Full Abstraction for the Second Order Subset of an ALGOL-like Language (Preliminary Report)

Kurt Sieber

Technischer Bericht A 01/94

FB 14 Informatik
Universität des Saarlandes
66041 Saarbrücken
Germany
`sieber@cs.uni-sb.de`

February 17, 1994

Abstract

We present a denotational semantics for an ALGOL-like language ALG which is fully abstract for the second order subset of ALG. This constitutes the first significant full abstraction result for a block structured language with local variables.

In this preliminary report we concentrate on the construction of the denotational model and on the main ideas of the full abstraction proof. For more background information about (problems involved with) the semantics of local variables, especially for further interesting examples of observational congruences we refer the reader to [MS88, OT93b].

1 Introduction

This paper solves a long-standing open problem concerning the semantics of local variables. We present a denotational model for an ALGOL-like language ALG, which is fully abstract for the second order subset of ALG. This means in particular that all the problematic observational congruences for ALGOL-like languages, which have been presented in the literature [MS88, Len93, OT93b], can be validated in our model. (The latter also holds for the parametric functor model in [OT93a, OT93b], but no full abstraction result has been proved for it.)

The general technique which we use for our model construction has already been developed in [MS88], namely 'relationally structured locally complete partial orders' with 'relation preserving locally continuous functions'. Our particular model differs from the one in [MS88] by having the 'finest possible relation structure', an idea which we have used in [Sie92] to construct a fully abstract model for the second order subset of sequential PCF [Plo77].

The overall structure of our full abstraction proof[1] is also taken from [Sie92]. In the first step[2] we show that for every function f and every finite set B of argument tuples for f there is a *definable* function which coincides with f on B (Theorem 3). Hence we can find a sequence of definable functions which 'approximate' f in the sense that they coincide with f on more and more argument tuples. But for proving full abstraction (Theroem 5) we must find approximations in the Scott topology, i.e. we must show that f is the *least upper bound* of a sequence (or directed set) of definable functions (Theorem 4). Bridging the gap between these two notions of 'approximation' turned out to be the most difficult part of our full abstraction proof, for which we had to develop completely new techniques (Definition 5 and Theorem 6).

Our ALGOL-like language ALG contains two (at least for non-insiders) unusual features, namely (a) a *parallel conditional* operator on the integers and (b) the so-called *snap back effect*, which goes back to a suggestion of J.C. Reynolds: Inside the bodies of function procedures, assignments to global variables are allowed, but after each function procedure call the store 'snaps back' to the contents which it had before the call, i.e. only a *temporary* side effect is caused by such assignments.

The parallel conditional often plays a prominent role in full abstraction proofs, but here it does not. If we remove it from ALG, then we can use the very same techniques as before to obtain a fully abstract model for the restricted language (cf. Conclusion). This 'smaller' model allows us to reason not only about local variables but also about sequentiality. In the light of [Sie92] this is not a big surprise, but nevertheless it is worth to be mentioned, because it distinguishes our approach from the one in [OT93a, OT93b] which is tailored to an ALGOL-like

[1] In the remainder of the Introduction we tacitly assume that we are not speaking about the full language but only about the second order subset.

[2] This first step has already been presented in [Sie93].

language with snap back effect *and* parallel conditional [OT].

The snap back effect plays a more important role than the parallel conditional. If function procedures have either *permanent* side effects [WF93] or *no* side effects at all [Len93], then it seems more difficult to determine the above mentioned 'finest possible relation structure' for the construction of a fully abstract model. This is the reason why our techniques do *not straightforwardly* carry over to these alternative languages. Nevertheless we believe that they can still be applied; this is the contents of current research.

Finally one might wonder whether similar techniques are applicable to call-by-value (i.e. ML-like as opposed to ALGOL-like) languages [PS93]. This is a question which we have not yet investigated. Observations in [PS93] indicate that additional problems might come up in the call-by-value setting, but we hope that our main ideas will still be helpful.

2 Syntax of the language ALG

We define our ALGOL-like language ALG as a subset of a simply typed λ-calculus. Its *types* τ are

$$\tau ::= loc \,\big|\, \sigma \qquad \text{with} \qquad \sigma ::= \theta \,\big|\, \tau \to \sigma, \quad \theta ::= iexp \,\big|\, cmd$$

The types $\sigma\,(\neq loc)$ are called *procedure types*. The *order* $ord(\tau)$ of a type τ is defined by $ord(loc) = 0$, $ord(\theta) = 1$ and $ord(\tau \to \sigma) = max\,(ord(\tau) + 1, ord(\sigma))$.

Elements of type *iexp* (= 'integer expresssion') and *cmd* (= 'command') will be functions which have the current store as an implicit parameter; in particular parameters of type *iexp* will be *thunks* in terms of the ALGOL jargon. Thus we follow the view that *call by name* should be the main parameter passing mechanism for ALGOL-like languages [Rey81]. Besides that, we have parameters of type *loc* (= 'location') which may be considered as *reference parameters*. They have been added as a mere convenience, because we anyways need identifiers of type *loc* as local variables.

The set of ALG-*constants* c and the *type* of each constant are

n :	$iexp$ for every $n \in \mathbb{Z}$	(numerals)
$succ, pred$:	$iexp \to iexp$	(successor and predecessor)
$cont$:	$loc \to iexp$	(dereferencing)
$asgn$:	$loc \to iexp \to cmd$	(assignment)
$skip$:	cmd	(empty command)
$cond_\theta$:	$iexp \to \theta \to \theta \to \theta$	(conditional with zero test)
seq_θ :	$cmd \to \theta \to \theta$	(sequencing)
new_θ :	$(loc \to \theta) \to \theta$	(*new*-operator)
Y_σ :	$(\sigma \to \sigma) \to \sigma$	(fixed point operator)
$pcond$:	$iexp \to iexp \to iexp \to iexp$	(parallel conditional with zero test)

As usual, we assume that there is an infinite set Id^τ of identifiers $x^\tau, y^\tau, z^\tau, \ldots$ for each type τ (the type superscripts will often be omitted). Identifiers of type *loc* are called *variables*. This means that we use the word 'variable' in the sense of imperative languages and not in the sense of the λ-calculus.

Expressions $M, N, P, \ldots$ of ALG are just the well-typed λ-expressions over the ALG-constants with the only restriction that the body of a λ-abstraction must not be of type *loc*. A block with a local variable x has the form **new** x **in** M and is considered as syntactic sugar for $new_\theta(\lambda\, x^{loc}.\, M)$ where θ is the type of M; this makes the binding of the local variable x visible. As further syntactic sugar we use $!\_$, $\_ := \_$, **if** _ **then** _ **else** _ and _ ; _ instead of $cont, asgn, cond_\theta$ and seq_θ. Finally we define a *program* P to be a closed expression of type *iexp*.

For purely technical reasons we also introduce so-called generalized expressions. Let *Loc* be an infinite set whose elements l are called *locations*. A *generalized expression* may contain (besides the other ALG-constants) locations l as constants of type *loc*. For generalized expressions we use the same metavariables $M, N, P, \ldots$ as for ordinary expressions. We let $locns(M)$ denote the set of locations which occur in M, and for every finite set $L \subseteq Loc$ we let Exp^τ_L denote the set of *closed* generalized expressions with $locns(M) \subseteq L$.

3 A Cartesian Closed Category

Notation: By a *dcpo* (*directed complete partial order*) we mean a partial order $(D, \sqsubseteq)$ in which every directed set Δ has a lub (least upper bound) $\bigsqcup \Delta$ (or $\bigsqcup_D \Delta$ if we want to be more precise). If D, E are dcpos, then $(D \overset{c}{\to} E)$ denotes the set of continuous functions from D to E. The category of dcpos and continuous functions is denoted **DCPO**.

We will now define the general framework which underlies our denotational semantics. The intuition is, that every element in the denotational model should only have access to finitely many locations. Hence we would like to identify, for every type τ and every finite set $L \subseteq Loc$, a dcpo $[\![\tau]\!]_L$ of 'elements of type τ which only have access to L' and then define $[\![\tau]\!]$ as the union of these dcpos $[\![\tau]\!]_L$. This motivates the following definition.

Definition 1 *Let $(W, \leq)$ be a directed set (of* worlds *w).*

(a) *A W-*locally complete partial order *(W-lcpo) is a partial order $(D, \sqsubseteq)$ together with a family of subsets $(D_w)_{w \in W}$ such that $D = \bigcup_{w\in W} D_w$ and for all $v, w \in W$*

- $v \leq w \Rightarrow D_v \subseteq D_w$
- *if $\Delta \subseteq D_w$ is directed, then $\bigsqcup_D \Delta$ exists and is contained in D_w (hence it is also the lub in D_w, i.e. $(D_w, \sqsubseteq)$ is a dcpo)*

(b) *A function $f : D \to E$ between W-lcpos D and E is called* locally continuous *if $(f \mid D_w) \in (D_w \xrightarrow{c} E_w)$ for every $w \in W$.*

W-lcpos and locally continuous functions form a Cartesian closed category (which may be considered as a full subcategory of the functor category $(W \Rightarrow \mathbf{DCPO})$). Terminal object and products are defined worldwise and the exponent $(D \to E)$ of two objects D and E is given by

$$\begin{array}{lcll} (D \to E)_w & = & \{f : D \to E \,|\, \forall v \geq w.\, (f \mid D_v) \in (D_v \xrightarrow{c} E_v)\} & \\ (D \to E) & = & \bigcup_{w \in W} (D \to E)_w & \text{with the pointwise order on functions} \end{array}$$

This is not yet the category which we need for our model construction; we must still add 'relation structure' to the W-lcpos.

Definition 2 *A W-*sorted (relation) signature *is a family $\Sigma = (\Sigma_n^w)_{w \in W, n \in \mathbb{N}}$ of sets Σ_n^w such that for all $m, n \in \mathbb{N}$ and $v, w \in W$*

$$m \neq n \Rightarrow \Sigma_m^v \cap \Sigma_n^w = \emptyset \qquad \textit{and} \qquad v \leq w \Rightarrow \Sigma_n^v \supseteq \Sigma_n^w$$

We use the notation

$$\Sigma_n = \bigcup_{w \in W} \Sigma_n^w, \quad \Sigma^w = \bigcup_{n \in \mathbb{N}} \Sigma_n^w \quad \textit{and (ambiguously)} \quad \Sigma = \bigcup_{n \in \mathbb{N}} \Sigma_n$$

*An element $r \in \Sigma_n$ is called an n-*ary relation symbol.

As we will extensively work with tuples and relations, we introduce some shorthand notation for them:

A vector $\vec{d}$ stands for a tuple $(d_1, \ldots, d_n) \in D^n$, where D and n are either known from the context or irrelevant. A term $T(\vec{d}, \vec{e}, \ldots)$ containing vectors $\vec{d}, \vec{e}, \ldots$ of the same length n stands for the tuple $(T(d_1, e_1, \ldots), \ldots, T(d_n, e_n, \ldots))$ and a formula $F(\vec{d}, \vec{e}, \ldots)$ for the conjunction $F(d_1, e_1, \ldots) \wedge \ldots \wedge F(d_n, e_n, \ldots)$. The term notation is generalized as usual to *sets* of tuples, i.e. to relations: If R, S are relations of the same arity n, then $T(R, S, \ldots)$ stands for the set $\{T(\vec{d}, \vec{e}, \ldots) \,|\, \vec{d} \in R, \vec{e} \in S, \ldots\}$. Finally, $\delta^n D$ (or just δD) denotes the diagonal $\{(d, \ldots, d) \,|\, d \in D\} \subseteq D^n$. (A helpful intuition is to consider vectors as *column* vectors and to read terms and formulas *linewise*.)

Definition 3 *Let Σ be a W-sorted signature.*

(a) *A* W-Σ-lcpo *is a pair $(D, \mathcal{I})$, where D is a W-lcpo and $\mathcal{I}$ is a function which maps every $r \in \Sigma_n$ to a relation $\mathcal{I}(r) \subseteq D^n$ such that for all $w \in W$*

- *$r \in \Sigma^w \Rightarrow \delta^n D_w \subseteq \mathcal{I}(r)$*
- *$\mathcal{I}(r) \cap D_w^n$ is closed under lubs of directed sets*

(b) *A function $f : D \to E$ between W-Σ-lcpos $(D, \mathcal{I}^D)$ and $(E, \mathcal{I}^E)$ is called a* Σ-homomorphism *if $f(\mathcal{I}^D(r)) \subseteq \mathcal{I}^E(r)$ for all $r \in \Sigma$.*

Theorem 1 *The category W-Σ-**LCPO** of W-Σ-lcpos and locally continuous Σ-homomorphisms is Cartesian closed. Terminal object and product are defined worldwise and the exponent $(D \to E)$ of two W-Σ-lcpos D and E is given by*

$$\begin{aligned}
(D \to E)_w &= \{ f : D \to E \mid \forall v \geq w. (f \,|\, D_v) \in (D_v \overset{c}{\to} E_v) \\
&\qquad\qquad\qquad \wedge \forall r \in \Sigma^w. f(\mathcal{I}^D(r)) \subseteq \mathcal{I}^E(r) \} \\
(D \to E) &= \textstyle\bigcup_{w \in W} (D \to E)_w \quad \textit{with the pointwise order on functions} \\
\mathcal{I}^{(D \to E)}(r) &= \{ \vec{f} \mid \vec{f}(\mathcal{I}^D(r)) \subseteq \mathcal{I}^E(r) \}
\end{aligned}$$

This is the category in which we will define our denotational model. It has a certain similarity with the category of 'parametric functors and (parametric) natural transformations' as defined in [OT93a, OT93b]. The precise relationship between the two approaches is not yet fully understood, but at least one difference seems to be important: Whereas the definition in [OT93a, OT93b] works with binary relations only (and can be generalized to relations of some fixed arity n [OT]), our approach allows us to have relations of *arbitrary* arity in *one* denotational model. This fact is exploited in our full abstraction proof (hence the proof does not automatically carry over to the parametric functor model) and—moreover—it allows us to obtain a fully abstract model for Alg *without* parallel conditional by the very same techniques as for Alg itself.

4 Denotational Semantics

We will now use the results of Section 3 to define a denotational semantics for Alg. We let

$$(W, \leq) = (\mathcal{P}_f(Loc), \subseteq)$$

where $\mathcal{P}_f(Loc)$ denotes the set of all finite sets $L \subseteq Loc$. The main question is how to define the W-sorted signature Σ. The basic idea is the same as for PCF in [Sie92]: In order to achieve full abstraction we must keep our denotational model 'as small as possible' and to this end we try to make the signature as large as possible. For PCF this was easy to achieve. We started from a flat ground type of integers and defined Σ to be the set of all ground type relations which are preserved by the (intended) meanings of the first order constants. This worked out, because all relations on a flat dcpo are closed under lubs of directed sets. For Alg the situation is more difficult, because the ground types $[\![iexp]\!]$ and $[\![cmd]\!]$ will certainly be *not* flat. Thus, in order to adapt the ideas of [Sie92] to the Alg setting, we introduce an additional semantic layer of flat dcpos *below* $[\![iexp]\!]$ and $[\![cmd]\!]$, and on this new layer we define certain auxiliary functions, which are closely related to the intended meanings of the Alg-constants.

Let $\Delta = \{loc, int, sto\}$, where int (= 'integer') and sto (= 'store') are auxiliary symbols. We use $sto \Rightarrow int$ and $sto \Rightarrow sto$ as alternative notation for $iexp$ and cmd. For every $\delta \in \Delta$ we define a dcpo D^δ by

$$D^{loc} = Loc \quad \text{(discrete dcpo)} \qquad D^{int} = \mathbb{Z}_\perp,\ D^{sto} = Stores_\perp \quad \text{(flat dcpos)}$$

where $Stores$ is the set of *stores* s, defined by

$$Stores = \bigcup_{L \in W} Stores_L \quad \text{with} \quad Stores_L = \{s : Loc \to \mathbb{Z} \,|\, \forall l \in Loc \setminus L.\, s\, l = 0\}$$

The set AUX of *auxiliary functions* consists of $Succ$, $Pred$, $Cont$, $Asgn$, $Const_n$ ($n \in \mathbb{N}$), $Cond_\delta$ ($\delta \neq loc$) and $Pcond$, where e.g.

$$\begin{array}{ll} Cont : \quad D^{loc} \to D^{sto} \to D^{int} & Asgn : \quad D^{loc} \to D^{int} \to D^{sto} \to D^{sto} \\ Cont\, l\, s = \begin{cases} \perp & \text{if } s = \perp \\ s\, l & \text{otherwise} \end{cases} & Asgn\, l\, d\, s = \begin{cases} \perp & \text{if } d = \perp \text{ or } s = \perp \\ s[d/l] & \text{otherwise} \end{cases} \end{array}$$

The list of the remaining functions is given in Appendix A.

As relation symbols of our signature we use so-called ground relations. By a *ground relation* of *arity* n we mean a triple $R = (R^\delta)_{\delta \in \Delta}$ such that $R^\delta \subseteq (D^\delta)^n$ for every $\delta \in \Delta$. We let $GRel_n$ denote the set of all ground relations of arity n, and we say that $f : D^{\delta_1} \to \ldots \to D^{\delta_k} \to D^\delta$ *preserves* $R \in GRel_n$ if $f R^{\delta_1} \ldots R^{\delta_k} \subseteq R^\delta$. Then we define $\Sigma = (\Sigma_n^L)_{L \in W, n \in \mathbb{N}}$ with

$$\begin{array}{rl} \Sigma_n^L = \{R \in GRel_n \,|\, & (\perp, \ldots, \perp) \in R^{sto}, \text{ every } f \in AUX \text{ preserves } R \\ & \text{and } \exists L' \in W.\, L \cap L' = \emptyset \wedge \delta^n(Loc \setminus L') \subseteq R^{loc}\,\} \end{array}$$

Finally we associate a W-Σ-lcpo $[\![\tau]\!] = (D^\tau, \mathcal{I}^\tau)$ with each type τ by

- $D_L^{loc} = L$
  $D^{loc} = Loc \quad$ (as before)
  $\mathcal{I}^{loc}(R) = R^{loc}$
- $D_L^{sto \Rightarrow \delta} = \{f : D^{sto} \to D^\delta \,|\, f \text{ preserves all } R \in \Sigma^L\}$
  $D^{sto \Rightarrow \delta} = \bigcup_{L \in W} D_L^{sto \Rightarrow \delta} \quad$ with the pointwise order on functions
  $\mathcal{I}^{sto \Rightarrow \delta}(R) = \{\vec{f} \in (D^{sto \Rightarrow \delta})^n \,|\, \vec{f} R^{sto} \subseteq R^\delta\} \quad$ if $R \in \Sigma_n$
- $[\![\tau \to \sigma]\!] = ([\![\tau]\!] \to [\![\sigma]\!]) \quad$ as defined in Theorem 1

Following usual mathematical convention we use $[\![\tau]\!]$ also as a notation for the W-lcpo (or the partial order or the set) D^τ, hence $[\![\tau]\!]_L$ denotes the dcpo D_L^τ. Moreover, we use R^τ as an abbreviation for $\mathcal{I}^\tau(R)$. From the definitions in Section 3 we then obtain the following important 'reasoning principles':

- $[\![\tau]\!] = \bigcup_{L \in W} [\![\tau]\!]_L$

- $[\![\tau \to \sigma]\!]_L \, [\![\tau]\!]_{L'} \subseteq [\![\sigma]\!]_{L'}$ whenever $L \subseteq L'$
- $f R^\tau \subseteq R^\sigma$ whenever $f \in [\![\tau \to \sigma]\!]_L$ and $R \in \Sigma^L$
- $(R^\tau)_{\tau \in Type}$ is a logical relation [Mit90] for every $R \in \Sigma$

To conclude the definition of the denotational semantics we must assign meanings $[\![c]\!]$ to the ALG-constants c. Some interesting cases are

$$\begin{array}{ll} [\![cont]\!] : [\![loc]\!] \to D^{sto} \to D^{int} & [\![asgn]\!] : [\![loc]\!] \to [\![iexp]\!] \to D^{sto} \to D^{sto} \\ [\![cont]\!] = Cont & [\![asgn]\!]\, l f s = Asgn\, l\, (f s)\, s \end{array}$$

$$\begin{array}{l} [\![seq_{sto \Rightarrow \delta}]\!] : [\![cmd]\!] \to [\![sto \Rightarrow \delta]\!] \to D^{sto} \to D^{\delta} \\ [\![seq_{sto \Rightarrow \delta}]\!] f g\, s = g\, (f s) \end{array}$$

$$\begin{array}{l} [\![new_{cmd}]\!] : [\![loc \to cmd]\!] \to D^{sto} \to D^{sto} \\ [\![new_{cmd}]\!] f s = Asgn\, l\, (Cont\, l\, s)\, (f\, l\, (Asgn\, l\, 0\, s)) \quad \text{with } l = next\,(support\,(f)) \end{array}$$

where $next : \mathcal{P}_f(Loc) \to Loc$ is an arbitrary function with $next\,(L) \notin L$ for every $L \in \mathcal{P}_f(Loc)$ and $support\,(d)$ is defined to be the set $\bigcap \{L \,|\, d \in [\![\tau]\!]_L\}$ for every $d \in [\![\tau]\!]$. The meanings of the remaining constants are given in Appendix B. The functions $[\![c]\!]$ are indeed contained in the model, more precisely:

Proposition 1 *If c is a constant of type σ, then $[\![c]\!] \in [\![\sigma]\!]_\emptyset$.*

Theorem 1 and Proposition 1 allow us to define the meaning of ALG-expressions in the style of the simply typed λ-calculus. Thus, for every expression $M : \tau$, we obtain a function $[\![M]\!] : Env \to [\![\tau]\!]$ where Env is the set of *environments* (= type preserving functions $\eta : \bigcup_\tau Id^\tau \to \bigcup_\tau [\![\tau]\!]$). The meaning function is extended to generalized expressions by defining $[\![l]\!] = l$ for every $l \in Loc$, and this leads to

Proposition 2 *Let $M : \tau$ be a generalized expression, let $\eta \in Env$ and let $L \in W$ be such that $locns\,(M) \subseteq L$ and $\eta x^{\tau'} \in [\![\tau']\!]_L$ for all free identifiers $x^{\tau'}$ in M. Then $[\![M]\!]\eta \in [\![\tau]\!]_L$. In particular*[3] *$[\![M]\!] \in [\![\tau]\!]_L$ whenever $M \in Exp^\tau_L$.*

The latter statement captures our intuition that a closed generalized expression has only access to those locations which explicitly occur in it and *not* to those which are temporarily bound to its local variables.

We finally remark that the particular choice of l in the clause for $[\![new_{cmd}]\!]$ does not play a role, i.e. instead of $next\,(support\,(f))$ we can use any other location $l \in Loc \setminus support\,(f)$. Thus we obtain for *every* $l \in Loc \setminus support\,([\![\lambda\, x.\, M]\!]\, \eta)$

$$[\![\mathbf{new}\; x \;\mathbf{in}\; M]\!]\, \eta\, s \;=\; \begin{cases} ([\![M]\!]\, \eta[l/x]\, s[0/l])\, [sl/l] & \text{if } [\![M]\!]\, \eta[l/x]\, s[0/l] \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

This possibility to choose the new location l freely from an infinite set is another important reasoning principle which we will use in the following.

[3] As usual we abbreviate $[\![M]\!]\,\eta$ by $[\![M]\!]$ if M is closed.

5 Reasoning about Local Variables

Notation: If A and B are sets, then $(A \xrightarrow{t} B)$ denotes the set of total functions from A to B. If $f, g \in (A \xrightarrow{t} B)$ and $C \subseteq A$, then $f \,|\, C$ denotes the restriction of f to C and $f =_C g$ stands for $f \,|\, C = g \,|\, C$.

We will now prove some basic properties of our model and illustrate by an example how semantic equivalences can be proved. The following set of ground relations will be useful for both purposes.

Definition 4 *Let $L \in W$. An n-ary ground relation R is called L-*definable*, if there is a relation $R_L \subseteq (L \xrightarrow{t} \mathbb{Z})^n$ such that*

- $R^{sto} = \{\bot\}^n \cup \{\vec{s} \in \mathit{Stores}^n \mid (\vec{s} \,|\, L) \in R_L \wedge \vec{s}\,(\mathit{Loc} \setminus L) \subseteq \delta^n \mathbb{Z}\}$
- $R^{int} = \delta^n D^{int}$
- $R^{loc} = \{\vec{l} \in (D^{loc})^n \mid \mathit{Cont}\,\vec{l}\,R^{sto} \subseteq R^{int} \wedge \mathit{Asgn}\,\vec{l}\,R^{int}\,R^{sto} \subseteq R^{sto}\}$

Note that an L-definable ground relation is uniquely determined by R^{sto}. We let DEF^L denote the set of L-definable ground relations.

Theorem 2 *Let $L, L' \in W$ with $L \cap L' = \emptyset$. Then $\mathit{DEF}^{L'} \subseteq \Sigma^L$.*

Proposition 3 *Let $L \in W, f \in [\![cmd]\!]_L, l \in \mathit{Loc} \setminus L$ and $s, s_1, s_2 \in \mathit{Stores}$. Then*

(a) $f \bot = \bot$

(b) $f s \neq \bot \Rightarrow f s\, l = s\, l$

(c) $s_1 =_L s_2 \Rightarrow (f s_1 = \bot = f s_2 \vee (f s_1, f s_2 \in \mathit{Stores} \wedge f s_1 =_L f s_2))$

Proof: Each of the three properties is proved by choosing an appropiate $R \in \Sigma^L$ and exploiting the fact that $f R^{sto} \subseteq R^{sto}$. For (a) we take $R \in \mathit{DEF}^{\emptyset}$ with $R^{sto} = \{\bot\}$, for (b) we take $R \in \mathit{DEF}^{\{l\}}$ with $R^{sto} = \{\bot\} \cup \{t \in \mathit{Stores} \mid t\, l = s\, l\}$ and for (c) we choose some $L' \in W$ with $L \cap L' = \emptyset$ and $s_1 =_{\mathit{Loc} \setminus L'} s_2$ and take $R \in \mathit{DEF}^{L'}$ with $R^{sto} = \{\bot\}^2 \cup \{\vec{t} \in \mathit{Stores}^2 \mid t_1 =_{\mathit{Loc} \setminus L'} t_2\}$. $\square$

The following example of a semantic equivalence will be needed in the full abstraction proof but is also interesting in its own.

Example 1 $\quad [\![y^{cmd \to cmd} z^{cmd}]\!] = [\![\textbf{new}\ x\ \textbf{in}\ x := 0;\ y\,(x := !\,x + 1;\ z)]\!]$

The local variable x is used here for counting the procedure calls of z (as long as no snap back effect occurs) during the computation of $y\,z$.[4] The equivalence

[4] Note that ALG, as a full-fledged λ-calculus, allows us to use an expression of type *cmd* on parameter position where ALGOL 60 would force us to introduce a new procedure identifier. Call-by-name ensures that the assignment $x := !\,x + 1$ is executed *whenever* y uses its parameter (and not only *once*, as in a call-by-value language).

shows that adding such a bookkeeping mechanism does not change the behavior of the program in which the procedure call $y\,z$ is contained, no matter how the procedures y and z are declared.

The typical approach for proving such an equivalence between two expressions is to find some $R \in \Sigma$ which (intuitively) relates corresponding states of their computations. The precise argumentation for Example 1 is as follows:

Let $\eta \in Env$ and $s \in Stores$. Let $L \in W$ with $\eta\, y \in [\![cmd \to cmd]\!]_L$ and $\eta\, z \in [\![cmd]\!]_L$. We may assume that the new location l is not in L and define $R \in DEF^{\{l\}}$ by $R^{sto} = \{\bot\}^2 \cup \{\vec{t} \in Stores^2 \mid t_1 =_{Loc\setminus\{l\}} t_2\}$. Then $(s, s[0/l]) \in R^{sto}$ and $(\eta\, z, [\![x := !\,x+1;\ z]\!]\,\eta[l/x]) \in R^{cmd}$, because—by part (c) of Proposition 3—$t_1 =_{Loc\setminus\{l\}} t_2$ always implies $\eta\, z\, t_1 =_{Loc\setminus\{l\}} \eta\, z\, t_2 =_{Loc\setminus\{l\}} [\![x := !\,x+1;\ z]\!]\,\eta[l/x]\,t_2$. Thus we obtain

$$([\![y\,z]\!]\,\eta\,s,\ [\![y\,(x := !\,x+1;\ z)]\!]\,\eta[l/x]\,s[0/l]) \ \in\ \eta\,y\,R^{cmd}R^{sto} \ \subseteq\ R^{cmd}R^{sto} \ \subseteq\ R^{sto}$$

and this implies $[\![y\,z]\!]\,\eta\,s = [\![\mathbf{new}\ x\ \mathbf{in}\ x := 0;\ y\,(x := !\,x+1;\ z)]\!]\,\eta\,s$.

6 Full Abstraction

Notation: If $\sigma = \tau_1 \to \ldots \to \tau_k \to sto \Rightarrow \delta$ $(k \geq 0)$ and $f \in [\![\sigma]\!]$, then we let f^d denote the *completely decurried version* of f, i.e.

$$f^d\colon [\![\tau_1]\!] \times \ldots \times [\![\tau_k]\!] \times D^{sto} \to D^{\delta} \quad \text{with} \quad f^d(d_1, \ldots, d_k, s) = f\,d_1 \ldots d_k\, s$$

and if $p \in ([\![\sigma]\!] \xrightarrow{t} [\![\sigma]\!])$, then we let p^D denote the corresponding function on the completely decurried versions, i.e.

$$p^D\colon [\![\sigma]\!]^d \to [\![\sigma]\!]^d \quad \text{with} \quad p^D f^d = (p\,f)^d$$

The first step towards full abstraction is

Theorem 3 *Let $\sigma = \tau_1 \to \ldots \to \tau_k \to \theta$ $(k \geq 0)$ with $ord(\sigma) \leq 2$. Let $L \in W$, $f \in [\![\sigma]\!]_L$ and let $B \subseteq [\![\tau_1]\!] \times \ldots \times [\![\tau_k]\!] \times D^{sto}$ be finite. Then there is some $M \in Exp_L^{\sigma}$ with $[\![M]\!]^d =_B f^d$.*

For the proof of Theorem 3 one needs a ground relation $R \in \Sigma_n^L$ where n is the cardinality of B. Hence it is important that we have relations of arbitrary arity in our model. We do not present any details here, because we want to concentrate on the remaining (more interesting) steps of the full abstraction proof.

From Theorem 3 we could obtain a sequence of definable functions which 'approximate' f in the sense that they coincide with f on more and more argument tuples. But instead we need approximations in the Scott topology, i.e. we must show that f is the *least upper bound* of a sequence (or a directed set) of definable functions. In order to bridge the gap between these two different notions of approximation we introduce the following concepts.

Definition 5

(a) *Let D, E be sets, $F \subseteq (D \xrightarrow{t} E)$ and $p \in (F \xrightarrow{t} F)$. $B \subseteq D$ is called a* base set *for p, if pf is uniquely determined by $f \mid B$, i.e. if $f =_B g$ implies $pf = pg$ for all $f, g \in F$. p is called* finitely based *if it has a finite base set.*

(b) *Let σ be a procedure type and let $L \in W$. An L-*projection sequence *on σ is a sequence of expressions $P_n \in Exp_L^{\sigma \to \sigma}$ such that $[\![P_n]\!]^D \mid ([\![\sigma]\!]_L)^d$ is finitely based for every $n \in \mathbb{N}$ and $([\![P_n]\!])_{n \in \mathbb{N}}$ is an ω-chain whose lub is the identity on $[\![\sigma]\!]$. σ is called an L-*limit *if an L-projection sequence exists on σ.*

If we can show that every procedure type of order 1 or 2 is an L-limit for every $L \in W$, then we obtain the desired approximations as follows.

Theorem 4 *Let $ord(\sigma) \leq 2$ and $L \in W$. Then every $f \in [\![\sigma]\!]_L$ is the lub of an ω-chain of functions which are definable by expressions in Exp_L^σ.*

Proof: Let $(P_n)_{n \in \mathbb{N}}$ be an L-projection sequence on σ, and for every $n \in \mathbb{N}$ let B_n be a finite base set for $([\![P_n]\!])^D \mid ([\![\sigma]\!]_L)^d$. By Theorem 3 there are expressions $M_n \in Exp_L^\sigma$ with $[\![M_n]\!]^d =_{B_n} f^d$, hence $[\![P_n M_n]\!]^d = [\![P_n]\!]^D [\![M_n]\!]^d = [\![P_n]\!]^D f^d = ([\![P_n]\!] f)^d$ for every $n \in \mathbb{N}$. This implies $f = \bigsqcup_{n \in \mathbb{N}} [\![P_n]\!] f = \bigsqcup_{n \in \mathbb{N}} [\![P_n M_n]\!]$. $\square$

In the absence of an operational semantics[5] we use the 'internal' definition of full abstraction which only refers to the denotational model: Two expressions are *observationally congruent*, if they can be replaced by each other in every program context without changing the meaning of the program. A denotational semantics is *fully abstract* if semantic equivalence coincides with observational congruence. Thus our main result reads as follows:

Theorem 5 (Full Abstraction) *Let $ord(\sigma) \leq 3$ and $M_1, M_2 \in Exp_\emptyset^\sigma$. Then*

$$[\![M_1]\!] = [\![M_2]\!] \quad \Leftrightarrow \quad [\![C[M_1]]\!] = [\![C[M_2]]\!] \text{ for every program context } C[\ ]$$

Proof: As usual, only '$\Leftarrow$' must be proved, because '$\Rightarrow$' already follows from the compositionality of the denotational semantics. Let $\sigma = \tau_1 \to \ldots \to \tau_k \to \theta$ $(k \geq 0)$ and assume that $[\![M_1]\!] \neq [\![M_2]\!]$, i.e. there are $d_j \in [\![\tau_j]\!]$ for $j = 1, \ldots, k$ and $s \in Stores$ such that

$$[\![M_1]\!]\, d_1 \ldots d_k\, s \quad \neq \quad [\![M_2]\!]\, d_1 \ldots d_k\, s$$

Let $L \in W$ be such that $d_j \in [\![\tau_j]\!]_L$ for $j = 1, \ldots, k$. By Theorem 4, every d_j is the lub of an ω-chain of definable elements in $[\![\tau_j]\!]_L$, hence the local continuity of $[\![M_1]\!]$ and $[\![M_2]\!]$ implies that there are $N_j \in Exp_L^{\tau_j}$ with

$$[\![M_1 N_1 \ldots N_k]\!]\, s \quad \neq \quad [\![M_2 N_1 \ldots N_k]\!]\, s$$

[5]An operational semantics (which is interesting in its own because of the snap back effect) has been presented in [Sie92].

From this, it is easy to construct a program context $C[\,]$ with $[\![C[M_1]]\!] \neq [\![C[M_2]]\!]$ (location constants in $N_1, \ldots, N_k$ must be replaced by local variables and these local variables must be initialized according to s). $\square$

We have formulated Theorem 5 for *closed* expressions of order ≤ 3. Alternatively we could have used *open* expressions of order ≤ 2 whose only free identifiers are of order ≤ 2; that's why we speak of 'full abstraction for the second order subset'.

The main challenge remains to prove

Theorem 6 *Let $ord(\sigma) \leq 2$ and $L \in W$. Then σ is an L-limit.*

Proof sketch: The proof for the first order types is simple. As an example we consider $\sigma = cmd$. For every $n \in \mathbb{N}$ we define[6]

$$\begin{array}{lcllcl} P_n^{sto} & \equiv & \textbf{if} \bigwedge_{l \in L} abs(!\,l) \leq n \textbf{ then } skip \textbf{ else } \Omega & \in & Exp_L^{cmd} \\ P_n^{cmd} & \equiv & \lambda\, y^{cmd}.\, (P_n^{sto};\, y;\, P_n^{sto}) & \in & Exp_L^{cmd \to cmd} \end{array}$$

It is easy to see that $([\![P_n^{cmd}]\!])_{n \in \mathbb{N}}$ is an ω-chain of (idempotent) functions whose lub is the identity on $[\![cmd]\!]$. Now let $f \in [\![cmd]\!]_L$. By Proposition 3, $[\![P_n^{cmd}]\!]f \in [\![cmd]\!]_L$ is uniquely determined by its restriction to $Stores_L$. Hence let $B = [\![P_n^{sto}]\!]Stores_L$. B is finite, and $[\![P_n^{cmd}]\!]f \,|\, Stores_L$ is uniquely determined by $f \,|\, B$ or even by $([\![P_n^{sto}]\!] \circ f) \,|\, B$. The former shows that B is a base set for $[\![P_n^{cmd}]\!] \,|\, [\![cmd]\!]_L$, i.e. $(P_n^{cmd})_{n \in \mathbb{N}}$ is an L-projection sequence on cmd (decurrying is not an issue here). The latter shows that $[\![P_n^{cmd}]\!][\![cmd]\!]_L$ is finite (i.e. $[\![cmd]\!]_L$ is an SFP-domain), because $([\![P_n^{sto}]\!] \circ f) \,|\, B$ can only range over the finitely many functions on B.

The proof for the second order types is rather sophisticated; we only sketch the main ideas for a single case, namely $\sigma = cmd \to cmd$. The first idea which comes to mind is to define $P_n^\sigma \in Exp_L^{\sigma \to \sigma}$ completely analogous to P_n^{cmd}, namely

$$P_n^\sigma \quad \equiv \quad \lambda\, y^\sigma.\, \lambda\, z^{cmd}.\, P_n^{cmd}\,(y\,(P_n^{cmd} z))$$

If the elements of $[\![\sigma]\!]_L$ were just functions from $[\![cmd]\!]_L$ to $[\![cmd]\!]_L$, then we could prove—by a similar argumentation as above—that $[\![P_n^{cmd}]\!][\![cmd]\!]_L \times [\![P_n^{sto}]\!]Stores_L$ is a (finite) base set for $[\![P_n^\sigma]\!]^D \,|\, ([\![\sigma]\!]_L)^d$. But of course this assumption is wrong and indeed it can be shown that $[\![P_n^\sigma]\!]^D \,|\, ([\![\sigma]\!]_L)^d$ does not have a finite base set.

Somewhat to our own surprise, a slight modification of the expressions P_n^σ suffices to solve this problem. For every $n \in \mathbb{N}$ let

$$\begin{array}{lcl} \bar{P}_n^\sigma & \equiv & \lambda y^\sigma.\, \lambda z^{cmd}. \\ & & \quad \textbf{new } x \textbf{ in } x := 0;\, P_n^{cmd}(y\,(\textbf{if } !\,x < n \textbf{ then } x := !\,x + 1;\; P_n^{cmd} z \textbf{ else } \Omega)) \end{array}$$

[6]Symbols like $\wedge, abs, \leq, \ldots$ are used with their standard interpretations; they are of course definable in ALG. Ω denotes the always diverging command $\mathrm{Y}_{cmd}(\lambda\, z^{cmd}.\, z)$.

$\bar{P}_n^\sigma$ differs from P_n^σ by using a local variable x to count the procedure calls of y's parameter $P_n^{cmd} z$, and as soon as the number of these procedure calls exceeds n, it enforces divergence. It is easy to see that $([\![\bar{P}_n^\sigma]\!])_{n\in\mathbb{N}}$ is an ω-chain with

$$\bigsqcup_{n\in\mathbb{N}} [\![\bar{P}_n^\sigma]\!] \quad = \quad [\![\lambda\, y.\, \lambda\, z.\, \mathbf{new}\, x \ \mathbf{in}\ x := 0;\ y\,(x := !\, x + 1;\ z)]\!]$$

and by Example 1 the right hand side equals the identity $[\![\lambda\, y.\, \lambda\, z.\, y\, z]\!]$. Hence it remains to be shown that every $[\![\bar{P}_n^\sigma]\!]^D \,|\, ([\![\sigma]\!]_L)^d$ has a finite base set.

To this end let $[\![P_n^{sto}]\!] Stores_L \setminus \{\bot\} = \{s_1, \ldots, s_k\}$ and let $l \in Loc \setminus L$. We may assume that sequences $w \in \{1, \ldots, k\}^*$ can be stored into l (by encoding them as integers). We let $Hist_n = \{w \in \{1, \ldots, k\}^* \,\big|\, |w| < n\}$, and for every function $\Phi : Hist_n \to [\![P_n^{cmd}]\!][\![cmd]\!]_L$ we define $c_\Phi \in [\![cmd]\!]_{L\cup\{l\}}$ by

$$c_\Phi s \quad = \quad \begin{cases} \Phi(sl)(s[i.sl/l]) & \text{if } sl \in Hist_n \ \wedge\ s =_L s_i \\ \bot & \text{if } sl \notin Hist_n \ \vee\ s \notin [\![P_n^{sto}]\!] Stores \end{cases}$$

Then it turns out that the finite set

$$B \quad = \quad \{(c_\Phi, s_i[\varepsilon/l]) \,\big|\, \Phi : Hist_n \to [\![P_n^{cmd}]\!][\![cmd]\!]_L,\ i \in \{1, \ldots, k\}\}$$

is a base set for $[\![\bar{P}_n^\sigma]\!]^D \,|\, ([\![\sigma]\!]_L)^d$. The details of the proof are too complicated to be presented here, but we want to provide some intuition:

Every procedure of the form c_Φ uses the location l for keeping a record of its own history of procedure calls and diverges as soon as the length of this history exceeds n; the index Φ describes how a call of c_Φ depends on the previously recorded history. Now let $f \in [\![\sigma]\!]_L$. Then, for every $g \in [\![cmd]\!]$ and $s \in Stores$, the computation for $[\![\bar{P}_n^\sigma]\!] f g s$ can be simulated by the computation for $[\![\bar{P}_n^\sigma]\!] f c_\Phi (s_i[\varepsilon/l])$ with some appropiate Φ and i, and this implies in turn that $([\![\bar{P}_n^\sigma]\!] f)^d$ is uniquely determined by $f^d \,|\, B$, i.e. that B is indeed a base set. The simulation is defined in such a way that calls of g exactly correspond to calls of c_Φ. It comes as a certain surprise that such a simulation is possible, because—on the one hand—g may have access to (finitely but) arbitrarily many locations outside L whose contents can in no way be restricted by $[\![\bar{P}_n^\sigma]\!] f \in [\![\sigma]\!]_L$ and—on the other hand—the c_Φ's only use a single additional location l in which they only store values from a finite set. The crucial point is that the contents of the locations outside L need not be *explicitly* encoded into the contents of l, because they are *implicitly* determined by the recorded history of procedure calls. □

7 Conclusion

We have already mentioned that the parallel conditional is not important for our result. In order to obtain the same full abstraction result for *sequential* ALG (without *pcond*), we can simply remove the function *Pcond* from *AUX* and then proceed as before. Thus we obtain a model with a larger signature Σ, in which additional semantic equivalences hold, e.g.

$$[\![y\, skip\, \Omega + y\, \Omega\, skip]\!] = [\![y\, \Omega\, \Omega + y\, \Omega\, \Omega]\!] \quad (\text{with } y : \mathit{cmd} \to \mathit{cmd} \to \mathit{iexp})$$

a variant of the famous observational congruence for sequential PCF [Plo77]. Following [Sie92] we can prove this equivalence with the aid of a ternary ground relation R, namely

$$R^{loc} = \delta^3 Loc, \quad R^{\delta} = \{\vec{d} \in (D^{\delta})^3 \,|\, d_1 = \bot \vee d_2 = \bot \vee d_1 = d_2 = d_3\} \ (\delta \neq loc)$$

On the other hand we can show that *no binary* relation works for this example, and by similar examples one sees that relations of any fixed arity n are not sufficient for reasoning about sequential Alg. For Alg itself we have not found such examples, hence it remains an open question whether binary relations as in [OT93a, OT93b] or relations of some fixed arity n are sufficient in the presence of a parallel conditional.

An interesting question is of course, what happens at types of order ≥ 3. We conjecture that neither our model nor the models in [OT93a, OT93b] are fully abstract for these higher types: Reasoning about local variables is closely related to the question of λ-definability (the intuition is that a global procedure acts on a local variable like a pure λ-term), and it follows from [Loa93] that (at least over finite ground types) λ-definability for functions of order ≥ 3 cannot be characterized with the aid of logical relations. As all the above models are based on logical relations, it seems unlikely that one of them be fully abstract for types of order ≥ 3. Hence our result seems the best one may expect for the current state of the art.

Acknowledgements. I'm grateful to Peter O'Hearn and Bob Tennent for discussions about the relationship between our approaches and to Jörg Zeyer for pointing out unclarities in an earlier draft.

References

[Len93] Arthur F. Lent. The category of functors from state shapes to bottomless cpos is adequate for block structure. In *Proc. ACM SIGPLAN Workshop on State in Programming Languages (Technical Report YALEU/DCS/RR-968, Yale University)*, pages 101–119, Copenhagen, Denmark, 1993.

[Loa93] Ralph Loader. The undecidability of λ-definability. Technical report, Mathematical Institute, Oxford University, June 1993.

[Mit90] John C. Mitchell. Type systems for programming languages. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, volume B*, chapter 8, pages 365–458. North-Holland, 1990.

[MS88] Albert R. Meyer and Kurt Sieber. Towards fully abstract semantics for local variables: Preliminary report. In *Proc. 15^{th} Annual ACM Symp. on Principles of Programming Languages*, pages 191–203, San Diego, 1988.

[OT] Peter W. O'Hearn and Robert D. Tennent. Personal communication.

[OT93a] Peter W. O'Hearn and Robert D. Tennent. Parametricity and local variables. Technical Report SU-CIS-93-30, School of Computer and Information Science, Syracuse University, October 1993.

[OT93b] Peter W. O'Hearn and Robert D. Tennent. Relational parametricity and local variables. In *Proc. 20^{th} Annual ACM Symposium on Principles of Programming Languages*, pages 171–184, 1993.

[Plo77] Gordon D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–256, 1977.

[PS93] Andrew M. Pitts and Ian D. B. Stark. Observable properties of higher order functions that dynamically create local names, or: What's *new*? In Andrzej M. Borzyszkowski and Stefan Sokołowski, editors, *Proc. 18th International Symposium on Mathematical Foundations of Computer Science*, LNCS 711, pages 122–141. Springer-Verlag, 1993.

[Rey81] John C. Reynolds. The essence of ALGOL. In J. deBakker and van Vliet, editors, *Int'l. Symp. Algorithmic Languages*, pages 345–372. IFIP, North-Holland, 1981.

[Sie92] Kurt Sieber. Reasoning about sequential functions via logical relations. In M. P. Fourman, P. T. Johnstone, and A. M. Pitts, editors, *Proc. LMS Symposium on Applications of Categories in Computer Science, Durham 1991*, LMS Lecture Note Series 177, pages 258–269. Cambridge University Press, 1992.

[Sie93] Kurt Sieber. New steps towards full abstraction for local variables. In *Proc. ACM SIGPLAN Workshop on State in Programming Languages (Technical Report YALEU/DCS/RR-968, Yale University)*, pages 88–100, Copenhagen, Denmark, 1993.

[WF93] Stephen Weeks and Matthias Felleisen. On the orthogonality of assignments and procedures in Algol. In *Proc. 20^{th} Annual ACM Symposium on Principles of Programming Languages*, pages 57–70, 1993.

A List of the remaining auxiliary functions

- $\mathit{Succ}: \quad D^{int} \to D^{int}$

  $$\mathit{Succ}\, d = \begin{cases} \perp & \text{if } d = \perp \\ d + 1 & \text{otherwise} \end{cases}$$

- $\mathit{Pred}: \quad D^{int} \to D^{int}$

  $$\mathit{Pred}\, d = \begin{cases} \perp & \text{if } d = \perp \\ d \perp 1 & \text{otherwise} \end{cases}$$

- $\mathit{Const}_n: \quad D^{sto} \to D^{int}$

  $$\mathit{Const}_n\, s = \begin{cases} \perp & \text{if } s = \perp \\ n & \text{otherwise} \end{cases}$$

- $\mathit{Cond}_\delta: \quad D^{int} \to D^\delta \to D^\delta \to D^\delta$

  $$\mathit{Cond}_\delta\, b\, d_1 d_2 = \begin{cases} \perp & \text{if } b = \perp \\ d_1 & \text{if } b = 0 \\ d_2 & \text{otherwise} \end{cases}$$

- $\mathit{Pcond}: \quad D^{int} \to D^{int} \to D^{int} \to D^{int}$

  $$\mathit{Pcond}\, b\, d_1 d_2 = \begin{cases} \perp & \text{if } b = \perp \text{ and } d_1 \neq d_2 \\ d_1 & \text{if } b = 0 \\ d_2 & \text{otherwise} \end{cases}$$

B Meanings of the remaining ALG-constants

$[\![n]\!]: \ D^{sto} \to D^{int}$
$[\![n]\!] = \mathit{Const}_n$

$[\![\mathit{skip}]\!]: \ D^{sto} \to D^{sto}$
$[\![\mathit{skip}]\!]\, s = s$

$[\![\mathit{succ}]\!]: \ [\![\mathit{iexp}]\!] \to D^{sto} \to D^{int}$
$[\![\mathit{succ}]\!] f s = \mathit{Succ}\,(f s)$

$[\![\mathit{pred}]\!]: \ [\![\mathit{iexp}]\!] \to D^{sto} \to D^{int}$
$[\![\mathit{pred}]\!] f s = \mathit{Pred}\,(f s)$

$[\![\mathit{pcond}]\!]: \ [\![\mathit{iexp}]\!] \to [\![\mathit{iexp}]\!] \to [\![\mathit{iexp}]\!] \to D^{sto} \to D^{int}$
$[\![\mathit{pcond}]\!]\, b f g\, s = \mathit{Pcond}\,(bs)\,(fs)\,(gs)$

$[\![\mathit{cond}_{sto \Rightarrow \delta}]\!]: \ [\![\mathit{iexp}]\!] \to [\![\mathit{sto} \Rightarrow \delta]\!] \to [\![\mathit{sto} \Rightarrow \delta]\!] \to D^{sto} \to D^\delta$
$[\![\mathit{cond}_{sto \Rightarrow \delta}]\!]\, b f g\, s = \mathit{Cond}_\delta(bs)\,(fs)\,(gs)$

$[\![\mathit{new}_{iexp}]\!]: \ [\![\mathit{loc} \to \mathit{iexp}]\!] \to D^{sto} \to D^{int}$
$[\![\mathit{new}_{iexp}]\!] f s = f\, l\,(\mathit{Asgn}\, l\, 0\, s) \quad \text{with } l = \mathit{next}\,(\mathit{support}\,(f))$

$[\![\mathrm{Y}_\sigma]\!]: \ [\![\sigma \to \sigma]\!] \to [\![\sigma]\!]$
$[\![\mathrm{Y}_\sigma]\!] f = \bigsqcup_{n \in \mathbb{N}} f^n \perp \quad$ (the least fixed point of f)