ then for all closing substitutions σ (respectively, closing *value* substitutions), if

$$\begin{array}{rl} & M\sigma{\Downarrow}^r \boldsymbol{c}((\bar{x}_1)M_1, \ldots, (\bar{x}_n)M_n) \\ \text{then} & N\sigma{\Downarrow}^s \boldsymbol{c}((\bar{x}_1)N_1, \ldots, (\bar{x}_n)N_n) \end{array}$$

for some s, and $N_1 \ldots N_n$ such that $r \geq s$ and $M_i \mathcal{I} N_i$, $i \in 1 \ldots n$.

THEOREM 8.2

(i) For any monotonic instrumented GDSOS if $\mathcal{I}$ is an open improvement simulation then $\mathcal{I} \subseteq \gtrsim$.

(ii) For any monotonic instrumented call-by-value GDSOS if $\mathcal{I}$ is an open value improvement simulation then $\mathcal{I} \subseteq \gtrsim$.

PROOF. In both cases it is sufficient to prove that the maximal improvement simulation, given by the union of all improvement simulations, is a precongruence relation. This is established by instantiating the main result of our earlier work [San91], which extends Howe's proof techniques [How89, How96] to handle improvement relations. □

The proof technique also applies to operational approximation, taking the trivial (necessarily monotonic) instrumentation where the resource is just a singleton set. Unfortunately, the proof technique is not always complete; for some instrumented GDSOS languages there are pairs of closed terms which are not contained in any improvement simulation, but which are in the improvement relation. General conditions for completeness are somewhat difficult to establish, although for the special case of operational approximation completeness amounts to showing that there are sufficiently many "destructors" for each constructor — see [How96] for a precise formulation.

Bisimulation upto Improvement and Context

In [San97] we described a bisimulation-like proof technique for equivalence based on the Improvement Theorem of [San96b], with something of the flavour of Sangiorgi's "bisimulation up to context and up to expansion" for the pi-calculus [San95b, San94], where "expansion" is analogous to an improvement relation based on the number of silent transitions a process can perform. It seems that a similar development to [San97] can be carried out in the setting of a well-founded GDSOS, and roughly speaking, amounts to generalising the improvement induction principle from a pair of expressions to a possibly infinite set of pairs. Such a development would put improvement induction on a bisimulation-like footing, but we leave it for future investigation.

9 Conclusion and Further Work

We have established a number of proof principles for functional languages whose validity for any particular language can be established by simply considering the syntactic form of the SOS rules. These principles include the standard "denotational" properties which characterise a recursive definition in terms of its finite unwindings, as well as the standard operational techniques based on (bi)simulations and coinduction. We have also introduced a new operational proof technique, improvement induction, of which the Improvement Theorem is a corollary.

In future work we hope to investigate more expressive formats, possibly dealing with control operators like *callcc*, with bounded nondeterminism, and with operations acting on state. It may be natural in these settings to consider alternative semantic styles, such as natural semantics, or evaluation-context style (see the appendix for an example). A promising starting point is the recent metatheory for languages with control and effects described by Talcott [Tal97]. Talcott describes some properties which must hold of a reduction relation which guarantee that a form of context lemma is sound. The abstract properties given are closely related to certain fundamental technical properties that we establish (Proposition A.6)for the simpler DSOS format in the appendix. The class of languages considered are call-by-value lambda calculi extended with binding-free, completely strict operators. It would be interesting to see if an improvement induction principle can be established for Talcott's class of languages. It would also be interesting to provide a syntactic "front-end", in the form of a rule format, for Talcott's work.

Other "metatheory" worth investigating includes *abstract interpretation* [CC92, Sch95, GD95] and call-by-need theories [Lau93, AFM$^+$95].

Acknowledgements The more basic DSOS rule format given in the appendix was communicated to the author by Davide Sangiorgi (INRIA) [personal communication], who proposed the format during an extended discussion on proof techniques for functional programs (in particular the Improvement Theorem) and the relation to proof techniques for the pi-calculus. Thanks are due to Davide for his valuable contribution. Thanks are also due to Andrew Moran, Caroline Talcott and the anonymous POPL referees for several useful suggestions and criticisms on an earlier draft.

References

[Abr90] S. Abramsky. The lazy lambda calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison Wesley, 1990.

[ABV94] Luca Aceto, Bard Bloom, and Frits Vaandrager. Turning SOS rules into equations. *Information and Computation*, 111(1):1–52, 15 May 1994.

[AFM$^+$95] Z. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. The call-by-need lambda calculus. In *The 22nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '95)*, New York, 1995. ACM Press.

[BIM95] Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, January 1995.

[Blo90] Bard Bloom. Can LCF be topped? Flat lattice models of typed λ-calculus. *Information and Computation*, 87(1/2):263–300, jul / aug 1990.

[BW90] J. C. M. Baeten and W. P. Weijland. *Process Algebra*, volume 18 of *Tracts in Theoretical Computer Science*. Cambridge Univ. Press, 1990.

[Car84] Robert Cartwright. Recursive programs as definitions in first order logic. *Siam Journal of Computing*, 13(2):374–408, May 1984.

[CC92] P. Cousot and R. Cousot. Inductive definitions, Semantics and Abstract Interpretation. In *19th POPL, Albuquerque, New Mexico*, pages 83–94. ACM Press, January 1992.

[Cou79] B. Courcelle. Infinite trees in normal form and recursive equations having a unique solution. *Mathematical Systems Theory*, 13(1):131–180, 1979.

[Dam94] L. Dami. *Software Composition: Towards an Integration of Functional and Object-Oriented Approaches.* PhD thesis, Faculté des Sciences Économiques et Scociales, Université de Genève, 1994.

[FFK87] M. Felleisen, D. Friedman, and E. Kohlbecker. A syntactic theory of sequential control. *Theoretical Computer Science*, 52(1):205–237, 1987.

[FV95] W. Fokkink and C. Verhoef. A conservative look at term deduction systems with variable bindings. Technical Report Utrecht Logic Group Preprint 140 / Eindhoven Computer Science report 95-28, Utrecht University/Eindhoven University of Technology, 1995.

[GD95] V. Gouranton and D. Le Métayer. Derivation of static analysers of functional programs from path properties of a natural semantics. Rapport de recherche 2607, INRIA, Rennes, 1995.

[Gor95] A. D. Gordon. Bisimilarity as a theory of functional programming. Technical Report BRICS NS-95-3, BRICS, Aarhus University, Denmark, 1995. Preliminary version in MFPS'95.

[Gur91] D. Gurr. *Semantic Frameworks for Complexity.* PhD thesis, Department of Computer Science, Edinburgh, 1991. (Available as reports CST-72-91 and ECS-LFCS-91-130).

[GV92] J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, October 1992.

[How89] D. J. Howe. Equality in lazy computation systems. In *Fourth annual symposium on Logic In Computer Science*, pages 198–203. IEEE, 1989.

[How91] D. J. Howe. On computational open-endedness in Martin-Löf's type theory. In *Sixth annual symposium on Logic In Computer Science*, pages 162–172, 1991.

[How96] D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, February 1996.

[JM91] T. Jim and A. R. Meyer. Full abstraction and the context lemma (preliminary report). In *STACS*. LNCS 526, 1991. (Full version to appear in *Siam J. Comp*, 1996).

[Klo80] J.W. Klop. *Combinatory Reduction Systems*, volume 127 of *Mathematical Centre Tracts*. Mathematischen Centrum, 413 Kruislaan, Amsterdam, 1980.

[Lau93] J. Launchbury. A natural semantics for lazy evaluation. In *Conference record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, (POPL'92)*, pages 144–154. ACM Press, 1993.

[MST96] I. A. Mason, S. Smith, and C. L. Talcott. From Operational Semantics to Domain Theory. *Information and Computation*, 1996. to appear.

[MT91] I. Mason and C. Talcott. Equivalence in functional languages with effects. *Journal of Functional Programming*, 1(3):287–327, July 1991.

[Nie96] K. Nielsen. A unified approach to partial evaluation and deforestation. Master's thesis, DIKU, University of Copenhagen, September 1996.

[NPS90] Bengt Nordström, Kent Petersson, and Jan M. Smith. *Programming in Martin-Löf's Type Theory: An Introduction*, volume 7 of *International Series of Monographs on Computer Science*. Oxford University Press, 1990.

[PE88] F. Pfenning and C. Elliott. Higher-order abstract syntax. In *Proceedings of the SIGPLAN '88 Conference on Programming Lanugage Design and Implementation (SIGPLAN '88)*, pages 199–208. ACM Press, June 1988.

[Pit99] A. M. Pitts. Operationally-based theories of program equivalence. In P. Dybjer and A. M. Pitts, editors, *Semantics and Logics of Computation*. Cambridge University Press, 199? Based on lectures given at the CLICS-II Summer School on Semantics and Logics of Computation, Isaac Newton Institute for Mathematical Sciences, Cambridge UK, September 1995.

[Plo77] G. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5(3):223–256, 1977.

[Plo81] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN–19, Computer Science Department, Aahus University, Denmark, September 1981.

[San91] D. Sands. Operational theories of improvement in functional languages (extended abstract). In *Proceedings of the Fourth Glasgow Workshop on Functional Programming*, pages 298–311, Skye, August 1991. Springer Workshop Series.

[San94] D. Sangiorgi. Locality and non-interleaving semantics in calculi for mobile processes. Technical report, LFCS, University of Edinburgh, Edinburgh, U.K., 1994.

[San95a] D. Sands. Higher-order expression procedures. In *Proceeding of the ACM SIGPLAN Syposium on Partial Evaluation and Semantics-Based Program Manipulation, PEPM'95*, pages 190–201, New York, 1995. ACM.

[San95b] D. Sangiorgi. Lazy functions and mobile processes. Rapport de recherche 2515, INRIA Sophia Antipolis, 1995.

[San96a] D. Sands. Proving the correctness of recursion-based automatic program transformations. *Theoretical Computer Science*, A(167), October 1996. Preliminary version in TAPSOFT'95, LNCS 915.

[San96b] D. Sands. Total correctness by local improvement in the transformation of functional programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 18(2):175–234, March 1996.

[San97] D. Sands. Improvement theory and its applications. In A. Gordon and A. Pitts, editors, *Higher-Order operational Techniques in Semantics.* Cambridge University Press, 1997. (to appear).

[Sch95] D. A. Schmidt. Natural-semantics-based abstract interpretation. In *Proc. 2d Static Analysis Symposium*, volume 983 of *LNCS*, pages 1–18. Springer-Verlag, 1995.

[Smi92] Scott Smith. From operational to denotational semantics. In *Conference on Mathematical Foundations of Programming Language Semantics*, volume 598 of *Lecture Notes in Computer Science.* Springer-Verlag, 1992.

[Tal85] C. L. Talcott. *The Essence of Rum, A Theory of the intensional and extensional aspects of Lisp-type computation.* PhD thesis, Stanford University, August 1985.

[Tal97] C. Talcott. Reasoning about functions with effects. In A. Gordon and A. Pitts, editors, *Higher-Order Operational Techniques in Semantics.* Cambridge University Press, 1997. (to appear).

A Technical Development

In this section we present highlights of the technical development, leading to sketch proofs of the main theorems, syntactic continuity and improvement induction.

The first part of the technical development is to reduce the task of proving theorems about GDSOS to that of proving corresponding theorems about a simpler rule format which we call DSOS. DSOS is a subset of GDSOS in which the one-step evaluation relation is deterministic, and all constructors are lazy.

In order that theorems about DSOS can be applied to GDSOS we need to show that there are fully-abstract translations from any GDSOS language to a DSOS language. The result of these constructions is the following: in proofs of syntactic continuity and improvement induction rules we can assume, without loss of generality, that the rules are in DSOS format.

A.1 GDSOS reduces to lazy-GDSOS

A GDSOS language is in *lazy-*GDSOS *format* if all constructors are lazy (strict in zero arguments). The following proposition states that there is a fully abstract translation from any instrumented GDSOS language to an instrumented lazy-GDSOS language.

PROPOSITION A.1 For any language L in instrumented GDSOS format there exists a language K in lazy-GDSOS format and a compositional translation, $\overline{\cdot}$, from $T(L)$ to $T(K)$ such that

$$\forall M, N \in T(L). \quad M \underset{\sim}{\triangleright}_L N \iff \overline{M} \underset{\sim}{\triangleright}_K \overline{N}$$

The proof is by a uniform construction of a language K and a translation from $T(L)$ to $T(K)$. The general construction is rather technical so we will not present it here. But the basic idea is to introduce a new lazy constructor for the "constructor components" of every value (c.f. Nielsen's *value skeletons* [Nie96]); the nonstrict parts form the respective (lazy) arguments.

A.2 lazy-GDSOS reduces to DSOS

We outline how any language in lazy-GDSOS format can be systematically represented by an extended language (that is to say, a language with some additional operators) whose SOS rules are in a simpler format: *Deterministic Structural Operational Semantics.* This can be done in such a way that the evaluation relation, and the operational approximation (improvement) relation for the extended language are conservative extensions of those relations for the underlying language.

DEFINITION A.2 (DSOS **format**) A language is in DSOS format, if it is in lazy-GDSOS format, and if each operator is strict in either zero or one argument (in which we assume that it is strict in the first argument).

As a simple consequence of the definition, the one-step transition relation for a DSOS language is deterministic. For example, the application rule for the lazy lambda-calculus is in DSOS format, since it is strict in one argument.

The idea is that any GDSOS language can be extended such that all the rules of the extended language are in DSOS format. For example, the call-by-value application operator is strict in both arguments:

$$\boxed{\frac{\mathcal{Z} \mapsto \mathcal{Z}'}{@(\mathcal{Z},\mathcal{Y}) \mapsto @(\mathcal{Z}',\mathcal{Y})} \qquad \frac{\mathcal{Z} \mapsto \mathcal{Z}'}{@(\mathcal{X},\mathcal{Z}) \mapsto @(\mathcal{X},\mathcal{Z}')} \qquad \frac{}{@(\lambda\mathcal{X},\mathcal{V}) \mapsto \mathcal{X}\cdot\mathcal{V}}}$$

To represent this by a DSOS language, we add a new operator $@_\lambda$ of arity $(0,1)$. The DSOS rules in the extended language are:

$$\boxed{\frac{\mathcal{Z} \mapsto \mathcal{Z}'}{@(\mathcal{Z},\mathcal{Y}) \mapsto @(\mathcal{Z}',\mathcal{Y})} \qquad \frac{}{@(\lambda\mathcal{X},\mathcal{Y}) \mapsto @_\lambda(\mathcal{Y},\mathcal{X})}} \quad \boxed{\frac{\mathcal{Z} \mapsto \mathcal{Z}'}{@_\lambda(\mathcal{Z},\mathcal{X}) \mapsto @_\lambda(\mathcal{Z}',\mathcal{X})} \qquad \frac{}{@_\lambda(\mathcal{V},\mathcal{X}) \mapsto \mathcal{X}\cdot\mathcal{V}}}$$

This idea extends uniformly to all operators which are strict in more than one argument, and this provides the basis for the following:

PROPOSITION A.3 Any language L whose operators are in (instrumented) GDSOS format can be represented by a language L' in DSOS format such that L' includes all the operators in L, and such that for all M, $N \in T(L)$, $M \underset{\sim}{\triangleright}_L N \iff M \underset{\sim}{\triangleright}_{L'} N$.

We omit the details of the proof, but it is based on a (compositional) encoding like the one above, with the slight difference that we eliminate all value metavariables (like $\mathcal{V}$ above) in favour of a rule for each possible constructor. The idea of the additional operators is that they "remember" the constructors of the values which have been evaluated so far; the arguments of the additional operators include one argument for each operand of these constructors.

A.3 Context Notation and Reduction Contexts

The definition of operational approximation and equivalence involve the notions of a context. The definition of a context is the usual one. In particular the holes in a context occur in place of *terms*, and not arbitrary operands (abstractions). As is also usual, a context may capture free variables in the term placed in its holes. In reasoning directly about the operational orderings for a language in DSOS format we will do some proofs directly involving manipulation of contexts. For this we need a more general form of context in which more than one distinct type of hole may occur. These are sometimes called *polyadic* contexts. In fact for present purposes we will be able to make do with just two distinct "holes", denoted by $[\,]$ and $\langle\,\rangle$.

Contexts C, C' etc. will now denote contexts containing zero or more occurrences of two distinct types of hole, $[\,]$ and $\langle\,\rangle$. We will adopt the convention that when we write $C\langle\,\rangle$ we are indicating that context C contains no occurrences of the hole $[\,]$, and vice-versa. As an example, if C is the context $(\lambda x.[\,])\langle\,\rangle$ then $C[x]\langle \boldsymbol{true}\rangle$ is the term $(\lambda x.x)\,\boldsymbol{true}$.

Reduction Contexts An alternative (but equivalent) presentation of the one step evaluation relation is in terms of a simple form of contexts containing a single occurrence of a single hole, known as *reduction context* [FFK87] (or *evaluation contexts*). A reduction context is used to specify the position in a term where the next reduction step can be performed. As an example, for the language in Figure 1 the reduction contexts $\mathcal{R}$ are contexts given inductively by: $\mathcal{R} ::= [\,] \mid \mathcal{R}\,M \mid \textbf{if}\ \mathcal{R}\ \textbf{then}\ M\ \textbf{else}\ N$. We will use a reduction context presentation of the rules to simplify some of our reasoning.

Definition A.4 The reduction contexts for a language in DSOS format are defined inductively as follows:

- $[\,]$ is a reduction context;
- if $\mathcal{R}$ is a reduction context, and p is a nonconstructor operator strict in its first argument, then a well-formed context of the form $p(\mathcal{R}, (\vec{x}_2)M_2, \ldots, (\vec{x}_n)M_n)$ is a reduction context.

Given this definition, it can be seen that the transition relation generated by the rules of a DSOS can be equivalently represented by the collection of axioms

$$\{\mathcal{R}[\mathcal{M}] \mapsto \mathcal{R}[\mathcal{N}] \mid \mathcal{M} \mapsto \mathcal{N} \text{ is an axiom of the DSOS }\}.$$

The determinacy of the system guarantees that if $M \mapsto N$ then there exists a unique reduction context $\mathcal{R}$, and unique terms M' and N' such that $M \equiv \mathcal{R}[M']$, $N \equiv \mathcal{R}[N']$, and such that $M' \mapsto N'$ is an instance of an axiom of the DSOS (we say that M' is a *redex*).

Reduction Contexts with Holes A reduction step of a term of the form $C\langle M\rangle$, if $M \notin \mathit{Val}$, satisfies the following informal property: either the reduction step does not depend on M (it is uniform in M) or it occurs inside M. This property holds in any language in DSOS format, and captures a common case analysis used (usually informally) when reasoning directly about operational approximation. The property is formalised below.

The following defines a particular set of contexts – namely reduction contexts with additional $\langle\,\rangle$-holes:

Definition A.5 Let $\hat{\mathcal{R}}$, $\hat{\mathcal{R}}'$, the *reduction contexts with holes*, be defined as follows:

- $[\,]$ is a reduction context with holes;
- if p is a nonconstructor of arity $(0, k_2, \ldots k_n)$ which is strict in one argument (the first argument), and $\vec{x}_i$, $i \in \{2, \ldots, n\}$ is a vector of k_i distinct variables, then
$$p(\hat{\mathcal{R}}, (\vec{x}_2).C_2\langle\,\rangle, \ldots (\vec{x}_n).C_n\langle\,\rangle)$$
is a reduction context with holes.

Lemma A.6 For any instrumented DSOS language, for all contexts $C\langle\,\rangle$, and all closed expressions $M \notin \mathit{Val}$, if $C\langle M\rangle \overset{r}{\mapsto} M'$, then there exists contexts $\hat{\mathcal{R}}$, $C'\langle\,\rangle$ such that $C \equiv \hat{\mathcal{R}}\langle\,\rangle[C']$ and exactly one of the following two properties hold:

(i) either the reduction is uniform in M:
there exists a context C'', such that for all closed N, $C'\langle N\rangle \overset{r}{\mapsto} C''\langle N\rangle$

(ii) or the reduction is "inside" M: $C' = \langle\,\rangle$ and $M \overset{r}{\mapsto}$.

The proof is by induction on the context C.

A.4 Proof of Syntactic Continuity

Recalling Theorem 5.2 we need to show

$$C\langle \mathbf{f}\rangle \sqsubseteq M \iff \forall i.\, C\langle \mathbf{f}^i\rangle \sqsubseteq M$$

where $\mathbf{f} \triangleq C\langle \mathbf{f}\rangle$, $\mathbf{f}^0 \triangleq \mathbf{f}^0$ and $\mathbf{f}^{i+1} \triangleq C\langle \mathbf{f}^i\rangle$. By Proposition A.3, it is sufficient to assume that the language is in DSOS format.

($\Rightarrow$) Follows from Prop 5.1, together with the congruence property of $\sqsubseteq$.

($\Leftarrow$) Under the assumption that $\forall i.\, C\langle \mathbf{f}^i\rangle \sqsubseteq M$ we will show that for all closing contexts $C_0[\,]$, and all natural numbers n, if $C_0[C\langle \mathbf{f}\rangle]$ converges in n steps then $C_0[M]$ converges. From the assumption, and the congruence property of $\sqsubseteq$, it will be sufficient to show that $C_0[C\langle \mathbf{f}^n\rangle]$ converges (intuitively, if the computation takes n steps, then we can't need more than n unwindings of the recursion), and this we do by induction on n.

Let $C_1\langle\ \rangle$ denote the context $C_0[C]$.

Base: $n = 0$ Then $C_1\langle \mathbf{f}\rangle \in \mathit{Val}$, and hence C_1 must have an outermost constructor. Hence $C_1\langle \mathbf{f}^n\rangle \in \mathit{Val}$.

Induction: $n > 0$ In this case we have $C_1\langle \mathbf{f}\rangle \mapsto N$ for some N such that N converges in $n-1$ steps. We proceed by case analysis according to Lemma A.6. C_1 can be written as $\hat{\mathcal{R}}\langle\ \rangle[C']$ for some $\hat{\mathcal{R}}$ and $C'\langle\ \rangle$. The Lemma gives us two cases, corresponding to whether $\mathbf{f}$ is involved in the reduction step or not. In the first case there exists a C'' such that for all closed L, $C'\langle L\rangle \mapsto C''\langle L\rangle$. So in particular we have that

$$\begin{array}{rl} C_1\langle \mathbf{f}\rangle \equiv \hat{\mathcal{R}}\langle \mathbf{f}\rangle[C'\langle \mathbf{f}\rangle] & \mapsto \hat{\mathcal{R}}\langle \mathbf{f}\rangle[C''\langle \mathbf{f}\rangle] \equiv N \\ \text{and}\quad C_1\langle \mathbf{f}^n\rangle \equiv \hat{\mathcal{R}}\langle \mathbf{f}^n\rangle[C'\langle \mathbf{f}^n\rangle] & \mapsto \hat{\mathcal{R}}\langle \mathbf{f}^n\rangle[C''\langle \mathbf{f}^n\rangle]. \end{array}$$

Now since $\hat{\mathcal{R}}\langle \mathbf{f}\rangle[C''\langle \mathbf{f}\rangle]$ converges in $n-1$ steps, by the induction hypothesis we conclude that $\hat{\mathcal{R}}\langle \mathbf{f}^{n-1}\rangle[C''\langle \mathbf{f}^{n-1}\rangle]{\Downarrow}$.

Since $\mathbf{f}^{n-1} \underset{\sim}{\sqsubset} \mathbf{f}^n$ we can conclude by congruence that $\hat{\mathcal{R}}\langle \mathbf{f}^n\rangle[C''\langle \mathbf{f}^n\rangle]{\Downarrow}$, i.e., that $C_1\langle \mathbf{f}^n\rangle{\Downarrow}$.

In the second case we have $C' = \langle\ \rangle$ and hence the redex is $\mathbf{f}$ itself. By definition, $\mathbf{f} \mapsto C_{\mathbf{f}}\langle \mathbf{f}\rangle$ and $\mathbf{f}^n \mapsto C_{\mathbf{f}}\langle \mathbf{f}^{n-1}\rangle$. So in this case we have

$$\begin{array}{rl} C_1\langle \mathbf{f}\rangle \equiv \hat{\mathcal{R}}\langle \mathbf{f}\rangle[\mathbf{f}] & \mapsto \hat{\mathcal{R}}\langle \mathbf{f}\rangle[C_{\mathbf{f}}\langle \mathbf{f}\rangle] \equiv N \\ \text{and}\quad C_1\langle \mathbf{f}^n\rangle \equiv \hat{\mathcal{R}}\langle \mathbf{f}^n\rangle[\mathbf{f}^n] & \mapsto \hat{\mathcal{R}}\langle \mathbf{f}^n\rangle[C_{\mathbf{f}}\langle \mathbf{f}^{n-1}\rangle]. \end{array}$$

But $\hat{\mathcal{R}}\langle \mathbf{f}\rangle[C_{\mathbf{f}}\langle \mathbf{f}\rangle]$ converges in $(n-1)$-steps, and so by the induction hypothesis we have $\hat{\mathcal{R}}\langle \mathbf{f}^{n-1}\rangle[C_{\mathbf{f}}\langle \mathbf{f}^{n-1}\rangle]{\Downarrow}$ which implies that $\hat{\mathcal{R}}\langle \mathbf{f}^n\rangle[C_{\mathbf{f}}\langle \mathbf{f}^{n-1}\rangle]{\Downarrow}$. Hence we can conclude that $C_1\langle \mathbf{f}^n\rangle{\Downarrow}$ as required. □

A.5 Proof of Improvement Induction

We will make use of the following notations:

- $M{\Downarrow}^r \iff \exists V.\, M{\Downarrow}^r V$
- $M{\Downarrow}^{r \le s} \iff M{\Downarrow}^r \;\&\; r \le s$
- $M{\Downarrow}^{\le s} \iff \exists r.\, M{\Downarrow}^r \;\&\; r \le s$

We will need the following simple properties about the improvement relation over any instrumented DSOS language:

LEMMA A.7 For all closed M and N

(i) $M \underset{\sim}{\triangleright}^r N \Rightarrow \mathcal{R}[M] \underset{\sim}{\triangleright}^r \mathcal{R}[N]$

(ii) $(M \underset{\sim}{\triangleright}^r N \;\&\; M{\Downarrow}^s) \Rightarrow N{\Downarrow}^t$ where $s \ge r \cdot t$

(iii) $M \underset{\sim}{\triangleleft\!\triangleright}^r N \Rightarrow (M{\Downarrow}^{t \cdot r} \iff N{\Downarrow}^t)$

The proofs are straightforward from the definition of $\underset{\sim}{\triangleright}^r$ and the fact that improvement is a congruence.

Now we proceed to the proof of the improvement induction principle. From the assumptions that $M \underset{\sim}{\triangleright}^r C_0\langle M\rangle$ and $N \underset{\sim}{\triangleleft\!\triangleright}^r C_0\langle N\rangle$ for some $r > \mathbf{0}$, we are required to show that $M \underset{\sim}{\triangleright} N$. We prove this by showing that

> for all closed contexts C, if $C\langle M\rangle{\Downarrow}^u$ in l reduction steps then $C\langle N\rangle{\Downarrow}^{\le u}$

by lexicographic induction on $\langle u, l\rangle$ (recalling that u is an element of a well-founded resource structure).

Assume for some arbitrary closed C, that $C\langle M\rangle{\Downarrow}^u$ in l steps. We must show that $C\langle N\rangle{\Downarrow}^{\le u}$ using the induction hypothesis:

$$\boxed{\begin{array}{rl} & \forall C'.\forall s < u.\quad C'\langle M\rangle{\Downarrow}^s \Rightarrow C'\langle N\rangle{\Downarrow}^{\le s} \\ \text{and} & \forall C'.\quad (C'\langle M\rangle{\Downarrow}^u \text{ in } l' < l \text{ steps}) \Rightarrow C'\langle N\rangle{\Downarrow}^{\le u} \end{array}}$$

We proceed by case analysis according to the length l of the computation:

Case 1 ($l = 0$) In this case $C\langle M\rangle \in \mathit{Val}$, and hence $u = \mathbf{0}$. Since $r > \mathbf{0}$ it follows from Proposition A.7(ii) that $C \not\equiv \langle\ \rangle$, and hence C has an outermost constructor. This implies $C\langle N\rangle$ is also a value, and hence $C\langle N\rangle{\Downarrow}^{\mathbf{0}}$.

Case 2 ($l > 0$) In this case $C\langle M\rangle \overset{s}{\mapsto} M'$ for some s, M'. Since M is not a value (as argued above), we can consider cases according to Proposition A.6:

Case 2.1 (The reduction step is uniform in M) In this case C can be written as $\hat{\mathcal{R}}\langle\ \rangle[C'\langle\ \rangle]$ and

$$\begin{array}{rcl} \hat{\mathcal{R}}\langle M\rangle[C'\langle M\rangle] & \overset{s}{\mapsto} & \hat{\mathcal{R}}\langle M\rangle[C''\langle M\rangle]{\Downarrow}^t \\ \hat{\mathcal{R}}\langle N\rangle[C'\langle N\rangle] & \overset{s}{\mapsto} & \hat{\mathcal{R}}\langle N\rangle[C''\langle N\rangle] \end{array}$$

for some C'', and some t such that $s \cdot t = u$. Now we consider cases according to the value of s:

Case 2.1.1 ($s = \mathbf{0}$) In this case $\hat{\mathcal{R}}\langle M\rangle[C''\langle M\rangle]{\Downarrow}^u$ in $l-1$ steps, so by the induction hypothesis $\hat{\mathcal{R}}\langle N\rangle[C''\langle N\rangle]{\Downarrow}^{\le u}$. But since $t = \mathbf{0}$ then $C\langle N\rangle{\Downarrow}^{\le u}$.

Case 2.1.2 ($s > \mathbf{0}$) Since composition is $<$-monotonic for a well-founded resource, it follows that $t < u$, so by the induction hypothesis we have $\hat{\mathcal{R}}\langle N\rangle[C''\langle N\rangle]{\Downarrow}^{\le t}$. Hence $C\langle N\rangle{\Downarrow}^{\le s \cdot t = u}$.

Case 2.2 (The reduction step is inside M) In this case C can be written as $\hat{\mathcal{R}}\langle\ \rangle[\langle\ \rangle]$. Since $M \underset{\sim}{\triangleright}^r C_0\langle M\rangle$, by Lemma A.7 (i) we have $\hat{\mathcal{R}}\langle M\rangle[M] \underset{\sim}{\triangleright}^r \hat{\mathcal{R}}\langle M\rangle[C_0\langle M\rangle]$ and hence by Lemma A.7 (ii) $\hat{\mathcal{R}}\langle M\rangle[C_0\langle M\rangle]{\Downarrow}^t$ for some t such that $r \cdot t \le u$. Since $r > \mathbf{0}$ it follows that $t < u$, and so by the induction hypothesis $\hat{\mathcal{R}}\langle N\rangle[C_0\langle N\rangle]{\Downarrow}^{\le t}$. Now from Lemma A.7 (iii) it follows that $\hat{\mathcal{R}}\langle N\rangle[N]{\Downarrow}^{\le r \cdot t \le u}$. □

The proof of the extension of the theorem to open expressions, in the special cases given in Theorem 7.4, follows exactly the same structure as the above proof. The difference is that we need a more elaborate version of Lemma A.6 in which uniform computation is described by extending the evaluation relation to contexts. This is technically a little problematic (see for example [Tal97] for one approach based on substitution-decorated holes); our approach — which will be described elsewhere — is to use second-order syntax for contexts. This combines smoothly with the use of a second-order syntax for SOS-rules so as to obtain, "for free", a definition of context-evaluation which commutes with hole-filling.