Operational Semantics for Program Equivalence

Andrew M. Pitts

Cambridge University Computer Laboratory

http://www.cl.cam.ac.uk/users/ap/talks

Invited talk at

MFPS XIII, CMU, Pittsburgh, 23–26 March 1997

by

Andrew M. Pitts
Cambridge University Computer Laboratory
Pembroke Street, Cambridge CB2 3QG, UK

`ap@cl.cam.ac.uk`

Abstract

Languages such as Scheme and ML are *functional* programming languages because they treat functions as values on a par with more concrete forms of data: functions can be passed as arguments, can be returned as the result of computation, can be recursively defined, and so on. They are also *procedural* languages because they permit the use of references (or 'cells', or 'locations') for storing values: references can be declared locally in functions and then created dynamically and their contents read and updated as function applications are evaluated. This mixing of call-by-value, higher order functions with local, dynamically allocated state results in many subtle properties of these functions up to operational equivalence. The traditional methods of denotational semantics do not capture these subtleties very well—domain-based models tend to be either far from 'fully abstract', or very complicated, or both. Consequently a sort of 'back to basics' movement has arisen that attempts to develop theories of program equivalence for high-level languages based directly on operational semantics (see Pitts 1997a for some of the literature). The starting point for such work is of course the *structural* approach to operational semantics initiated by Plotkin (1981), Milner, Kahn, and others. This comes in several different

flavours. In this talk I will attempt a whirlwind tour of the different approaches to operational semantics, pointing out the advantages and disadvantages of each for reasoning about operational equivalence of programs. I will argue that one particular and possibly unfamiliar approach—a 'continuation-style' operational semantics derived from the approach of Felleisen and Hieb (1992) (and used in the recent redefinition of ML by Harper and Stone 1996)—provides a more convenient basis for developing properties of operational equivalence than does the evaluation (or 'natural', or 'big-step') semantics used in the official definition of Standard ML (Milner, Tofte, and Harper 1990).

References

Felleisen, M. and R. Hieb (1992). The revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science 103*, 235–271.

Harper, R. and C. Stone (1996, September). A type-theoretic account of Standard ML 1996 (version 2). Technical Report CMU–CS–96–136R, Carnegie Mellon University, Pittsburgh, PA.

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

Milner, R., M. Tofte, and R. Harper (1990). *The Definition of Standard ML*. MIT Press.

Pitts, A. M. (1997a). Operationally-based theories of program equivalence. In P. Dybjer and A. M. Pitts (Eds.), *Semantics and Logics of Computation*, Publications of the Newton Institute, pp. 241–298. Cambridge University Press.

Pitts, A. M. (1997b). Reasoning about local variables with operationally-based logical relations. In P. W. O'Hearn and R. D. Tennent (Eds.), *Algol-Like Languages*, Volume 2, Chapter 17, pp. 173–193. Birkhauser. First appeared in *Proceedings Eleventh Annual IEEE Symposium on Logic in Computer Science*, Brunswick, NJ, July 1996, pp 152–163.

Plotkin, G. D. (1981). A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University.

“Every lecture should make only one main point”

Gian-Carlo Rota

Ten Lessons I wish I Had Been Taught

Notices AMS 44(1997)22–25

Mine is:

A particular style of operational semantics enables a ‘structural’ definition of termination that’s very useful for reasoning about operational equivalence of programs.

But of course it will take me some time to explain what I mean and to try to convince you that it is a good point!

Here is the plan ...

Outline

- *Contextual equivalence* of Standard ML program phrases.
- Styles of *structural operational semantics*—a brief tour.
- A structural definition of termination, via ‘continuation-style’ operational semantics.
- Applications to reasoning about contextual equivalence.

Contextual equivalence (in general)

Two phrases of a programming language are contextually equivalent ($\cong_{\text{ctx}}$) if any occurrences of the first phrase in a complete program can be replaced by the second phrase without affecting the observable results of executing the program.

Also known as *operational*, or *observational* equivalence.

Standard ML

Structural operational semantics is part of the Definition of SML (Milner, Tofte, Harper, 1990).

So $\cong_{\mathrm{ctx}}$ is the *definitive* notion of semantic equivalence for SML.

Adequate ($[\![P]\!] = [\![P']\!] \Rightarrow P \cong_{\mathrm{ctx}} P'$) denotational models exist, but they are either far from ‘fully abstract’, or very complicated, or both (’cos of SML’s combination of features).

Instead: attempt to develop theory of $\cong_{\mathrm{ctx}}$ using syntactical techniques (e.g. structural and rule-based induction). But it’s hard!

SML's combination of

recursively defined, higher order, call-by-value functions
+
statically scoped, dynamically created, mutable state

makes properties of $\cong_{\mathrm{ctx}}$ very complicated.

To explain why direct, syntactical methods are hard use, we need to look at the precise definition of $\cong_{\mathrm{ctx}}$. First, I have to recall the form of structural operational semantics used in the Definition of SML. I'll use a slightly simplified form ...

SML Evaluation Semantics (simplified)

Evaluation relation

$$\boxed{s, M \Rightarrow V, s'} \quad \begin{cases} s & = \text{initial state} \\ M & = \text{term to be evaluated} \\ V & = \text{resulting value} \\ s' & = \text{final state} \end{cases}$$

inductively generated by axioms and rules *following the structure of* M—e.g.

$$\frac{s, M_1 \Rightarrow V_1, s' \quad s', M_2[V_1/x] \Rightarrow V_2, s''}{s, \texttt{let}\,\texttt{val}\; x = M_1 \;\texttt{in}\; M_2 \;\texttt{end} \Rightarrow V_2, s''}$$

Evaluation semantics is also known as *big-step* (anon), *natural* (Kahn), or *relational* (Milner) semantics.

A *state* is (at least) an assignment of values to storage location names.

This is a simplified form of the evaluation relation actually used in the Definition of SML. The latter has an *environment* component binding free identifiers to values, whereas we will get by with this simpler form, in which the environment has been 'substituted in'. (SML also needs various auxiliary relations to deal with exception-handling, etc.)

This is an example of the *Structural approach to Operational Semantics* (SOS) which was popularised by Plotkin (1981). SOS more closely reflects our intuitive understanding of various language constructs compared to previous approaches using abstract machines, which tended to pull the syntax apart and build non-intuitive auxiliary data structures. Furthermore the structural nature of the axioms and rules is of great help when it comes to using 'rule induction' to prove properties of the inductively defined relation.

Contextual preorder / equivalence

$$M_1 \cong_{\mathrm{ctx}} M_2 \stackrel{\mathrm{def}}{\Leftrightarrow} M_1 \leq_{\mathrm{ctx}} M_2 \;\&\; M_2 \leq_{\mathrm{ctx}} M_1$$

$$M_1 \leq_{\mathrm{ctx}} M_2 \stackrel{\mathrm{def}}{\Leftrightarrow} \forall s, \mathcal{M}[-]\,(s, \mathcal{M}[M_1]{\Downarrow} \Rightarrow s, \mathcal{M}[M_2]{\Downarrow})$$

where s ranges over states and $\mathcal{M}[-]$ over contexts,

and where $s, M{\Downarrow}$ indicates termination:

$$s, M{\Downarrow} \stackrel{\mathrm{def}}{\Leftrightarrow} \exists s', V\,(s, M \Rightarrow V, s')$$

Other natural choices of what to observe apart from termination do not change $\cong_{\mathrm{ctx}}$.

Typical questions

Beta conversion

$$¿ \quad \mathtt{let\,val}\ x = M_1\ \mathtt{in}\ M_2\ \mathtt{end} \cong_{\mathrm{ctx}} M_2[M_1/x] \quad ?$$

Extensionality

$$¿ \quad (\forall M\,.\,F_1 M \cong_{\mathrm{ctx}} F_2 M) \Rightarrow F_1 \cong_{\mathrm{ctx}} F_2 \quad ?$$

These fail in general (because of call-by-value and effects). What properties of M_1, M_2, F_1, F_2 can make them valid? (They are valid when M_1, F_1, F_2 are values; more generally ...?)

Beta-conversion (and similar equational properties) for various classes of M_1 (not necessarily just values) lies at the heart of several transformations used in compiler optimisations.

Extensionality is of course of great use in reasoning about properties of functions. It's validity when M_1 and M_2 are function values is not so easy to prove—e.g. on the face of it the property cannot be deduced from the existence of a merely adequate, extensional model, because of the conditional nature of the statement.

Termination $s, M{\Downarrow}$ is not defined by induction on the structure of M

E.g. $$\frac{s', M_2[V_1/x]{\Downarrow}}{s, \texttt{let val}\ x = M_1\ \texttt{in}\ M_2\ \texttt{end}{\Downarrow}}$$ if $s, M_1 \Rightarrow V_1, s'$

but $M_2[V_1/x]$ is not built from subphrases of $\texttt{let val}\ x = M_1\ \texttt{in}\ M_2\ \texttt{end}$.

Simple e.g. of difficulty this causes: to prove

if Ω is the standard divergent term, then $\Omega \leq_{\mathrm{ctx}} N$ for any N

obvious strategy is to show

$$s, M{\Downarrow}\ \Rightarrow\ \forall \mathcal{M}[-]\ (M = \mathcal{M}[\Omega] \Rightarrow s, \mathcal{M}[N]{\Downarrow})$$

by induction on the derivation of $s, M{\Downarrow}$.

This proof method fails because of the above problem.

Aim: reformulate the operational semantics to get a structurally inductive definition of termination and hence aid inductive reasoning about $\cong_{\mathrm{ctx}}$.

To achieve that, we need to review Plotkin's original approach to SOS and subsequent refinements of it ...

Transition relation $\boxed{(s, M) \to (s', M')}$

(s, s' states, M, M' terms)

inductively generated by axioms and rules following the structure of M—e.g.

- $$\frac{(s, M_1) \to (s', M_1')}{(s, \mathtt{let\,val}\ x = M_1\ \mathtt{in}\ M_2\ \mathtt{end}) \to (s', \mathtt{let\,val}\ x = M_1'\ \mathtt{in}\ M_2\ \mathtt{end})}$$

- $(s, \mathtt{let\,val}\ x = V_1\ \mathtt{in}\ M_2\ \mathtt{end}) \to (s, M_2[V_1/x])$

(V_1 a value)

Lemma. $s, M \Rightarrow V, s'$ *iff* $(s, M) \to^* (s', V)$.

Cf. (Plotkin 1981).

Transition semantics is also known as ‘reduction’, or ‘small-step’ semantics.

The axioms of a transition semantics give the basic steps of *reduction*, whereas the rules say how reductions may be performed within a context.

Felleisen-style presentation of $\rightarrow$

Lemma.

$$(s, M) \rightarrow (s, M') \Leftrightarrow \exists \textit{ evaluation context } \mathcal{E}[-], \quad \textit{reduction } (s, R) \rightarrow (s', R') \;. \quad M = \mathcal{E}[R] \;\&\; \mathcal{E}[R'] = M'$$

Evaluation contexts

$\mathcal{E} ::= - \mid \texttt{let}\,\texttt{val}\; x = \mathcal{E}\; \texttt{in}\; M\; \texttt{end} \mid \ldots$

contexts that want to evaluate their hole.

Reductions are the axioms in the inductive definition of $\rightarrow$ à la Plotkin.

The validity of this characterisation of $\longrightarrow$ relies upon ...

Lemma. *Every non-value is uniquely of the form $\mathcal{E}[R]$ for some evaluation context $\mathcal{E}[-]$ and redex R.*

This suggests separating $\mathcal{E}$ from R in the process of evaluating $(s, \mathcal{E}[R])$, i.e. using an operational semantics based on configurations of the form

$$(s, K, R)$$

where K is ...

Continuation stacks, $K ::= \mathtt{Id} \mid K \circ (x)M \mid \ldots$

Typical evaluation context (in a fragment of core SML without exception-handling)

$$
\begin{array}{l}
\mathcal{E}[-] = \mathtt{let\,val}\ x_1 = \\
\qquad \mathtt{let\,val}\ x_2 = \\
\qquad\quad \ldots \\
\qquad\qquad \mathtt{let\,val}\ x_n = [-]\ \mathtt{in}\ M_n\ \mathtt{end} \\
\qquad\quad \ldots \\
\qquad \mathtt{in}\ M_2\ \mathtt{end} \\
\quad \mathtt{in}\ M_1\ \mathtt{end}
\end{array}
$$

corresponds to the continuation stack

$$K = \mathtt{Id} \circ (x_1)M_1 \circ (x_2)M_2 \circ \ldots \circ (x_n)M_n$$

Denotationally, such a stack corresponds to a *composition of strict continuous functions*.

An abstract machine for evaluating configurations (s, K, M)

'Abstract machine' because its transitions

$(s, K, M) \to (s', K', M')$ are *defined by cases*

(i.e. no induction), according to the structure of M and (then) K:

- $(s, K \circ (x)M_2, V_1) \to (s, K, M_2[V_1/x])$ if V_1 a value
- $(s, K, \texttt{let}\,\texttt{val}\ x = M_1\ \texttt{in}\ M_2\ \texttt{end}) \to (s, K \circ (x)M_2, M_1)$
- $(s, K, \texttt{ref}(V)) \to (s[x \mapsto V], K, x)$ if $x \notin dom(s)$

etc ...

See Harper & Stone (1996) *Type-Theoretic Account of SML 1996* for a large-scale e.g. of this style of SOS.

$$\boxed{\textbf{Theorem. } (s, K, M) \to^* (s', \mathtt{Id}, V) \textit{ iff } s, K@M \Rightarrow V, s'.}$$

where

$$\begin{cases} \mathtt{Id}@M & \stackrel{\text{def}}{=} M \\ (K \circ (x)M')@M & \stackrel{\text{def}}{=} K@(\mathtt{let\,val}\ x = M\ \mathtt{in}\ M'\ \mathtt{end}). \end{cases}$$

Hence: $s, M\Downarrow$ iff $\exists s', V\,.\,(s, \mathtt{Id}, M) \to^* (s', \mathtt{Id}, V)$.

Simple, but key, observation:

$$\searrow \stackrel{\text{def}}{=} \{(s, K, M) \mid \exists s', V\,.\,(s, K, M) \to^* (s', \mathtt{Id}, V)\}$$

has a direct, inductive definition that follows the structure of M *and* K*, viz. ...*

Structurally inductive definition of termination

- $(s, \mathtt{Id}, V)\searrow$

- $$\frac{(s, K, M[V/x])\searrow}{(s, K \circ (x)M, V)\searrow}$$

- $$\frac{(s, K \circ (x)M_2, M_1)\searrow}{(s, K, \mathtt{let\,val}\ x = M_1\ \mathtt{in}\ M_2\ \mathtt{end})\searrow}$$

- $$\frac{(s[x \mapsto V], K, x)\searrow}{(s, K, \mathtt{ref}(V))\searrow} \text{ if } x \notin dom(s)$$

  etc ...

(V ranges over values)

We have thus achieved the aim of reformulating the structural operational semantics of SML to get a structurally inductive characterisation of termination. Before outlining what can be done with this, it's perhaps helpful to contemplate the following picture ...

The relation we are interested in

is a retract of

a larger one with better structural properties.

$$\begin{tikzcd}
\Downarrow \arrow[r, hook] \arrow[d, hook] & \searrow \arrow[l, bend left] \arrow[d, hook] \\
\mathbb{S} \times \mathbb{M} \arrow[r, hook] & \mathbb{S} \times \mathbb{K} \times \mathbb{M} \arrow[l, bend left]
\end{tikzcd}$$

$$(s, M) \longmapsto (s, \mathtt{Id}, M)$$

$$(s, K@M) \longmapsfrom (s, K, M)$$

Applications of termination-induction, I

One can use rule induction for $\searrow$ to prove instances of the contextual preorder / equivalence:

Proposition. *To prove* $M_1 \leq_{\mathrm{ctx}} M_2$ *it suffices to show that*

$$\begin{aligned}\{(s, K, M) \mid \forall \mathcal{S}[-], \mathcal{K}[-], \mathcal{M}[-]\,.\\ s = \mathcal{S}[M_1] \;\&\; K = \mathcal{K}[M_1] \;\&\; M = \mathcal{M}[M_1]\\ \Rightarrow\; (\mathcal{S}[M_2], \mathcal{K}[M_2], \mathcal{M}[M_2])\searrow\}\end{aligned}$$

is closed under the axioms and rules inductively defining $\searrow$.

E.g. this way we get an easy proof of $\boxed{\Omega \leq_{\mathrm{ctx}} M, \text{ any } M}$.

'Unwinding' Theorem. *Given* $F(f,x)$, *defining*

$$\begin{array}{rcl} F_0 & \stackrel{\mathrm{def}}{=} & \mathtt{fn}\, x \mathop{=\!>} \Omega \\ F_{n+1} & \stackrel{\mathrm{def}}{=} & \mathtt{fn}\, x \mathop{=\!>} F(F_n, x) \\ F_\omega & \stackrel{\mathrm{def}}{=} & \mathtt{fn}\, x \mathop{=\!>} \mathtt{let}\ \mathtt{fun}\, f x = F(f,x)\ \mathtt{in}\ F(f,x)\ \mathtt{end} \end{array}$$

then $\boxed{\forall s, \mathcal{M}[-]\,(s, \mathcal{M}[F_\omega]{\Downarrow} \;\Rightarrow\; \exists n < \omega \,.\, s, \mathcal{M}[F_n]{\Downarrow})}$.

Proof.

$$\begin{aligned} \{(s, K, M) \mid \forall \mathcal{S}[-], \mathcal{K}[-], \mathcal{M}[-] \,.\, & \\ s = \mathcal{S}[F_\omega] \;\&\; K = \mathcal{K}[F_\omega] \;\&\; M = \mathcal{M}[F_\omega] & \\ \Rightarrow\; \exists n < \omega\; (\mathcal{S}[F_n], \mathcal{K}[F_n], \mathcal{M}[F_n]){\searrow} \,\} & \end{aligned}$$

is closed under the axioms and rules inductively defining $\searrow$.

F_ω is a typical recursively defined function value in SML. The F_n are its finite ‘unwindings’.

The Unwinding Theorem has been proved by several different people (e.g. by Scott Smith) in various contexts. It allows one to use syntactic analogues of Dana Scott’s induction principle for least fixed points. It is also needed to prove the Fundamental Property for various kinds of syntactically defined logical relations when recursion is present (cf. Slide 21).

Applications of termination-induction, III

Mason-Talcott 'ciu' Theorem.

(ci) *Open terms* are $\cong_{\mathrm{ctx}}$ iff their closed substitution instances are.

(u) *Closed terms*: $M_1 \cong_{\mathrm{ctx}} M_2$ iff
$\forall s, K \,.\, (s, K, M_1)\searrow \;\Leftrightarrow\; (s, K, M_2)\searrow$.

$\exists$ nice proof adapting Howe's method for congruence of $\Rightarrow$-based applicative bisimilarity to $\searrow$-based ciu-equivalence.

Corollary. Validity of β_{v}-conversion (etc.) and extensionality for function values.

For Howe's method for congruence of $\Rightarrow$-based applicative bisimilarity see (Howe 1996).

My own motivation for developing the structurally inductive termination relation for SML mainly has to do with extending my work on operationally-based logical relations (Pitts 1997b) from Idealised Algol to SML ...

Operationally-based logical relations for SML

[AMP+Ian Stark,*Operational Reasoning for Functions with Local State*, in preparation.]

Provides a method for proving $\cong_{\mathrm{ctx}}$ that formalises intuitive uses of invariant properties of local state.

Novel formulation: relations (parameterised by state-relations) for terms *and continuations* whose definitions involve $\searrow$. Avoids technical difficulties in previous, $\Rightarrow$-based formulation.

So far, only storage of 'flat' datatypes. Arbitrary `ref`-types requires (at least) syntactic version of *minimal invariant* property of recursively defined domains. (Also needed for *operational theory of objects*—joint work with A. Gordon.)

The ‘technical difficulties’ mentioned on Slide 21 have to do with the fact that in SML the size (= number of addresses) of the final state in an evaluation may be larger than the size of the initial state—due to dynamic allocation of local state. The ‘natural’ definition of the logical relation based on $\Rightarrow$ involves an existential quantification over state-relations for the newly created state: this existential quantification messes up the proof of the Fundamental Property of the logical relation when it comes to recursively defined functions—because one needs a syntactic ‘admissibility’ property of the logical relation which is (probably) false. By contrast, in the formulation based on $\searrow$, one never mentions the final state and the problem is side-stepped.

Syntactic versions of the minimal invariant property of recursively defined domains have been established in various contexts by Scott Smith, by Lars Birkedal, and by Søren Lassen. However, this is not the only problem that has to be overcome to extend the AMP+Stark work on logical relations for SML from ‘flat’ to arbitrary state.

Conclusion

At the cost of introducing continuation stacks (formalised evaluation contexts) into the operational semantics, it is possible to have a *structural inductive characterisation of termination* which is very useful for reasoning about contextual equivalence in the presence of dynamically allocated local state.

For simpler languages (e.g. for pure functional ones) this approach is not optimal—coinductive methods based on $\Rightarrow$, or $\rightarrow$, give sharper results.

On the other hand, the approach outlined in this talk seems quite *robust*, e.g. adding exception-handling or first-class continuations may complicate the rules for $(s, K, M)\searrow$, but should not change the proof techniques (?).