ELSEVIER Theoretical Computer Science 168 (1996) 155–212

Theoretical Computer Science

Full abstraction for the second order subset of an ALGOL-like language

Kurt Sieber

FB 14 Informatik, Universität des Saarlandes, 66041 Saarbrücken, Germany

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. As all the published 'test equivalences' [8, 13, 22] for ALGOL-like languages are contained in the second order subset, they can all be validated (easily) in our denotational model.

The general technique for our model construction – namely 'relationally structured locally complete partial orders' with 'relation preserving locally continuous functions' – has already been developed in [13], but our particular model differs from the one in [13] in that we now use a larger set of relations. In a certain sense it is the 'largest possible' set of relations, an idea which we have successfully used in [32] to obtain a fully abstract model for the second order subset of the functional language PCF [26]. The overall structure of our full abstraction proof is also taken from [32], but for the single parts of the proof we had to solve considerable new problems which are specific to the imperative (ALGOL-like) setting.

1. Introduction

Difficulties with the denotational semantics of local variables were first observed in the context of ALGOL-like languages in the early eighties [5, 29–31, 37]. In [13] these difficulties were identified more precisely as having to do with a failure of full abstraction. Roughly speaking, a denotational semantics is fully abstract if it does not make any unnecessary distinctions. A more precise definition is as follows: Two program pieces M and N are *observationally congruent* (denoted $M \approx N$), if they can be replaced by each other in every program without changing the observable behavior of the program. Every reasonable denotational semantics should *only* identify observationally congruent program pieces, i.e. it should satisfy

$$[\![M]\!] = [\![N]\!] \Rightarrow M \approx N$$

If it even identifies *all* such program pieces, i.e. if it satisfies

$$[\![M]\!] = [\![N]\!] \Leftrightarrow M \approx N$$

then it is called *fully abstract*.

For many programming languages it is difficult to find a fully abstract denotational semantics, the most prominent one being the purely functional language PCF [26]. The reason is that most denotational models contain *nonstandard elements*, i.e. elements which are not definable in the language, and that some of these elements may be very different in nature from the definable ones. Two observationally congruent program pieces may then fail to be denotationally equivalent just when their free identifiers are bound to such 'critical' nonstandard elements. This means that full abstraction fails, and then the challenge is to find a 'smaller' denotational model from which these critical elements are ruled out. Usually this goal is achieved when every nonstandard element is the limit (i.e. the least upper bound of a directed set) of definable elements, because then full abstraction follows in a standard way from the continuity of the semantic functions [26].

In the traditional model for PCF the critical nonstandard elements are the functions with a 'parallel' nature. An example is the *parallel or* [26], which returns *true* precisely when one of its arguments is *true* – even if the other argument diverges. The full abstraction problem for PCF thus amounts to defining a smaller model which only consists of 'sequential' functions. In [32] we solved this problem for the second order subset of PCF by admitting only those continuous functions which preserve certain (logical) relations. Here we will transfer this idea from the purely functional language PCF to the (much) more complicated setting of an imperative (ALGOL-like) language.

In traditional models for ALGOL-like languages [2,3,19], termed *marked store models* in [13], the critical elements are the functions which have 'access' to an unbounded number of locations. We briefly sketch the definition of such a model in order to obtain some hints how our new model should be constructed:

Let *Loc* be some infinite set, whose elements l are called *locations*, let $\mathscr{P}_{fin}(Loc)$ be the set of finite subsets of *Loc* and let $next : \mathscr{P}_{fin}(Loc) \to Loc$ be a function with $next(L) \notin L$ for all $L \in \mathscr{P}_{fin}(Loc)$. We define a *marked store* to be a partial function $ms : Loc \hookrightarrow \mathbb{Z}$ whose domain $dom(ms)$ is finite. We think of $dom(ms)$ as the set of locations which are marked as active, $ms\,l$ as the contents of location l, and $next(dom(ms))$ as the first 'free' location, i.e. the one to be allocated next.[1] The

[1] Of course there are innumerable variants of this idea, e.g. a marked store can be represented as a pair (L,s) with a *total* function s, or the mark L can be introduced as part of the *environment* η. The reader may be assured that the problems which we illustrate here appear with necessary changes made for such alternative definitions.

meaning of a block with a local variable declaration can then be defined by[2]

$$[\![\textbf{new } x \textbf{ in } M \textbf{ end}]\!]\,\eta\, ms = \begin{cases} \bot & \text{if } [\![M]\!]\,\eta[l/x]\,(ms[0/l]) = \bot \\ ms' \setminus l & \text{if } [\![M]\!]\,\eta[l/x]\,(ms[0/l]) = ms' \neq \bot \end{cases}$$

where $l = next(dom(ms))$. This definition simply imitates an operational semantics: Upon block entry the first free location l is bound to the local variable x, marked as active and set to the initial value 0. Then the block body M is executed and – if it terminates – the location l is finally deallocated by removing it from the domain of the resulting store ms'. Now consider the block[3]

$$B \equiv \textbf{new}\, x \textbf{ in } y;\ \textbf{if } !x = 0 \textbf{ then } \Omega \textbf{ end}$$

which contains a call of some global parameterless procedure y. It is easy to argue informally (and it will be rigorously proved in Section 7) that B is observationally congruent to the always diverging command Ω: Due to the static scope rule in ALGOL-like languages, the global procedure y does not have access to the local variable x, hence – if the call of y terminates at all – the variable x will still contain its initial value 0 after the call, and this makes the block diverge. On the other hand, B is *not* denotationally equivalent to Ω: Let $\eta\, y = f$ where f is the function which sets all active locations to 1, i.e.

$$f\, ms = ms' \quad \text{with} \quad dom(ms') = dom(ms) \text{ and } ms'\, l = 1 \text{ for all } l \in dom(ms)$$

Then even the new location which is bound to the local variable x will be set to 1 (because it is active when y is called) and this implies $[\![B]\!]\,\eta \neq [\![\Omega]\!]\,\eta$. Thus we have shown that every model which contains the above function f (and the traditional models do contain this function) fails to be fully abstract.

It is clear which lesson we have to learn from this example if we want to achieve full abstraction: We must define a model in which every function only has access to some fixed finite set of locations (in contrast to the above function f), and we must carefully choose the new location which we bind to a local variable, so that the functions which are bound to the global procedure identifiers do not have access to it. This means in particular that we must somehow formalize the notion of 'access'. For first order procedures, it is easy to give an *ad hoc* definition (cf. Theorem 8.1), but when it comes to second order then we need a more systematic approach – and this is the point where (logical) relations come again into play.

The idea to use relations for constructing models of ALGOL-like languages originates with [13], but the particular model which was presented there, failed to be fully abstract because the set of relations was too small. The idea was resumed in [22, 33] with larger sets of relations, thus leading to improved models in which all the known test equivalences [8, 13, 22] for ALGOL-like languages could be validated, but a full

[2] In ALGOL 60 syntax such a block is written as **begin integer** x; M **end**

[3] We use the ML-notation $!x$ for explicitly dereferencing a variable x.

abstraction proof for these models was still missing. Here we will present such a proof for (a slight variant of) the model in [33].

It should be mentioned that our motivation for using logical relations is somewhat different from O'Hearn and Tennent's motivation in [22]. Their intention was to transfer Reynolds' concept of 'relational parametricity' [11] from polymorphic languages to ALGOL-like languages, because they see a close relationship between information hiding through type variables (in a polymorphic language) and information hiding through local variables. Our own view is somewhat more technical: We know that logical relations can often be used to characterize the definable functions [6, 27] or – if the model consists of dcpo's – the limits of definable functions [32]. Hence we try to use them a priori (as in [20]) to construct a model in which *all* elements are limits of definable ones, so that we obtain full abstraction by the standard continuity argument. Although it is not clear whether this technique works for *every* language, it is certainly not limited to polymorphism and local variables; rather it is based on the close relationship between definability and full abstraction.

Let us now briefly review the main concepts of an ALGOL-like language: Due to [5, 8, 30, 37] it should be a simply typed, statically scoped call-by-name language which obeys the so-called *stack discipline*. The latter means that a location never survives the block in which it has been allocated; this is considered as a semantic principle and not as a matter of implementation. Finally, there should be a clear distinction between locations and storable values, and also between commands and expressions: Commands alter the store but do not return values, expressions return values but do not have (permanent) side effects on the store.

In order to obtain our full abstraction result we had to include one somewhat unusual feature in the language ALG, namely the so-called *snap back effect* which goes back to a suggestion of J.C. Reynolds': Inside the body of a function procedure, assignments to global variables are allowed, but after each call of such a procedure the store 'snaps back' to the contents which it had before the call, i.e. only a *temporary* side effect is caused by such an assignment. We will allow the snap back effect to occur in arbitrary (integer) expressions. For an ALGOL-like language *without* snap back, our model definitely *fails* to be fully abstract, but our general techniques may well be suited for constructing a (different) fully abstract model for such an alternative language (Section 10). This is subject to further research.

The snap back effect can be considered as unnatural for an ALGOL-like language, because it violates the *single threadedness* of state which usually holds for imperative languages: Whenever a state change occurs, the old state is no longer available and hence there is no way to backtrack to an earlier state. But from a theoretical point of view this same observation is quite useful: Including the snap back effect in our language allows us to study the nature of local variables *in isolation* from other issues like single threadedness. Hence it seems not only legitimate but even reasonable to have such a feature in our language.

As another unusual feature we have included a *parallel conditional*. This operator often plays a prominent role in full abstraction proofs [26], but here it does *not*. If we remove it from ALG then we still obtain a (different) fully abstract model with the same techniques as before (Section 10). The interesting point is that the parallel conditional may allow us to simplify our model definition, namely, to use only relations of arity $\leqslant 2$ of a particular shape (Conjecture 11). This would not only increase the 'tastefulness' of our denotational model but it would also bring us closer to O'Hearn and Tennent's parametric functor model [22]. Hence it may finally turn out that their model is also fully abstract for the second order subset of ALG.

1.1. Preliminaries

Sets and functions: Let A,B be sets. We write $f : A \to B$ (resp. $f : A \hookrightarrow B$) to express that f is a total (resp. partial) function from A to B. $(A \xrightarrow{t} B)$ stands for the set of all total functions from A to B and $\mathscr{P}_{fin}(A)$ for the set of all finite subsets of A. If $f,g : A \hookrightarrow B$, $C \subseteq A$, $a, a_1, \ldots, a_n \in A$ and $b_1, \ldots, b_n \in B$, then we write
- $dom(f)$ for the domain of f
- $f \mid C$ for the restriction of f to C
- $f \setminus a$ for the restriction of f to $(dom(f) \setminus \{a\})$
- $f =_C g$ for $f \mid C = g \mid C$
- $f[b_1, \ldots, b_n/a_1, \ldots, a_n]$ or just $f[\bar{b}/\bar{a}]$ for the function $f' : A \hookrightarrow B$ with

$$dom(f') = dom(f) \cup \{a_1, \ldots, a_n\}$$

$$f'a = \begin{cases} b_i & \text{if } a = a_i \\ fa & \text{if } a \in dom(f') \setminus \{a_1, \ldots, a_n\} \end{cases}$$

Complete partial orders: Let $(D, \sqsubseteq)$ be a partial order. A set $\Delta \subseteq D$ is *directed*, if every finite set $S \subseteq \Delta$ has an upper bound in Δ. D is called *directed complete* (or a *dcpo*), if every directed set $\Delta \subseteq D$ has a least upper bound (*lub*) in D. This least upper bound is denoted $\bigsqcup_D \Delta$ or just $\bigsqcup \Delta$. A function $f : D \to E$ between dcpo's D and E is *continuous*, if $f(\bigsqcup_D \Delta) = \bigsqcup_E f\Delta$ for every directed set $\Delta \subseteq D$. $(D \xrightarrow{c} E)$ denotes the set of all continuous functions from D to E.

1.2. Overview

Our paper is structured as follows: In Section 2 we define the syntax of our language ALG. In Section 3 we present a structural operational semantics. This semantics is interesting in its own because of the snap back effect and the parallel conditional. In Section 4 we introduce the general framework for our denotational semantics; it is essentially a reformulation of the definitions in [13]. Section 5 contains the particular denotational model which we need for obtaining full abstraction, and in Section 6 we prove that the model is computationally adequate. In Section 7 we illustrate how to use the denotational semantics for proving particular observational congruences, and in Section 8 we take a closer look at (the semantic domains for) types of order $\leqslant 2$. The full abstraction proof itself is contained in Section 9. Section 10 discusses some variants of the language ALG and Section 11 contains some open questions.

2. Syntax of the language Alg

In the spirit of [7, 30] we define our Algol-like language Alg as a subset of a simply typed λ-calculus. Its *types* τ are given by the grammar

$$\begin{aligned}\tau &::= loc \mid \sigma\\ \sigma &::= \theta \mid (\tau \rightarrow \sigma)\\ \theta &::= iexp \mid cmd\end{aligned}$$

loc stands for 'location', *iexp* for 'integer expression' and *cmd* for 'command'. We let *Type* denote the set of all types. The types $\sigma\,(\neq loc)$ are called *procedure types*. As usual, '$\rightarrow$' associates to the right, hence every procedure type can be written as $\sigma = \tau_1 \rightarrow \cdots \rightarrow \tau_k \rightarrow \theta$ with some $k \geqslant 0$. We use $\tau^k \rightarrow \sigma$ as an abbreviation for $\underbrace{\tau \rightarrow \cdots \rightarrow \tau}_{k} \rightarrow \sigma$ $(k \geqslant 0)$. The *order* $ord(\tau)$ of a type τ is defined by

$$\begin{aligned}ord(loc) &= 0\\ ord(\theta) &= 1\\ ord(\tau \rightarrow \sigma) &= max(ord(\tau)+1, ord(\sigma))\end{aligned}$$

It may come as a surprise that we assign the order 1 to the ground (!) types *iexp* and *cmd*. This does make sense, because – semantically – elements of type *iexp* and *cmd* will be *functions* which have the current store as an implicit parameter; in particular, elements of type *iexp* will be *thunks* in terms of the Algol jargon.[4] From an operational point of view this means that parameters of type *iexp* (and *cmd*) are *called by name*, i.e. they are handled by β-reduction. Thus we follow the view that call-by-name should be the main parameter passing mechanism for an Algol-like language. In addition we have parameters of type *loc*; they have been added as a mere convenience because we need identifiers of type *loc* as local variables. Intuitively, they may be considered as *reference parameters*, but technically they can also be handled by β-reduction because the only terms of type *loc* are location constants and variables.

As usual, we assume that there is an infinite set Id^τ of *identifiers* $x^\tau, y^\tau, z^\tau, \ldots$ for every type τ; the type superscripts will be omitted when the type is clear from the context. Identifiers of procedure type σ are called *procedure identifiers*, those of type *loc* are called *location identifiers* or *variables*. This means that we use the word 'variable' in the sense of imperative programming languages and not in the (more general) sense of the λ-calculus. We will preferably use $y, z, \ldots$ as procedure identifiers and $x, x', \ldots$ as variables (or as generics for arbitrary identifiers).

[4] Put another way, *cmd* and *iexp* are the types of parameterless procedures and function procedures, corresponding to the ML-types $(unit \rightarrow unit)$ and $(unit \rightarrow int)$, and thus it is reasonable to assign the order 1 to them.

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

$l : loc$ for every $l \in Loc$		(location constants)
$n : iexp$ for every $n \in \mathbb{Z}$		(integer constants)
$succ, pred : iexp \to iexp$		(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)
$seq_\theta : cmd \to \theta \to \theta$		(sequencing)
$new_\theta : (loc \to \theta) \to \theta$		(*new*-operator)
$\mathrm{Y}_\sigma : (\sigma \to \sigma) \to \sigma$		(fixed point operator)
$pcond : iexp \to iexp \to iexp \to iexp$		(parallel conditional with zero test)

Terms $M, N, P, \ldots$ of ALG are just the well-typed λ-terms over the ALG-constants with the restriction that the body of a λ-abstraction must not be of type *loc*, in other words: The sets ALG^τ of ALG -terms of type τ are inductively defined by

$c \in \mathrm{ALG}^\tau$	if c is a constant of type τ	(constant)
$x^\tau \in \mathrm{ALG}^\tau$		(identifier)
$M \in \mathrm{ALG}^{\tau\to\sigma} \wedge N \in \mathrm{ALG}^\tau \Rightarrow (MN) \in \mathrm{ALG}^\sigma$		(application)
$M \in \mathrm{ALG}^\sigma \Rightarrow (\lambda x^\tau . M) \in \mathrm{ALG}^{\tau\to\sigma}$		(λ-abstraction)

As usual, application associates to the left and the scope of a λ-abstraction extends as far as possible to the right.

We let *locns*(M) stand for the set of locations which occur (as constants) in M and *free*(M) for the set of identifiers which occur free in M. M is *closed* if *free*(M) $= \emptyset$. $M[x := N]$ is the term which is obtained from M by substituting N for each free occurrence of x (with the usual renaming of bound identifiers in order to avoid name clashes), and $M[x_1, \ldots, x_k := N_1, \ldots, N_k]$ or simply $M[\bar{x} := \bar{N}]$ is the term which is obtained by a *simultaneous* substitution. As further notation we use

$$\mathrm{ALG}_L^\tau = \{M \in \mathrm{ALG}^\tau \mid locns(M) \subseteq L\}$$
$$c\text{-}\mathrm{ALG}^\tau = \{M \in \mathrm{ALG}^\tau \mid free(M) = \emptyset\}$$
$$c\text{-}\mathrm{ALG}_L^\tau = \{M \in \mathrm{ALG}^\tau \mid free(M) = \emptyset \wedge locns(M) \subseteq L\}$$

where L ranges over finite subsets of *Loc*.

Finally, we define a *program* to be a term $P \in c\text{-}\mathrm{ALG}_\emptyset^{iexp}$. Note that location constants (which may be thought of as explicit storage addresses) must not occur in programs at all. Terms with location constants will be useful for defining the operational semantics; besides that they will play a technical role in the full abstraction proof.

We conclude this section by introducing some syntactic sugar. First, we generalize conditional, sequencing and *new*-operators to arbitrary procedure types: If $\sigma = \tau_1 \to$

$\dots \to \tau_k \to \theta$ $(k \geqslant 1)$, then we let

$$\begin{aligned}
cond_\sigma &=_{def} \lambda\, y^{iexp}, z_1^\sigma, z_2^\sigma, x_1^{\tau_1}, \dots, x_k^{\tau_k}.\, cond_\theta\, y\, (z_1 x_1 \dots x_k)\, (z_2 x_1 \dots x_k)\\
seq_\sigma &=_{def} \lambda\, y^{cmd}, z^\sigma, x_1^{\tau_1}, \dots, x_k^{\tau_k}.\, seq_\theta\, y\, (z\, x_1 \dots x_k)\\
new_\sigma &=_{def} \lambda\, y^{loc \to \sigma}, x_1^{\tau_1}, \dots, x_k^{\tau_k}.\, new_\theta\, (\lambda\, x^{loc}.\, y\, x\, x_1 \dots x_k)
\end{aligned}$$

Besides that, we introduce some notation which looks more familiar for imperative languages, namely

$$\begin{aligned}
!\,M &=_{def} cont\, M\\
M := N &=_{def} asgn\, M\, N\\
M; N &=_{def} seq_\sigma M\, N\\
\textbf{if } M \textbf{ then } N \textbf{ else } P &=_{def} cond_\sigma M\, N\, P\\
\textbf{new}\, x \textbf{ in } M \textbf{ end} &=_{def} new_\sigma(\lambda\, x^{loc}.M)\\
\textbf{proc } y^\sigma : M \textbf{ in } N \textbf{ end} &=_{def} (\lambda\, y^\sigma.N)(\mathrm{Y}_\sigma\, (\lambda\, y^\sigma.M))\\
\textbf{if } M \textbf{ then } N &=_{def} \textbf{if } M \textbf{ then } N \textbf{ else } skip\\
\textbf{new}\, x_1, \dots, x_n \textbf{ in } M \textbf{ end} &=_{def} \textbf{new}\, x_1 \textbf{ in } \dots \textbf{new}\, x_n \textbf{ in } M \textbf{ end} \dots \textbf{ end}
\end{aligned}$$

In each case we insist that the term on the right hand side be already well-typed. Note that some of these constructs are defined more generally than in traditional imperative languages because σ ranges over arbitrary procedure types. Finally we let Ω_σ or just Ω stand for a diverging term of type σ, say $\Omega_\sigma =_{def} \mathrm{Y}_\sigma(\lambda\, y^\sigma.\, y)$.

3. Operational semantics

In this section we define a structural operational semantics [28] for our language ALG , i.e. we define a transition relation '$\to$' on a set of (machine) configurations. Our definition of configurations is somewhat unusual, because the language ALG is not single threaded.

In a *single threaded* language, a configuration can be defined to be a pair (M, ms) where – intuitively – ms is the current (marked) store and M is the term to be evaluated next. For the language ALG, single threadedness fails because of the snap back effect and the parallel conditional. The snap back effect forces us to keep book of *earlier* stores into which the computation might snap back after the evaluation of an integer expression. The parallel conditional forces us to make copies of the *current* store, because we do not want to allow any interaction between computations which run in parallel, hence we insist that each argument of the parallel conditional works on its own 'private' copy of the store. In order to handle both features together we define the set of configurations K by

$$\begin{aligned} K ::= {} & (M, ms) \mid \\ & succ\, K \mid pred\, K \mid \\ & (asgn\, l\, K, ms) \mid (cond_\theta KMN, ms) \mid \\ & seq_\theta KM \mid dealloc_\theta\, l\, K \mid \\ & pcond K_1 K_2 K_3 \end{aligned}$$

where M and N are *closed* terms.

The rules for deriving transition steps between configurations are presented in Table 1. An auxiliary transition relation '$\rightarrowtriangle$' between closed ALG -terms is defined in Table 2. We explicitly distinguish between '$\longrightarrow$' and '$\rightarrowtriangle$' in order to emphasize that the operational semantics of an ALGOL-like language is naturally separated into two layers [8, 9, 30, 38]: '$\rightarrowtriangle$' describes the purely functional layer, in which the only transition steps are β-reduction and recursion unfolding. '$\longrightarrow$' describes the imperative layer, where transition steps can depend on the store and/or change the store. The only connection between the two layers is given by the (interaction)-rule of Table 2.

Table 1
Rules for '$\longrightarrow$'

(succ-init)	$(succ\, M, ms) \longrightarrow succ\,(M, ms)$
(succ-exec)	$succ\,(n, ms) \longrightarrow (n+1, ms)$
(pred-init)	$(pred\, M, ms) \longrightarrow pred\,(M, ms)$
(pred-exec)	$pred\,(n, ms) \longrightarrow (n-1, ms)$
(cont)	$(cont\, l, ms) \longrightarrow (ms\, l, ms)$ if $l \in dom(ms)$
(asgn-init)	$(asgn\, l\, M, ms) \longrightarrow (asgn\, l\,(M, ms), ms)$
(asgn-exec)	$(asgn\, l\,(n, ms'), ms) \longrightarrow (skip, ms[n/l])$ if $l \in dom(ms)$
(cond-init)	$(cond_\theta MNP, ms) \longrightarrow (cond_\theta(M, ms)NP, ms)$
(cond-left)	$(cond_\theta(0, ms')NP, ms) \longrightarrow (N, ms)$
(cond-right)	$(cond_\theta(n, ms')NP, ms) \longrightarrow (P, ms)$ if $n \neq 0$
(seq-init)	$(seq_\theta MN, ms) \longrightarrow seq_\theta(M, ms)N$
(seq-finish)	$seq_\theta(skip, ms)N \longrightarrow (N, ms)$
(new-init)	$(new_\theta M, ms) \longrightarrow dealloc_\theta\, l\,(Ml, ms[0/l])$ if $l = next(dom(ms))$
(new-finish)	$dealloc_\theta\, l\,(c, ms) \longrightarrow (c, ms \setminus l)$ if $l \in dom(ms)$
(pcond-init)	$(pcond MNP, ms) \longrightarrow pcond\,(M, ms)\,(N, ms)\,(P, ms)$
(pcond-left)	$pcond\,(0, ms)\, K_1 K_2 \longrightarrow K_1$
(pcond-right)	$pcond\,(n, ms)\, K_1 K_2 \longrightarrow K_2$ if $n \neq 0$
(pcond-par)	$pcond\, K\,(n, ms)\,(n, ms') \longrightarrow (n, ms)$
(context)	$\dfrac{K_i \longrightarrow K_i' \text{ for } i = 1, \ldots, n}{E[K_1, \ldots, K_n] \longrightarrow E[K'_1, \ldots, K_n']}$

Table 2
Rules for '$\rightarrowtriangle$' and interaction

(β-reduction)	$(\lambda x^\tau.\, M)N \rightarrowtriangle M[x := N]$
(recursion)	$\mathrm{Y}_\sigma M \rightarrowtriangle M(\mathrm{Y}_\sigma M)$
(application)	$\dfrac{M \rightarrowtriangle M'}{MN \rightarrowtriangle M'N}$
(interaction)	$\dfrac{M \rightarrowtriangle M'}{(M, ms) \longrightarrow (M', ms)}$

In the (context)-rule we make use of the so-called evaluation contexts [38]. In our setting, an *evaluation context* E is a particular 'configuration with at least one hole' defined by

$$\begin{aligned} E ::= {} & \mathit{succ}[\ ] \mid \mathit{pred}[\ ] \mid \\ & (\mathit{asgn}\, l\, [\ ], \mathit{ms}) \mid (\mathit{cond}_\theta[\ ]MN, \mathit{ms}) \mid \\ & \mathit{seq}_\theta[\ ]M \mid \mathit{dealloc}_\theta\, l\, [\ ] \mid \\ & \mathit{pcond}[\ ][\ ][\ ] \mid \\ & \mathit{pcond}[\ ][\ ](n, \mathit{ms}) \mid \mathit{pcond}[\ ](n, \mathit{ms})[\ ] \mid \\ & \mathit{pcond}[\ ](n, \mathit{ms})(n', \mathit{ms}') \quad \text{with } n \neq n' \end{aligned}$$

As usual, [ ] specifies a hole, and $E[K_1, \ldots, K_n]$ denotes the configuration which is obtained by filling $K_1, \ldots, K_n$ into the n holes of E. The intuition (expressed by the (context)-rule) is that an evaluation context E enforces the parallel evaluation of the configurations $K_1, \ldots, K_n$ which are placed in its holes.

With the aid of the transition relation '$\to$' we define the *observable behavior* of a program P to be the set

$$\mathit{beh}(P) = \{ n \mid (P, \mathit{ms}_{\mathit{init}}) \xrightarrow{*} (n, \mathit{ms}_{\mathit{init}}) \}$$

where $\mathit{ms}_{\mathit{init}}$ is the (unique) marked store with $\mathit{dom}(\mathit{ms}_{\mathit{init}}) = \emptyset$ and '$\xrightarrow{*}$' denotes the reflexive transitive closure of '$\to$'. We will see below, that $\mathit{beh}(P)$ contains at most one element and that $\mathit{beh}(P) = \emptyset$ if and only if the computation for $(P, \mathit{ms}_{\mathit{init}})$ diverges. But first we give a small example to illustrate this somewhat unusual operational semantics and in particular to illustrate the snap back effect.

Example 3.1. Consider the program

$$\begin{aligned} P &\equiv \textbf{new}\, x \textbf{ in if } x := 1;\ !x \textbf{ then } 1 \textbf{ else } !x \textbf{ end} \\ &\equiv \mathit{new}_{\mathit{iexp}}(\lambda x^{\mathit{loc}}.\, \mathit{cond}_{\mathit{iexp}}(\mathit{seq}_{\mathit{iexp}}(\mathit{asgn}\, x\, 1)(\mathit{cont}\, x))\, 1\, (\mathit{cont}\, x)) \end{aligned}$$

We show that $(P, \mathit{ms}_{\mathit{init}}) \xrightarrow{*} (0, \mathit{ms}_{\mathit{init}})$. Let $l = \mathit{next}(\emptyset)$ and let $[l : n]$ denote the marked store ms with $\mathit{dom}(\mathit{ms}) = \{l\}$ and $\mathit{ms}\, l = n$. Then

$$\begin{aligned} (P, \mathit{ms}_{\mathit{init}}) &\to \mathit{dealloc}\, l\, ((\lambda x.\, \mathit{cond}\, (\mathit{seq}\, (\mathit{asgn}\, x\, 1)\, (\mathit{cont}\, x))\, 1\, (\mathit{cont}\, x))\, l, [l : 0]) \\ &\to \mathit{dealloc}\, l\, (\mathit{cond}\, (\mathit{seq}\, (\mathit{asgn}\, l\, 1)\, (\mathit{cont}\, l))\, 1\, (\mathit{cont}\, l), [l : 0]) \\ &\to \mathit{dealloc}\, l\, (\mathit{cond}\, (\mathit{seq}\, (\mathit{asgn}\, l\, 1)\, (\mathit{cont}\, l), [l : 0])\, 1\, (\mathit{cont}\, l), [l : 0]) \\ &\xrightarrow{*} \mathit{dealloc}\, l\, (\mathit{cond}\, (1, [l : 1])\, 1\, (\mathit{cont}\, l), [l : 0]) \\ &\to \mathit{dealloc}\, l\, (\mathit{cont}\, l, [l : 0]) \\ &\to \mathit{dealloc}\, l\, (0, [l : 0]) \\ &\to (0, \mathit{ms}_{\mathit{init}}) \end{aligned}$$

where '$\xrightarrow{*}$' follows by the (context)-rule from

$$\begin{aligned}(seq\,(asgn\,l\,1)\,(cont\,l), [l:0]) &\rightarrow seq\,(asgn\,l\,1, [l:0])\,(cont\,l)\\ &\rightarrow seq\,(asgn\,l\,(1,[l:0]),[l:0])\,(cont\,l)\\ &\rightarrow seq\,(skip,[l:1])\,(cont\,l)\\ &\rightarrow (cont\,l,[l:1])\\ &\rightarrow (1,[l:1])\end{aligned}$$

Note that the marked store $[l:0]$ is duplicated in the third step of the computation and that the first copy of $[l:0]$ is changed to $[l:1]$ by the evaluation of $seq\,(asgn\,l\,1)\,(cont\,l)$. But then the computation *snaps back* to (the second copy of) $[l:0]$, and thus the evaluation of $cont\,l$ finally delivers 0.

We conclude this section by proving some useful properties of our operational semantics, in particular we want to show that "computations do not get stuck". To this end we will prove that all configurations which occur during the evaluation of a program have a certain reasonable shape: For $\theta \in \{iexp, cmd\}$ and $L \in \mathscr{P}_{fin}(Loc)$ we define the sets $Conf_L^\theta$ inductively by

- $M \in c\text{-}\mathrm{ALG}_L^\theta \wedge dom(ms) = L \Rightarrow (M, ms) \in Conf_L^\theta$
- $K \in Conf_L^{iexp} \Rightarrow succ\,K, pred\,K \in Conf_L^{iexp}$
- $K \in Conf_L^{iexp} \wedge l \in L \wedge dom(ms) = L \Rightarrow (asgn\,l\,K, ms) \in Conf_L^{cmd}$
- $K \in Conf_L^{iexp} \wedge M, N \in c\text{-}\mathrm{ALG}_L^\theta \wedge dom(ms) = L$
  $\Rightarrow (cond_\theta KMN, ms) \in Conf_L^\theta$
- $K \in Conf_L^{cmd} \wedge M \in c\text{-}\mathrm{ALG}_L^\theta \Rightarrow seq_\theta KM \in Conf_L^\theta$
- $K \in Conf_L^\theta \wedge l \in L \Rightarrow dealloc_\theta\,l\,K \in Conf_{L\setminus\{l\}}^\theta$
- $K_1, K_2, K_3 \in Conf_L^{iexp} \Rightarrow pcond\,K_1K_2K_3 \in Conf_L^{iexp}$

We say that a configuration is *consistent* if it is contained in one of the sets $Conf_L^\theta$. Intuitively, a consistent configuration is 'well-typed' and does not contain any *dangling references* [15, 16]. The latter means that every location which occurs in a consistent configuration is 'active' in the sense that it is contained in the domain of the corresponding marked store and hence a computation will never get stuck because of the restriction '$l \in dom(ms)$' in the rules (cont) or (asgn-exec). This is just a particular instance of

Theorem 3.2 (Properties of the operational semantics). (i) *The transition relations '$\twoheadrightarrow$' and '$\rightarrow$' are partial functions.*

(ii) *If $M \in c\text{-}\mathrm{ALG}_L^\tau$ and $M \twoheadrightarrow M'$ then $M' \in c\text{-}\mathrm{ALG}_L^\tau$. If $K \in Conf_L^\theta$ and $K \rightarrow K'$ then $K' \in Conf_L^\theta$.*

(iii) *$K \in Conf_L^{iexp}$ is in normal form iff it is of the form (n, ms). $K \in Conf_L^{cmd}$ is in normal form iff it is of the form $(skip, ms)$.*

(i) means that all computations are deterministic, (ii) means that transition steps preserve types and consistency, and together with (iii) this implies that each computation which starts with a consistent configuration $K \in Conf^{\theta}_{L}$ either diverges or terminates with a 'proper result' $(c, ms) \in Conf^{\theta}_{L}$. As $(P, ms_{init}) \in Conf^{iexp}_{\emptyset}$ for every program P, this implies in particular that the evaluation of a program can never get stuck; it either diverges or terminates with a unique result of the form (n, ms_{init}). Note that the final configuration (n, ms_{init}) does not contain any garbage, because our operational semantics explicitly follows the stack discipline: Every location which is allocated upon block entry by rule (new-init) is eventually deallocated upon block exit by rule (new-finish), and thus – in contrast to the situation in a call-by-value language [15, 16] – a location never survives the block in which it has been allocated.

Proof. (i) is omitted. It is a routine argumentation about the applicability of rules. (ii) The first part is obvious, because a transition step $M \multimap\!\!\!\!\rhd M'$ cannot create any new location constants. The second part is proved by induction on the derivation of $K \to K'$. We consider a few cases in which locations play a role; the proofs for the remaining cases are absolutely straightforward.

Case 1: $K \equiv (asgn\, l\, (n, ms'), ms) \to K' \equiv (skip, ms[n/l])$ by rule (asgn-exec)
Then $l \in dom(ms)$ and we obtain

$$\begin{aligned} K \in Conf^{cmd}_{L} &\Rightarrow dom(ms) = L \\ &\Rightarrow dom(ms[n/l]) = L \\ &\Rightarrow K' \in Conf^{cmd}_{L} \end{aligned}$$

Case 2: $K \equiv (new_{\theta} M, ms) \to K' \equiv dealloc_{\theta}\, l\, (Ml, ms[0/l])$ by rule (new-init)
Here we have $l = next(dom(ms))$ and thus we obtain

$$\begin{aligned} K \in Conf^{\theta}_{L} &\Rightarrow M \in c\text{-}\mathrm{ALG}^{loc \to \theta}_{L} \wedge dom(ms) = L \quad (\text{hence } l \notin L) \\ &\Rightarrow Ml \in c\text{-}\mathrm{ALG}^{\theta}_{L \cup \{l\}} \wedge dom(ms[0/l]) = L \cup \{l\} \\ &\Rightarrow (Ml, ms[0/l]) \in Conf^{\theta}_{L \cup \{l\}} \\ &\Rightarrow K' \in Conf^{\theta}_{(L \cup \{l\}) \setminus \{l\}} = Conf^{\theta}_{L} \end{aligned}$$

Case 3: $K \equiv dealloc_{\theta}\, l\, (c, ms) \to K' \equiv (c, ms \setminus l)$ by rule (new-finish)
Again $l \in dom(ms)$ and hence

$$\begin{aligned} K \in Conf^{\theta}_{L} &\Rightarrow dom(ms) = L \cup \{l\} \wedge l \notin L \\ &\Rightarrow dom(ms \setminus l) = L \\ &\Rightarrow K' \in Conf^{\theta}_{L} \end{aligned}$$

(iii) The 'if'-parts are obvious from the rules; 'only if' is proved by induction on the structure of K: Assume that $K \in Conf^{\theta}_{L}$ is *not* of the form (n, ms) or $(skip, ms)$.

Case 1: $K \equiv (M, ms)$

From the definition of $Conf_L^\theta$ it follows that $M \in c\text{-}\mathrm{ALG}_L^\theta$ and $dom(ms) = L$. By assumption, M is not a constant, and as θ is a ground type, M cannot be a λ-abstraction as well, hence it must be an application which can be written as $M \equiv M_0 M_1 \ldots M_k$ ($k \geqslant 1$) where M_0 is *not* an application. If M_0 is a λ-abstraction or a fixed point operator, then $M \mathrel{-\!\!\triangleright} M'$ for some $M' \in c\text{-}\mathrm{ALG}_L^\theta$, hence $K \rightarrow (M', ms)$. Otherwise, M_0 must be a constant $c \not\equiv \mathrm{Y}_\sigma$, and then one of the (init)-rules applies in each case. For example, if $M_0 \equiv asgn$, then K must be of the form $(asgn\, l\, M_2, ms)$ with $l \in L = dom(ms)$, hence $K \rightarrow (asgn\, l\, (M_2, ms), ms)$ by rule (asgn-init). The argumentation for the remaining constants is similar.

Case 2: $K \equiv E[K']$, where E does not start with '*pcond*'
If $K' \rightarrow K''$ then $K \rightarrow E[K'']$ by the (context)-rule. If K' is in normal form, then one of the other rules of Table 1 can be applied. For example, if $E \equiv (asgn\, l\, [\ ], ms)$, then $K' \in Conf_L^{iexp}$ and $l \in L = dom(ms)$, hence $K' \equiv (n, ms')$ by induction hypothesis and this implies $K \equiv (asgn\, l\, (n, ms'), ms) \rightarrow (skip, ms[n/l])$. The argumentation for the remaining evaluation contexts is similar.

Case 3: $K \equiv pcond K_1 K_2 K_3$
Here either the (context)-rule or one of the (pcond)-rules can be applied, depending on which of the configurations K_1, K_2, K_3 are in normal form. □

4. A cartesian closed category

In this section we define the general framework for our denotational semantics. The intuition is, that every function in the denotational model should only have access to some fixed finite set of locations. Hence we would like to identify – for every type τ and every $L \in \mathscr{P}_{fin}(Loc)$ – a dcpo $[\![\tau]\!]_L$ of 'elements of type τ which only have access to L' and then define $[\![\tau]\!]$ as the union of the dcpo's $[\![\tau]\!]_L$. From this intuition it should be clear that $[\![\tau]\!]_L \subseteq [\![\tau]\!]_{L'}$ whenever $L \subseteq L'$ and that $[\![\tau]\!]$ itself need not be a dcpo (because a sequnece of functions which have access to more and more locations may have a limit which has access to infinitely many locations). This is the motivation for

Definition 4.1. Let $(W, \leqslant)$ be a directed set (of 'worlds' w).

(1) 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 $v, w \in W$
- $v \leqslant w \Rightarrow D_v \subseteq D_w$
- 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)

D is called *pointed*, if it has a least element which is contained in all D_w.

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

Note that every finite subset S of a W-lcpo D is entirely contained in one of the dcpo's D_w, because W is directed and $v \leqslant w$ implies $D_v \subseteq D_w$. This implies in turn that every locally continuous function $f : D \rightarrow E$ is monotone on the whole of D.

For every directed set W, we let W-**LCPO** denote the category whose objects are W-lcpos and whose morphisms are locally continuous functions. It can be easily checked that this is indeed a category.

Theorem 4.2 (W-**LCPO** is a ccc). *The category W-***LCPO** *is cartesian closed. The terminal object T, the product $D \times E$ and the exponent $(D \to E)$ of two W-lcpos D and E are defined by*

(i) $T_w = \{\emptyset\}$

$T = \{\emptyset\}$

(ii) $(D \times E)_w = D_w \times E_w$

$D \times E = \bigcup_{w \in W} (D \times E)_w$ *with the componentwise order on pairs*

(iii) $(D \to E)_w = \{f : D \to E \mid \forall v \geqslant w . (f \mid D_v) \in (D_v \xrightarrow{c} E_v)\}$

$(D \to E) = \bigcup_{w \in W} (D \to E)_w$ *with the pointwise order on functions*

Projection morphisms and pairing, evaluation morphisms and currying and the unique morphisms to the terminal object are defined as usual.

The experienced reader certainly realizes the similarity with a functor category, in particular (iii) looks like functor exponentiation [36]. Indeed, a W-lcpo D can be considered as a functor from the category W to the category **DCPO** of dcpo's and continuous functions, which maps every morphism $f : v \to w$ in W to the inclusion map $i : D_v \to D_w$ (which is continuous, because $\bigsqcup_{D_v} \Delta = \bigsqcup_D \Delta = \bigsqcup_{D_w} \Delta$ for every directed set $\Delta \subseteq D_v$). The locally continuous functions between two W-lcpos then correspond exactly to the natural transformations between the functors, and exponentiation in W-**LCPO** corresponds to functor exponentiation. Hence W-**LCPO** can be identified with a full subcategory of the functor category ($W \Rightarrow$ **DCPO**) which has the same terminal object, products and exponents as ($W \Rightarrow$ **DCPO**) itself.

We omit the proof of Theorem 4.2, because the category W-**LCPO** is not sufficient for our purposes. We are aiming for a denotational model in which the function types $[\![\tau \to \sigma]\!]$ contain only those locally continuous functions which preserve certain (logical) relations. To this end we must add 'relation structure' to the W-lcpos and then refine the definition of the exponent $(D \to E)$.

Definition 4.3. 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 $v, w \in W$

$$m \neq n \Rightarrow \Sigma_m^v \cap \Sigma_n^w = \emptyset, \qquad v \leqslant w \Rightarrow \Sigma_n^v \supseteq \Sigma_n^w$$

An element $r \in \Sigma_n$ is called a *relation symbol* of *arity n*. We use the abbreviations

$$\Sigma_n =_{def} \bigcup_{w \in W} \Sigma_n^w, \qquad \Sigma^w =_{def} \bigcup_{n \in \mathbb{N}} \Sigma_n^w, \qquad \Sigma =_{def} \bigcup_{n \in \mathbb{N}} \Sigma_n$$

As we will make extensive use of 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 known from the context. A term $T(\vec{d},\vec{e},\ldots)$ containing vectors $\vec{d},\vec{e},\ldots$ of the same length n stands for $(T(d_1,e_1,\ldots),\ldots,T(d_n,e_n,\ldots))$. This 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 $\{T(\vec{d},\vec{e},\ldots) \mid \vec{d} \in R,\, \vec{e} \in S,\ldots\}$. Finally, $\delta^n D$ or just δD denotes the diagonal $\{(d,\ldots,d) \mid d \in D\} \subseteq D^n$. A few typical examples for this notation are

$f\vec{d}$	for	$(fd_1,\ldots,fd_n)$
$\vec{f}\vec{d}$	for	$(f_1d_1,\ldots,f_nd_n)$
fR	for	$\{(fd_1,\ldots,fd_n) \mid (d_1,\ldots,d_n) \in R\}$
$\vec{f}R$	for	$\{(f_1d_1,\ldots,f_nd_n) \mid (d_1,\ldots,d_n) \in R\}$
$\vec{f}(\delta D)$	for	$\{(f_1d,\ldots,f_nd) \mid d \in D\}$
RS	for	$\{(f_1d_1,\ldots,f_nd_n) \mid (f_1,\ldots,f_n) \in R,\, (d_1,\ldots,d_n) \in S\}$

Definition 4.4. Let Σ be a W-sorted signature.

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

– $r \in \Sigma^w \Rightarrow \delta^n D_w \subseteq \mathscr{I}(r)$

– $\mathscr{I}(r) \cap D_w^n$ is closed under least upper bounds of directed sets

$(D,\mathscr{I})$ is called *pointed* if the underlying W-lcpo D is pointed.

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

Note that the relations $\mathscr{I}(r)$ are themselves pointed whenever $(D,\mathscr{I})$ is pointed, because $r \in \Sigma_n^w$ implies $(\bot,\ldots,\bot) \in \delta^n D_w \subseteq \mathscr{I}(r)$.

For every directed set W and W-sorted signature Σ, we let W-Σ-**LCPO** denote the category whose objects are W-Σ-lcpos and whose morphisms are locally continuous Σ-homomorphisms. Again, it can be easily checked that this is a category.

Theorem 4.5 (W-Σ-**LCPO** is a ccc). *The category W-Σ-**LCPO** is cartesian closed. The terminal object T, the product $D \times E$ and the exponent $(D \to E)$ of two W-Σ-lcpos D and E are defined by*

(i) $T_w = \{\emptyset\}, \quad T = \{\emptyset\}, \quad \mathscr{I}^T(r) = \{\emptyset\}^n$ *if* $r \in \Sigma_n$

(ii) $(D \times E)_w = D_w \times E_w$

$D \times E = \bigcup_{w \in W} (D \times E)_w$ *with the componentwise order on pairs*

$\mathscr{I}^{D\times E}(r) = (\mathscr{I}^D(r),\mathscr{I}^E(r)) (= \{((d_1,e_1),\ldots,(d_n,e_n)) \mid \vec{d} \in \mathscr{I}^D(r), \vec{e} \in \mathscr{I}^E(r)\})$

(iii) $(D \to E)_w = \{f : D \to E \mid \forall v \geqslant w.\, (f \mid D_v) \in (D_v \xrightarrow{c} E_v)$
$\qquad \wedge\ \forall r \in \Sigma^w.\, f(\mathscr{I}^D(r)) \subseteq \mathscr{I}^E(r)\}$

$(D \to E) = \bigcup_{w \in W} (D \to E)_w$ *with the pointwise order on functions*

$\mathscr{I}^{(D\to E)}(r) = \{\vec{f} \mid \vec{f}(\mathscr{I}^D(r)) \subseteq \mathscr{I}^E(r)\}$

Projection morphisms and pairing, evaluation morphisms and currying and the unique morphisms to the terminal object are defined as usual.

Proof. The proofs for the terminal object and the product are straightforward, hence we only consider the exponent. We must first show that $(D \to E)$ is a well-defined W-Σ-lcpo. Obviously, $(D \to E)$ is a partial order and $v \leqslant w$ implies $(D \to E)_v \subseteq (D \to E)_w$, because $\Sigma^v \supseteq \Sigma^w$ in this case. For the remaining steps it is sufficient to show that for all $w \in W$ and $r \in \Sigma_n$

(1) if $\Delta \subseteq (D \to E)_w$ is directed, then $f : D \to E$ with $fd = \bigsqcup \Delta\, d$ is a well-defined function in $(D \to E)_w$ (hence it is the lub of Δ in D);

(2) if $r \in \Sigma^w$ then $\delta^n (D \to E)_w \subseteq \mathscr{I}^{(D\to E)}(r)$;

(3) if $\Delta \subseteq \mathscr{I}^{(D\to E)}(r) \cap (D \to E)_w^n$ is directed, then $\bigsqcup \Delta \in \mathscr{I}^{(D\to E)}(r)$.

The proofs are straightforward and are left to the reader. It remains to be shown that $(D \to E)$ is indeed the exponent of D and E in the category W-Σ-lcpo. To this end we prove that

(4) the function

$$\begin{aligned} &eval : (D \to E) \times D \to E \\ &eval\, f\, d = f\, d \end{aligned}$$

is a locally continuous Σ-homomorphism, and

(5) if C is a W-Σ-lcpo and $f : C \times D \to E$ is a locally continuous Σ-homomorphism, then the function

$$\begin{aligned} &\Lambda f : C \to (D \to E) \\ &\Lambda f\, c\, d = f\,(c,d) \end{aligned}$$

is well-defined and is a locally continuous Σ-homomorphism.

Proof of (4): For every $w \in W$, $eval\,((D \to E)_w \times D_w) = (D \to E)_w D_w \subseteq E_w$, and the restriction of $eval$ to $(D \to E)_w \times D_w$ is continuous, because lub's are defined pointwise on $(D \to E)_w$ and because every $f \in (D \to E)_w$ is continuous on D_w. Moreover, $eval\,(\mathscr{I}^{(D\to E)\times D}(r)) = eval\,(\mathscr{I}^{(D\to E)}(r), \mathscr{I}^D(r)) = \mathscr{I}^{(D\to E)}(r)\,(\mathscr{I}^D(r)) \subseteq \mathscr{I}^E(r)$ for every $r \in \Sigma$.

Proof of (5): Let $f : C \times D \to E$ be a locally continuous Σ-homomorphism, let $w \in W$ and $c \in C_w$. If $v \geqslant w$, then $\Lambda f\, c\, D_v = f\,(\{c\} \times D_v) \subseteq f\,(C_v \times D_v) \subseteq E_v$ and $\Lambda f\, c \,|\, D_v$ is continuous because $f \,|\, C_v \times D_v$ is continuous. Moreover, $\Lambda f\, c\,(\mathscr{I}^D(r)) = f\,(c, \mathscr{I}^D(r)) \subseteq f\,(\delta\, C_w, \mathscr{I}^D(r)) \subseteq f\,(\mathscr{I}^C(r), \mathscr{I}^D(r)) \subseteq f\,(\mathscr{I}^{C\times D}(r)) \subseteq \mathscr{I}^E(r)$ for all $r \in \Sigma^w$. This shows that $\Lambda f\, c \in (D \to E)_w$. Hence Λf is a well-defined function which maps C_w to $(D \to E)_w$ for every $w \in W$. The restriction of Λf to each C_w is continuous, because f is continuous on $C_w \times \{d\}$ for every $d \in D$ and because lub's are defined pointwise on $(D \to E)_w$. Finally, $\Lambda f\,(\mathscr{I}^C(r))(\mathscr{I}^D(r)) = f\,(\mathscr{I}^C(r), \mathscr{I}^D(r)) = f\,(\mathscr{I}^{C\times D}(r)) \subseteq \mathscr{I}^E(r)$, i.e. $\Lambda f(\mathscr{I}^C(r)) \subseteq \mathscr{I}^{(D\to E)}(r)$ for every $r \in \Sigma$. $\square$

We finally remark that $(D \to E)$ is pointed whenever E is pointed: If $\bot_E$ is the least element of E, then $(\underline{\lambda} d \in D.\, \bot_E) \in (D \to E)_w$ for all $w \in W$ because $\bot_E \in E_w$ for all $w \in W$ and $(\bot_E, \ldots, \bot_E) \in \bigcap_{w \in W} \delta^n E_w \subseteq \mathscr{I}^E(r)$ for all $r \in \Sigma$. Together with the following theorem this guarantees that enough fixed point operators will be contained in our denotational model.

Theorem 4.6 (Least fixed point operators). *Let D be a pointed W-Σ-lcpo and let $f \in (D \to D)$. Then f has a least fixed point $\mu f \in D$, which can be characterized as usual by*

$$\mu f = \bigsqcup_{n \in \mathbb{N}} f^n \bot$$

Moreover, the least fixed point operator

$$\mu_D : (D \to D) \to D$$
$$\mu_D f = \mu f$$

is a locally continuous Σ-homomorphism.

Proof. Let $f \in (D \to D)_w$. As $(f \,|\, D_w) \in (D_w \xrightarrow{c} D_w)$, we know that $\bigsqcup_{n \in \mathbb{N}} f^n \bot$ exists and is the least fixed point of f in D_w. But then it is also its least fixed point in D, because – by monotonicity of f – $f^n \bot \sqsubseteq d$ for every other fixed point d. This shows already that μ_D maps $(D \to D)_w$ to D_w for every $w \in W$. Moreover, μ_D is continuous on every $(D \to D)_w$ because it is the pointwise lub of the functions $\underline{\lambda} f. f^n \bot$ ($n \in \mathbb{N}$), which are continuous on $(D \to D)_w$ by Theorem 4.5. Finally, let $r \in \Sigma_m$ and $\vec{f} \in \mathcal{I}^{(D \to D)}(r)$. Then $f_1, \ldots, f_m \in D_w$ for some $w \in W$, hence – by induction on n – $(f_1^n \bot, \ldots, f_m^n \bot) \in \mathcal{I}^D(r) \cap D_w^m$ for all $n \in \mathbb{N}$, and this implies $\mu_D \vec{f} \in \mathcal{I}^D(r)$. □

W-Σ-**LCPO** is the category in which we will define our denotational model (with an appropriate choice of W and Σ). It has a certain similarity with a category of 'parametric functors and natural transformations' [22, 23], and indeed we succeeded to prove a connection: Let $\mathscr{D}$ be the reflexive graph with vertex category **DCPO** as defined in [22]. Then – for every W-sorted signature Σ – we can define a reflexive graph $\mathscr{W}$ with vertex category W such that W-Σ-**LCPO** can be identified with a full subcategory of the parametric functor category $(\mathscr{W} \Rightarrow \mathscr{D})$ which has the same terminal object, products and exponents as $(\mathscr{W} \Rightarrow \mathscr{D})$ itself. We do not want to elaborate on this any further because it seems like a purely technical insight.

5. Denotational semantics

We will now use the techniques of Section 4 to define a denotational semantics for Alg . Of course we choose

$$(W, \leqslant) = (P_{fin}(Loc), \subseteq)$$

as the directed set of worlds, but the question remains how to define a W-sorted signature Σ which serves our purposes. The basic idea is the same as for our PCF-model in [32] [5]: In order to achieve full abstraction we try to keep the denotational model 'as small as possible' and to this end we try to make the relation signature 'as large as possible'. For the purely functional language PCF this was easy to achieve. We simply used *all* relations on the flat ground type of integers which are preserved by the meanings of the first order PCF-constants. This worked out, because all relations on a flat dcpo are automatically closed under lub's of directed sets (as required in Definition 4.4) and because the only higher order PCF-constants are fixed point operators. For the imperative language ALG the situation is more difficult, because the ground types $[\![iexp]\!]$ and $[\![cmd]\!]$ will certainly be *not* flat. Thus, in order to transfer the ideas of [32] to the ALG setting, we first introduce an additional semantic layer of flat dcpo's *below* the ground types $[\![iexp]\!]$ and $[\![cmd]\!]$, and on this new layer we define certain auxiliary functions, which are closely related to the intended meanings of the ALG -constants.

To begin with, we define the set *Stores* of *stores* s by

$$Stores = \bigcup_{L \in W} Stores_L$$

where

$$Stores_L = \{s : Loc \to \mathbb{Z} \mid \forall l \in Loc \setminus L.\, s\, l = 0\}$$

Note that a store s – in contrast to a marked store ms – is a *total* function which delivers 0 for all but finitely many locations. Working with total instead of partial functions is a technical trick which makes our denotational semantics somewhat simpler.

Now let $\Gamma = \{loc, int, sto\}$, 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 $\gamma \in \Gamma$ we define a dcpo D^γ by

$$\begin{array}{ll} D^{loc} = Loc & \text{(discrete dcpo)} \\ D^{int} = \mathbb{Z}_\perp,\ D^{sto} = Stores_\perp & \text{(flat dcpo's)} \end{array}$$

We write $\perp_\gamma$ for the bottom element of D^γ (if we want to be precise about γ) and id_γ for the identity on D^γ. The set *AUX* of *auxiliary functions* is then defined by

$$AUX = \{Const_n, Succ, Pred, Cont, Asgn, Cond_\gamma, Pcond \mid n \in \mathbb{Z}, \gamma \neq loc\}$$

where

- $Const_n : D^{sto} \to D^{int}$, $\quad Const_n\, s = \begin{cases} \perp & \text{if } s = \perp \\ n & \text{otherwise} \end{cases}$

[5] There are some purely technical differences between [32] and the new approach which we use here, e.g. we did not speak of a 'signature' in [32] and we used an extensional collapse for the model construction instead of defining a cartesian closed category. We ignore these technical issues here, because they have nothing to do with the difference between PCF and ALG but only with the particular presentation of the model.

- $Succ: \ D^{int} \to D^{int}, \qquad Succ\, d = \begin{cases} \bot & \text{if } d = \bot \\ d+1 & \text{otherwise} \end{cases}$

- $Pred: \ D^{int} \to D^{int}, \qquad Pred\, d = \begin{cases} \bot & \text{if } d = \bot \\ d-1 & \text{otherwise} \end{cases}$

- $Cont: \ D^{loc} \to D^{sto} \to D^{int}, \qquad Cont\, l\, s = \begin{cases} \bot & \text{if } s = \bot \\ s\, l & \text{otherwise} \end{cases}$

- $Asgn: \ D^{loc} \to D^{int} \to D^{sto} \to D^{sto}, \qquad Asgn\, l\, d\, s = \begin{cases} \bot & \text{if } d = \bot \text{ or } s = \bot \\ s[d/l] & \text{otherwise} \end{cases}$

- $Cond_\gamma: \ D^{int} \to D^\gamma \to D^\gamma \to D^\gamma, \qquad Cond_\gamma\, b\, d_1 d_2 = \begin{cases} \bot & \text{if } b = \bot \\ d_1 & \text{if } b = 0 \\ d_2 & \text{otherwise} \end{cases}$

- $Pcond: \ D^{int} \to D^{int} \to D^{int} \to D^{int}, \qquad Pcond\, b\, d_1 d_2 = \begin{cases} \bot & \text{if } b = \bot \text{ and } d_1 \neq d_2 \\ d_1 & \text{if } b = 0 \\ d_2 & \text{otherwise} \end{cases}$

With the aid of these auxiliary functions we can now define the signature Σ. The relation symbols of Σ are the so-called ground relations. A *ground relation* of *arity* n is simply a triple $R = (R^\gamma)_{\gamma \in \Gamma}$ such that $R^\gamma \subseteq (D^\gamma)^n$ for every $\gamma \in \Gamma$. We say that a function $f : D^{\gamma_1} \to \cdots \to D^{\gamma_k} \to D^\gamma$ *preserves* the ground relation R if $fR^{\gamma_1} \ldots R^{\gamma_k} \subseteq R^\gamma$. Finally, we let $\Sigma = (\Sigma_n^L)_{L \in W, n \in \mathbb{N}}$ where Σ_n^L is the set of all ground relations R of arity n such that

(a) every $f \in AUX$ preserves R

(b) $\delta^n(Loc \setminus L') \subseteq R^{loc}$ for some $L' \in W$ with $L \cap L' = \emptyset$ (i.e. R^{loc} contains a cofinite part of the diagonal $\delta^n Loc$ which includes $\delta^n L$)

(c) $(\bot_{sto}, \ldots, \bot_{sto}) \in R^{sto}$ (and hence $(\bot_{int}, \ldots, \bot_{int}) \in R^{int}$ by (a))

Note that Σ is indeed a W-sorted signature, because $L \subseteq L'$ implies $\Sigma_n^L \supseteq \Sigma_n^{L'}$. The motivation for choosing this particular signature Σ is as follows: Condition (a) will guarantee that $[\![n]\!], [\![succ]\!], [\![pred]\!], [\![cont]\!], [\![asgn]\!], [\![cond_\theta]\!]$ and $[\![pcond]\!]$ are Σ-homomorphisms. Together with (b) this will imply that every $R \in \Sigma^L$ is preserved by the functions $[\![cont]\!]\, l$ and $[\![asgn]\!]\, l$ not only for all $l \in L$ but also for all but finitely many $l \notin L$. The latter will play a role in the proof that the meanings of the *new*-operators are Σ-homomorphisms. Finally, (c) will be needed for handling the fixed point operators. Altogether these are the necessary conditions for Σ, if we want to define a denotational semantics for ALG in the category W-Σ-**LCPO**. This means that we have indeed chosen the 'largest possible' signature Σ for our purposes, and thus we can hope for a full abstraction proof along the lines of [32].

With the definition of W and Σ we have fixed the category in which we want to define our denotational model. The next step is to associate an object of this category with each type. For every type τ we define a W-Σ-lcpo $[\![\tau]\!] = (D^\tau, \mathscr{S}^\tau)$ by

– $D_L^{loc} = L \qquad D^{loc} = Loc \quad$ (as before) $\qquad \mathscr{S}^{loc}(R) = R^{loc}$

– $D_L^{sto \Rightarrow \gamma} = \{f : D^{sto} \to D^\gamma \mid f R^{sto} \subseteq R^\gamma$ for all $R \in \Sigma^L\}$

$D^{sto \Rightarrow \gamma} = \bigcup_{L \in W} D_L^{sto \Rightarrow \gamma}$ with the pointwise order on functions

$\mathscr{S}^{sto \Rightarrow \gamma}(R) = \{\vec{f} \in (D^{sto \Rightarrow \gamma})^n \mid \vec{f} R^{sto} \subseteq R^\gamma\} \quad$ if $R \in \Sigma_n$

– $[\![\tau \to \sigma]\!] = ([\![\tau]\!] \to [\![\sigma]\!])$ as defined in Theorem 4.5

It can be easily checked that the first two clauses indeed define W-Σ-lcpos, in particular $D_L^{sto \Rightarrow \gamma}$ is always closed under lub's of directed sets, because every R^γ is (trivially) closed under lub's of directed sets. Note also that $[\![\sigma]\!]$ is pointed for every procedure type σ (by a straightforward induction on σ). We write $\bot_\sigma$ for the bottom element of $[\![\sigma]\!]$ and id_σ for the identity on $[\![\sigma]\!]$.

The reader may have realized that the ground types $[\![iexp]\!] = [\![sto \Rightarrow int]\!]$ and $[\![cmd]\!] = [\![sto \Rightarrow sto]\!]$ have a certain similarity with our function types (Theorem 4.5) in that they consist of relation preserving functions. Hence the question may arise whether our model definition can be simplified by introducing *sto* and *int* as ground types and defining *iexp* and *cmd* as function types $(sto \to int)$ and $(sto \to sto)$. Unfortunately this is *not* possible. There is no way to define a W-Σ-lcpo $[\![sto]\!]$ such that $[\![cmd]\!]$ (as defined above) coincides with the exponent $([\![sto]\!] \to [\![sto]\!])$. O'Hearn and Tennent have occasionally used 'contra-exponentiation' instead of ordinary exponentiation to overcome this difficulty [35, 21], but for our purposes it doesnot seem worth to introduce such an extra concept; the above ad hoc definition of $[\![iexp]\!]$ and $[\![cmd]\!]$ is entirely sufficient.

We follow usual mathematical convention and use $[\![\tau]\!]$ not only as a notation for the W-Σ-lcpo $(D^\tau, \mathscr{S}^\tau)$ but also for the underlying 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 $\mathscr{S}^\tau(R)$. As immediate consequences of the definitions in Section 4 we then obtain the following 'reasoning principles' which will be frequently used throughout the rest of the paper.

(1) $[\![\tau \to \sigma]\!]_L \, [\![\tau]\!]_{L'} \subseteq [\![\sigma]\!]_{L'}$ whenever $L \subseteq L'$

(2) $f R^\tau \subseteq R^\sigma$ whenever $f \in [\![\tau \to \sigma]\!]_L$ and $R \in \Sigma^L$

(3) $R^{\tau \to \sigma} = \{\vec{f} \in [\![\tau \to \sigma]\!]^n \mid \vec{f} R^\tau \subseteq R^\sigma\}$ whenever $R \in \Sigma_n$

Reasoning principle (1) is equivalent to

(1′) $[\![\tau \to \sigma]\!]_L \, [\![\tau]\!]_{L'} \subseteq [\![\sigma]\!]_{L \cup L'}$ for all $L, L' \in W$

which can be rephrased in more intuitive terms as

"A procedure call fd can only have access to those locations to which either the procedure $f \in [\![\tau \to \sigma]\!]$ or the parameter $d \in [\![\tau]\!]$ has access".

For (2) we do not (yet) have such an intuitive formulation, because our current definition of the sets Σ^L is very technical, but we will come back to this in a moment.

(3) means that the family $(R^\tau)_{\tau \in Type}$ is a logical relation [18] for every $R \in \Sigma$. Logical relations are known to be a useful tool for reasoning about λ-terms [6, 20, 27, 32, 34].

We finally define the *support* of an element $d \in [\![\tau]\!]$ to be the set

$$supp(d) = \bigcap \{L \mid d \in [\![\tau]\!]_L\}$$

One may wonder whether $d \in [\![\tau]\!]_{supp(d)}$, i.e. whether there is a *smallest* set L with $d \in [\![\tau]\!]_L$. We have not examined this question, as it is irrelevant for our purposes.

As mentioned above, we are not yet satisfied with our current, rather technical characterization of the signature Σ. It is well suited for the full abstraction proof (especially for the proof of Theorem 9.3), but for other purposes – like proofs of particular observational congruences – a more concrete description of Σ would certainly be useful. Unfortunately, we have not found a (tasteful) concrete description of the full signature Σ, but instead we have identified the following 'sub-signature', which seems to be sufficient for proving all observational congruences (cf. Section 7 and Conjecture 11).

Definition 5.1. 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

$$R^{sto} = \{\bot\}^n \cup \{\vec{s} \in Stores^n \mid (\vec{s} \mid L) \in R_L \wedge \vec{s}(\delta^n(Loc \setminus L)) \subseteq \delta^n \mathbb{Z}\}$$

$$R^{int} = \delta^n D^{int}$$

$$R^{loc} = \{\vec{l} \in (D^{loc})^n \mid Cont\, \vec{l}\, R^{sto} \subseteq R^{int} \wedge Asgn\, \vec{l}\, R^{int}\, R^{sto} \subseteq R^{sto}\}$$

Note that every L-definable ground relation R is uniquely determined by R^{sto} or even by R_L and that – on the other hand – *every* set $S \subseteq (L \xrightarrow{t} Z)^n$ determines an L-definable ground relation R with $R_L = S$. We let DEF_n^L denote the set of all L-definable ground relations of arity n and $OUT_n^L = \bigcup_{L' \in W \wedge L \cap L' = \emptyset} DEF_n^{L'}$ the set of those which are definable outside L. Note that $OUT_n^L \supseteq OUT_n^{L'}$ whenever $L \subseteq L'$, hence $OUT = (OUT_n^L)_{L \in W, n \in \mathbb{N}}$ is itself a W-sorted signature.

Theorem 5.2 (A sub-signature of Σ). *$OUT_n^L \subseteq \Sigma_n^L$ for every $L \in W$ and $n \in \mathbb{N}$.*

Proof. Let $R \in DEF_n^{L'}$ for some $L' \in W$ with $L \cap L' = \emptyset$. Then $(\bot, \ldots, \bot) \in R^{sto}$, and it is easy to see that every function $f \in AUX$ preserves R. Hence it is sufficient to show that $\delta^n(Loc \setminus L') \subseteq R^{loc}$.

Let $l \in Loc \setminus L'$, let $\vec{s} \in R^{sto}$, $\vec{d} \in R^{int}$, $\vec{e} = Cont\, l\, \vec{s}$ and $\vec{t} = Asgn\, l\, \vec{d}\, \vec{s}$. If $\vec{s} = (\bot, \ldots, \bot)$, then $\vec{e} = (\bot, \ldots, \bot) \in R^{int}$. Otherwise $\vec{e} = \vec{s}\, l \in \delta^n \mathbb{Z} \subseteq R^{int}$. If $\vec{d} = (\bot, \ldots, \bot)$ or $\vec{s} = (\bot, \ldots, \bot)$, then $\vec{t} = (\bot, \ldots, \bot) \in R^{sto}$. Otherwise $(\vec{t} \mid L') = (\vec{s} \mid L') \in R_{L'}$, $\vec{t}\, l = \vec{d} \in \delta^n \mathbb{Z}$ and $\vec{t}\, l' = \vec{s}\, l' \in \delta^n \mathbb{Z}$ for all $l' \in Loc \setminus L'$ with $l' \neq l$, hence again $\vec{t} \in R^{sto}$. Thus we have proved that $Cont\, l\, R^{sto} \subseteq R^{int}$ and $Asgn\, l\, R^{int} R^{sto} \subseteq R^{sto}$, i.e. $(l, \ldots, l) \in R^{loc}$. □

As immediate consequences of Theorem 5.2 we obtain
– $fR^{sto} \subseteq R^{\gamma}$ whenever $f \in [\![sto \Rightarrow \gamma]\!]_L$ and $R \in OUT^L$
– $fR^{\tau} \subseteq R^{\sigma}$ whenever $f \in [\![\tau \to \sigma]\!]_L$ and $R \in OUT^L$
In more intuitive terms both can be summarized as

"A procedure $f \in [\![\sigma]\!]_L$ preserves all relations which are definable outside L."

This is the most important reasoning principle for proving observational congruences (Section 7) as well as other, more general properties of our denotational model like Theorem 5.3 below. A particular instance of this reasoning principle is obtained by permutations of locations: Let $\varphi : Loc \to Loc$ be a *finite permutation*, i.e. a bijective function whose 'support' $Supp(\varphi) =_{def} \{l \in Loc \mid \varphi\, l \neq l\}$ is finite. Then we define $R_{\varphi} \in DEF_2^{Supp(\varphi)}$ by

$$R_{\varphi}^{sto} = \{\bot\}^2 \cup \{(s, s \circ \varphi) \mid s \in Stores\}$$

In order to see that R_{φ} is indeed $Supp(\varphi)$-definable, note that it can be rewritten as

$$R_{\varphi}^{sto} = \{\bot\}^2 \cup \{\vec{s} \in Stores^2 \mid \forall l \in Supp(\varphi).\, s_1(\varphi\, l) = s_2 l \,\wedge\, s_1 =_{Loc \setminus Supp(\varphi)} s_2\}$$

and that $\varphi(Supp(\varphi)) = Supp(\varphi)$. If we finally let $Fix(L)$ denote the set of all finite permutations $\varphi : Loc \to Loc$ which leave the locations of L fixed (i.e. those with $L \cap Supp(\varphi) = \emptyset$), then we have

$$\varphi \in Fix(L) \;\Rightarrow\; R_{\varphi} \in OUT_2^L$$

We will now make use of these new relations in order to prove some important properties of the domains $[\![\sigma]\!]$ with $ord(\sigma) = 1$. But first we extend the notation for function application, function coincidence and for the variant of a function to bottom elements by defining

$$\begin{array}{ll} \bot_{sto}\, l = \bot_{int} & \text{for all } l \in Loc \\ \bot_{sto} =_L \bot_{sto} & \text{for all } L \in W \\ s[d/l] = \bot_{sto} & \text{if } s = \bot_{sto} \text{ or } d = \bot_{int} \end{array}$$

Theorem 5.3 (Properties of the first order domains). *Let $L \in W$, $s, s' \in Stores$, $l_1, \ldots, l_m \in Loc$ $(m \geqslant 0)$ and $\varphi \in Fix(L)$. Then*
(i) *If $f \in [\![\theta]\!]$, then $f\bot = \bot$.*
(ii) *If $f \in [\![iexp]\!]_L$, then $s =_L s' \Rightarrow fs = fs'$.*
(iii) *If $f \in [\![cmd]\!]_L$, then $fs \neq \bot \Rightarrow fs =_{Loc \setminus L} s$ and $s =_L s' \Rightarrow fs =_L fs'$.*
(iv) *If $f \in [\![loc^m \to iexp]\!]_L$, then $f(\varphi\, l_1) \ldots (\varphi\, l_m) s = f l_1 \ldots l_m (s \circ \varphi)$.*
(v) *If $f \in [\![loc^m \to cmd]\!]_L$, then $f(\varphi\, l_1) \ldots (\varphi\, l_m) s = (f l_1 \ldots l_m (s \circ \varphi)) \circ \varphi^{-1}$.*

Note that by (ii) and (iii), a function $f \in [\![\theta]\!]_L$ is uniquely determined by its restriction $f \mid Stores_L$. Intuitively, (ii) means that $f \in [\![iexp]\!]_L$ cannot read on locations outside L, (iii) means that $f \in [\![cmd]\!]_L$ can neither read nor write outside L and (iv) and

(v) mean that a function $f \in [\![loc^m \to \theta]\!]_L$ behaves uniformly on locations outside L. Taking into account that two stores can only be different on a finite set of locations, we can reformulate (ii) and (iii) as

(ii′) If $f \in [\![iexp]\!]_L$ and $l \in Loc \setminus L$, then $f(s[n/l]) = fs$ for all $n \in \mathbb{Z}$.

(iii′) If $f \in [\![cmd]\!]_L$ and $l \in Loc \setminus L$, then $f(s[n/l]) = (fs)[n/l]$ for all $n \in \mathbb{Z}$.

Proof. We prove (i), (iii) and (v); the proofs for (ii) and (iv) are similar.

(i) Let $\theta = sto \Rightarrow \gamma$ and $f \in [\![\theta]\!]_L$. Consider the unary ground relation R with $R^{loc} = Loc$, $R^{sto} = \{\bot_{sto}\}$ and $R^{int} = \{\bot_{int}\}$. R is clearly preserved by all $f \in AUX$ and $\delta^1 Loc \subseteq R^{loc}$, hence $R \in \Sigma^L$. This implies $fR^{sto} \subseteq R^{\gamma}$ and hence $f\bot_{sto} = \bot_{\gamma}$.

(iii) Let $f \in [\![cmd]\!]_L$.

If $s \in Stores$ and $l \in Loc \setminus L$, then let $R \in DEF_1^{\{l\}}$ be defined by $R^{sto} = \{\bot\} \cup \{t \in Stores \mid t\,l = s\,l\}$. Clearly $s \in R^{sto}$ and $fR^{sto} \subseteq R^{sto}$ because $R \in OUT^L$. This implies $fs \in R^{sto}$, hence $fs = \bot$ or $fs\,l = s\,l$. Thus we have proved that $fs \neq \bot$ implies $fs =_{Loc\setminus L} s$.

If $s, s' \in Stores$ with $s =_L s'$, then there is some $L' \in W$ with $L \cap L' = \emptyset$ and $s =_{Loc\setminus L'} s'$. Let $R \in DEF_2^{L'}$ with $R^{sto} = \{\bot\}^2 \cup \{\vec{t} \in Stores^2 \mid t_1 =_{Loc\setminus L'} t_2\}$. Then $(s, s') \in R^{sto}$ and $fR^{sto} \subseteq R^{sto}$ because $R \in OUT^L$. This implies $(fs, fs') \in R^{sto}$, hence $fs =_L fs'$.

(v) Let $f \in [\![loc^m \to cmd]\!]_L$. We know that $R_\varphi \in OUT_2^L$, and it is easy to see that $(\varphi\, l, l) \in R_\varphi^{loc}$ for all $l \in Loc$, hence $(f(\varphi\, l_1)\ldots(\varphi\, l_m)s, fl_1\ldots l_m(s \circ \varphi)) \in fR_\varphi^{loc}\ldots R_\varphi^{loc}R_\varphi^{sto} \subseteq R_\varphi^{sto}$, i.e. $f(\varphi\, l_1)\ldots(\varphi\, l_m)s = (fl_1\ldots l_m(s\circ\varphi))\circ\varphi^{-1}$.

We now conclude the definition of the denotational semantics by assigning meanings to the constants. We make extensive use of the auxiliary functions in the following definition. This does not only lead to a compact notation but it will also be helpful for later purposes. For every ALG-constant c we define the *meaning* $[\![c]\!]$ by

$[\![l]\!] \in D^{loc}$	$[\![n]\!] : D^{sto} \to D^{int}$
$[\![l]\!] = l$	$[\![n]\!] = Const_n$
$[\![succ]\!] : [\![iexp]\!] \to D^{sto} \to D^{int}$	$[\![pred]\!] : [\![iexp]\!] \to D^{sto} \to D^{int}$
$[\![succ]\!]fs = Succ(fs)$	$[\![pred]\!]fs = Pred(fs)$
$[\![cont]\!] : [\![loc]\!] \to D^{sto} \to D^{int}$	$[\![asgn]\!] : [\![loc]\!] \to [\![iexp]\!] \to D^{sto} \to D^{sto}$
$[\![cont]\!] = Cont$	$[\![asgn]\!]\, l\, fs = Asgn\, l\,(fs)s$
$[\![skip]\!] : D^{sto} \to D^{sto}$	
$[\![skip]\!]s = s$	

$$[\![cond_{sto\Rightarrow\gamma}]\!] : [\![iexp]\!] \to [\![sto \Rightarrow \gamma]\!] \to [\![sto \Rightarrow \gamma]\!] \to D^{sto} \to D^{\gamma}$$
$$[\![cond_{sto\Rightarrow\gamma}]\!]\, bfgs = Cond_\gamma(bs)(fs)(gs)$$

$$[\![seq_{sto\Rightarrow\gamma}]\!] : [\![cmd]\!] \to [\![sto \Rightarrow \gamma]\!] \to D^{sto} \to D^{\gamma}$$
$$[\![seq_{sto\Rightarrow\gamma}]\!]fgs = g(fs)$$

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

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

$$[\![Y_\sigma]\!] : [\![\sigma \to \sigma]\!] \to [\![\sigma]\!]$$
$$[\![Y_\sigma]\!] = \mu_{[\![\sigma]\!]}$$

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

Note that the fixed point operators $\mu_{[\![\sigma]\!]}$ are (well-defined) locally continuous Σ-homomorphisms by Theorem 4.6, hence $[\![Y_\sigma]\!] \in [\![(\sigma \to \sigma) \to \sigma]\!]_\emptyset$ for every procedure type σ. The meanings of the other constants are also well-defined, but it remains to be proved that they are 'contained in the model', i.e. that $[\![c]\!] \in [\![\tau]\!]$ for every constant c of type τ. The first step into this direction is to show that the particular choice of l in the clauses for $[\![new_{iexp}]\!]$ and $[\![new_{cmd}]\!]$ does not play a role, i.e. instead of $l = next\,(supp(f))$ we can use any arbitrary location $l \notin supp\,(f)$. This will be needed in Proposition 5.6 for proving the local continuity of the *new*-operators.

Proposition 5.4. *For every* $l \in Loc \setminus supp\,(f)$ *we have*

$$[\![new_{iexp}]\!] f s = f\, l\, (s\,[0/l])$$
$$[\![new_{cmd}]\!] f s = (f\, l\, (s\,[0/l]))\,[s\, l/l]$$

Proof. We only consider $[\![new_{cmd}]\!]$, the proof for $[\![new_{iexp}]\!]$ is similar. Let $L \in W$ be such that $f \in [\![loc \to cmd]\!]_L$, let $l_1, l_2 \in Loc \setminus L$ and let φ be the *transposition* of l_1 and l_2, i.e. the permutation which only interchanges l_1 and l_2. Then we obtain

$$\begin{aligned}
f\, l_1(s[0/l_1]) &=_{L\cup\{l_1\}} f\, l_1(s[0/l_1][0/l_2]) \\
&\qquad \text{by Theorem 5.3 (iii), because } f\, l_1 \in [\![cmd]\!]_{L\cup\{l_1\}} \\
&= (f\, l_2\, (s[0/l_1][0/l_2])) \circ \varphi \\
&\qquad \text{by Theorem 5.3 (v), because } \varphi = \varphi^{-1} \\
&=_L f\, l_2\, (s[0/l_1][0/l_2]) \\
&\qquad \text{because } \varphi \in Fix(L) \\
&=_{L\cup\{l_2\}} f\, l_2\, (s[0/l_2]) \\
&\qquad \text{by Theorem 5.3 (iii), because } f\, l_2 \in [\![cmd]\!]_{L\cup\{l_2\}}
\end{aligned}$$

This implies $(f\, l_1\, (s[0/l_1]))\,[s\, l_1/l_1] =_L (f\, l_2\, (s[0/l_2]))\,[s\, l_2/l_2]$ and if they are different from $\bot$, then they both coincide with s on $Loc \setminus L$ by Theorem 5.3(iii), hence they are equal in any case.

Now let $l_1, l_2 \in Loc \setminus supp\,(f)$, say $l_1 = next\,(supp\,(f))$. Then there are $L_1, L_2 \in W$ with $f \in [\![loc \to cmd]\!]_{L_i}$ and $l_i \in Loc \setminus L_i$ for $i = 1,2$, and by choosing some arbitrary $l \in Loc \setminus (L_1 \cup L_2)$ we obtain $[\![new_{cmd}]\!] f\, s = (f\, l_1\, (s[0/l_1]))[s\, l_1/l_1] = (f\, l\, (s[0/l]))[s\, l/l] = (f\, l_2\, (s[0/l_2]))[s\, l_2/l_2]$. □

Proposition 5.4 captures the 'operational intuition' that the particular choice of the new location which we bind to a local variable does not play a role, and thus it already gives us some confidence in our denotational semantics. Indeed, Proposition 5.4 will be needed for the computational adequacy as well as for the full abstraction of our denotational model.

As to computational adequacy, note that there is a gap between the operational and the denotational definition of a 'new' location l. In the operational semantics we work with marked stores ms and we let $l = next(dom(ms))$ in rule (new-init), i.e. we choose l to be the 'first location which is not marked as active'. In the denotational semantics we work with ordinary stores and we let $l = next(supp(f))$ where f corresponds to the body of the block in which the local variable is declared, i.e. we choose l to be the 'first location to which the body of the block does not have access'. This gap can be closed if we know that the denotational definition is independent of l as long as $l \notin supp(f)$ and that $supp(f) \subseteq dom(ms)$, i.e. that the body of the block can only have access to locations which are marked as active.

While these considerations about computational adequacy are somewhat technical (and could perhaps be avoided by an alternative definition of the denotational semantics), the role of Proposition 5.4 for full abstraction is more significant. If the particular choice of l in the definition of $[\![new_\theta]\!]$ did play a role, then the meaning of a block with two local variables could depend on the order in which these local variables are declared. This means that certain observational congruences (e.g. Example 7.3) would not be provable in our denotational semantics and thus full abstraction would indeed fail.

We continue with a purely technical lemma.

Lemma 5.5. *Let $L \in W, k \in \mathbb{N}$ and let $f : [\![\tau_1]\!] \to \cdots \to [\![\tau_k]\!] \to D^{sto} \to D^\gamma$. If*

(1) $f R^{\tau_1} \ldots R^{\tau_k} R^{sto} \subseteq R^\gamma$ for every $R \in \Sigma^L$ and

(2) $f d_1 \ldots d_{j-1}$ is continuous on $[\![\tau_j]\!]_{L'}$ for all $j \in \{1,\ldots,k\}$, $(d_1,\ldots,d_{j-1}) \in [\![\tau_1]\!] \times \cdots \times [\![\tau_{j-1}]\!]$ and $L' \in W$

then $f \in [\![\tau_1 \to \cdots \to \tau_k \to sto \Rightarrow \gamma]\!]_L$.

Proof. By induction on k.

$k = 0$: If $f R^{sto} \subseteq R^\gamma$ for all $R \in \Sigma^L$, then $f \in [\![sto \Rightarrow \gamma]\!]_L$ per definition.

$k > 0$: Assume that (1) and (2) hold for f. If $L' \in W$, $L \subseteq L'$ and $d_1 \in [\![\tau_1]\!]_{L'}$, then we obtain for all $R \in \Sigma_n^{L'}$

$$f d_1 R^{\tau_2} \ldots R^{\tau_k} R^{sto} \subseteq f\, (\delta^n [\![\tau_1]\!]_{L'}) R^{\tau_2} \ldots R^{\tau_k} R^{sto} \subseteq f R^{\tau_1} \ldots R^{\tau_k} R^{sto} \subseteq R^\gamma$$

This means that (1) holds for $f d_1$ with L' instead of L, and of course (2) also holds for $f d_1$. Hence $f d_1 \in [\![\tau_2 \to \cdots \to \tau_k \to sto \Rightarrow \gamma]\!]_{L'}$ by induction hypothesis, i.e. we have proved $f([\![\tau_1]\!]_{L'}) \subseteq [\![\tau_2 \to \cdots \to \tau_k \to sto \Rightarrow \gamma]\!]_{L'}$ for all

$L' \supseteq L$ and thus also $f\,[\![\tau_1]\!] \subseteq [\![\tau_2 \to \cdots \to \tau_k \to sto \Rightarrow \gamma]\!]$. But then (1) means that $f\,R^{\tau_1} \subseteq R^{\tau_2 \to \cdots \to \tau_k \to sto \Rightarrow \gamma}$ for every $R \in \Sigma^L$ and from (2) we know in particular that f itself is continuous on all $[\![\tau_1]\!]_{L'}$. Hence $f \in [\![\tau_1 \to \cdots \to \tau_k \to sto \Rightarrow \gamma]\!]_L$. $\square$

Now we are ready to prove

Proposition 5.6. *If c is a constant of procedure type σ, then $[\![c]\!] \in [\![\sigma]\!]_\emptyset$.*

Proof. We only consider two sample cases, namely $c \equiv asgn$ as a routine case and $c \equiv new_{cmd}$ as the most interesting case.

Case 1: $c \equiv asgn$

If $R \in \Sigma$ $(= \Sigma^\emptyset)$, then $[\![asgn]\!]R^{loc}R^{iexp}R^{sto} \subseteq Asgn\,R^{loc}\,(R^{iexp}R^{sto})\,R^{sto} \subseteq Asgn\,R^{loc}R^{int}R^{sto} \subseteq R^{sto}$. The function $[\![asgn]\!]$ itself is continuous and it is easy to see that $[\![asgn]\!]\,l$ is continuous on $[\![iexp]\!]_L$ for all $l \in D^{loc}$ and $L \in W$. Hence Lemma 5.5 implies $[\![asgn]\!] \in [\![loc \to iexp \to cmd]\!]_\emptyset$.

Case 2: $c \equiv new_{cmd}$

Let $R \in \Sigma$, $\vec{f} \in R^{loc \to cmd}$ and $\vec{s} \in R^{sto}$. Let $L \in W$ with $f_1, \ldots, f_n \in [\![loc \to cmd]\!]_L$ and $l \in Loc \setminus L$ with $(l, \ldots, l) \in R^{loc}$. Then, by Proposition 5.4,

$$\begin{aligned}
[\![new_{cmd}]\!]\vec{f}\vec{s} &= Asgn\,l\,(Cont\,l\,\vec{s})\,(\vec{f}\,l\,(Asgn\,l\,0\,\vec{s}))\\
&\in Asgn\,R^{loc}\,(Cont\,R^{loc}R^{sto})\,(R^{loc\to cmd}R^{loc}\,(Asgn\,R^{loc}R^{int}R^{sto}))\\
&\subseteq Asgn\,R^{loc}R^{int}\,(R^{cmd}R^{sto})\\
&\subseteq R^{sto}
\end{aligned}$$

Now let $L' \in W$. By choosing some $l \in Loc \setminus L'$ we obtain by Proposition 5.4: $[\![new_{cmd}]\!]f = \underline{\lambda} s \in Stores.\,Asgn\,l\,(Cont\,s\,l)\,(f\,l\,(Asgn\,l\,0\,s))$ for all $f \in [\![loc \to cmd]\!]_L$. This shows that $[\![new_{cmd}]\!]$ is continuous on $[\![loc \to cmd]\!]_L$ and hence $[\![new_{cmd}]\!] \in [\![(loc \to cmd) \to cmd]\!]_\emptyset$ by Lemma 5.5. $\square$

Theorem 4.5 and Proposition 5.6 allow us to define the meaning of ALG -terms in an environment model [18] of the simply typed λ-calculus: Let *Env* be the set of all *environments*, i.e. the set of all type preserving functions

$$\eta : \bigcup_{\tau \in Type} Id^\tau \to \bigcup_{\tau \in Type} [\![\tau]\!]$$

Then, for every $M \in \text{ALG}^\tau$, the *meaning* $[\![M]\!] : Env \to [\![\tau]\!]$ is inductively defined by

$$\begin{aligned}
[\![c]\!]\eta &= [\![c]\!] \quad \text{as defined before}\\
[\![x]\!]\eta &= \eta\,x\\
[\![MN]\!]\eta &= ([\![M]\!]\eta)([\![N]\!]\eta)\\
[\![\lambda x^\tau.M]\!]\eta &= \underline{\lambda} d \in [\![\tau]\!].\,[\![M]\!]\eta[d/x]
\end{aligned}$$

As usual, $[\![M]\!]\eta$ only depends on the restriction of η to *free*(M); in particular it is independent of η, if M is closed, and then we usually write $[\![M]\!]$ instead of $[\![M]\!]\eta$.

Proposition 5.7. (i) If $M \in \textsc{Alg}_L^\tau$ and $\eta x^{\tau'} \in [\![\tau']\!]_L$ for every $x^{\tau'} \in$ *free*(M), then $[\![M]\!]\eta \in [\![\tau]\!]_L$.

(ii) If $M \in c\text{-}\textsc{Alg}_L^\tau$, then $[\![M]\!] \in [\![\tau]\!]_L$.

Part (ii) captures our intuition that a closed Alg-term has access only to those locations which explicitly occur in it and *not* to those which are temporarily bound to its local variables. An open term may (of course) also have access to the locations which are bound to its free variables and it may have 'indirect' access to additional locations via the functions which are bound to its free procedure identifiers.

Proof. Of course it is sufficient to prove (i). For location free terms we can apply general principles: For every constant c of procedure type σ the meaning $[\![c]\!] \in [\![\sigma]\!]_\emptyset$ can be considered as a morphism from the terminal object T to the object $[\![\sigma]\!]$, hence – by the categorical semantics of the λ-calculus [4] – the meaning of a term $M \in \textsc{Alg}_\emptyset^\tau$ with *free*$(M) = \{x_1^{\tau_1}, \cdots, x_k^{\tau_k}\}$ is a morphism from $[\![\tau_1]\!] \times \cdots \times [\![\tau_n]\!]$ to $[\![\tau]\!]$, i.e. it maps $[\![\tau_1]\!]_L \times \cdots \times [\![\tau_n]\!]_L$ to $[\![\tau]\!]_L$. The generalization to terms *with* locations is straightforward. □

Note that our semantics of Alg -terms *with* locations is *not* a standard categorical semantics of the simply typed λ-calculus, because location constants cannot be considered as morphisms from the terminal object T to the object $[\![loc]\!]$. This observation is not harmful, because we will make no further use of the categorical view, but it should be kept in mind in order to avoid any confusion.

We conclude this section by explicitly presenting the meaning of a block with a local variable declaration. Remember that **new** x **in** M **end** is syntactic sugar for $new_\theta(\lambda x^{loc}.M)$, if M is of type θ. Thus we obtain

$$[\![\textbf{new}\, x \textbf{ in } M \textbf{ end}]\!]\,\eta s = [\![M]\!]\,\eta[l/x]\,s[0/l] \qquad \text{if } M \in \textsc{Alg}^{iexp}$$
$$[\![\textbf{new}\, x \textbf{ in } M \textbf{ end}]\!]\,\eta s = ([\![M]\!]\,\eta[l/x]\,s[0/l])\,[s\,l/l] \quad \text{if } M \in \textsc{Alg}^{cmd}$$

where, by Proposition 5.4, l is an *arbitrary* location in $Loc \setminus supp([\![\lambda x.M]\!]\,\eta)$. The possibility to choose l freely from an infinite set will be important in the following sections, because we will often need a location which is different from finitely many given ones. In such cases we sometimes briefly say that we choose a *new* location and leave it to the reader to spell out precisely what is meant by 'new' in a particular case.

6. Computational adequacy

In this section we will see that our denotational semantics is *computationally adequate*. Computational adequacy means [12] that the observable behavior *beh*(P) of a program P can be (easily) derived from its denotational meaning $[\![P]\!]$. Concretely, we will show that for every $n \in \mathbb{Z}$

$$n \in beh(P) \Leftrightarrow [\![P]\!]\, s_{init} = n$$

where s_{init} is the constant 0 store, i.e. the (unique) store in $Stores_{\emptyset}$. This implies that $beh(P)$ contains at most one element (as we know already from Theorem 3.2) and that $beh(P) = \emptyset \Leftrightarrow [\![P]\!]\, s_{init} = \bot$.

We begin with '$\Rightarrow$'. For a purely functional language, this direction is usually proved by showing that each transition step of the operational semantics preserves the denotational meaning of terms [4, 39]. This is also the main idea for our proof, but as our transition relation works on configurations as opposed to terms, we first extend our meaning function: For every marked store ms we define $\overline{ms} \in Stores$ by

$$\overline{ms}\ l = \begin{cases} ms\, l & \text{if } l \in dom(ms) \\ 0 & \text{otherwise} \end{cases}$$

and for every *consistent* configuration $K \in Conf_L^{sto \Rightarrow \gamma}$ we define its *meaning* $[\![K]\!] \in D^{\gamma}$ inductively by

- $[\![(M, ms)]\!] = [\![M]\!]\, \overline{ms}$
- $[\![succ\, K]\!] = Succ\, [\![K]\!]$
- $[\![pred\, K]\!] = Pred\, [\![K]\!]$
- $[\![(asgn\, l\, K, ms)]\!] = Asgn\, l\, [\![K]\!]\, \overline{ms}$
- $[\![(cond_{sto \Rightarrow \gamma} KMN, ms)]\!] = Cond_{\gamma} [\![K]\!]\, ([\![M]\!]\, \overline{ms})\, ([\![N]\!]\, \overline{ms})$
- $[\![seq_{\theta} KM]\!] = [\![M]\!]\, [\![K]\!]$
- $[\![dealloc_{iexp}\, l\, K]\!] = [\![K]\!]$
- $[\![dealloc_{cmd}\, l\, K]\!] = Asgn\, l\, 0\, [\![K]\!]$
- $[\![pcond K_1 K_2 K_3]\!] = Pcond\, [\![K_1]\!]\, [\![K_2]\!]\, [\![K_3]\!]$

With this definition it is straightforward to prove

Lemma 6.1. *Every transition step is meaning-preserving, i.e.*
(i) $M \twoheadrightarrow M'$ *implies* $[\![M]\!] = [\![M']\!]$ *for closed terms* M, M'
(ii) $K \rightarrow K'$ *implies* $[\![K]\!] = [\![K']\!]$ *for consistent configurations* K, K'.

From this lemma we obtain the '$\Rightarrow$'-part of computational adequacy. For the '$\Leftarrow$'-part we define a relation $\leqslant_L^{\tau} \subseteq [\![\tau]\!]_L \times c\text{-}\mathrm{ALG}_L^{\tau}$ for every type τ and every $L \in W$ such that
- $(\leqslant^{\tau})_{\tau \in Type, L \in W}$ is a Kripke logical relation between applicative structures [18]
- $[\![M]\!] \leqslant_L^{\tau} M$ for every $M \in c\text{-}\mathrm{ALG}_L^{\tau}$

These relations are defined by induction on τ as follows

$$l \leqslant_L^{loc} l' \;\Leftrightarrow\; l = l'$$

$$f \leqslant_L^{iexp} M \;\Leftrightarrow\; \forall L' \supseteq L, s \in Stores, n \in \mathbb{Z}.\ f s = n \Rightarrow \exists ms. (M, s \mid L') \xrightarrow{*} (n, ms)$$

$$f \leqslant_L^{cmd} M \;\Leftrightarrow\; \forall L' \supseteq L, s, s' \in Stores.\; fs = s' \;\Rightarrow\; (M, s \mid L') \xrightarrow{*} (skip, s' \mid L')$$

$$f \leqslant_L^{\tau \to \sigma} M \;\Leftrightarrow\; \forall L' \supseteq L, d \in [\![\tau]\!]_{L'}, P \in c\text{-}\mathrm{ALG}_{L'}^{\tau}.\; d \leqslant_{L'}^{\tau} P \;\Rightarrow\; fd \leqslant_{L'}^{\sigma} MP$$

and their relevant properties are summarized in

Lemma 6.2. *Let σ be a procedure type, let $L \in W$, $f, g \in [\![\sigma]\!]_L$, $\Delta \subseteq [\![\sigma]\!]_L$ directed and $M, N \in c\text{-}\mathrm{ALG}_L^{\sigma}$. Then*

(i) $\bot_\sigma \leqslant_L^\sigma M$

(ii) $f \sqsubseteq g \wedge g \leqslant_L^\sigma M \Rightarrow f \leqslant_L^\sigma M$

(iii) $(\forall f \in \Delta. f \leqslant_L^\sigma M) \Rightarrow \bigsqcup \Delta \leqslant_L^\sigma M$

(iv) $M \multimap\!\!\!\!\rhd N \wedge f \leqslant_L^\sigma N \Rightarrow f \leqslant_L^\sigma M$

Now the key to computational adequacy is

Lemma 6.3. $M \in c\text{-}\mathrm{ALG}_L^\tau \Rightarrow [\![M]\!] \leqslant_L^\tau M$

Proof. As usual [39] this assertion is first generalized to open terms:

Let $M \in \mathrm{ALG}_L^\tau$ with $free(M) \subseteq \{x_1^{\tau_1}, \ldots, x_k^{\tau_k}\}$ and let $d_i \in [\![\tau_i]\!]_L, N_i \in c\text{-}\mathrm{ALG}_L^{\tau_i}$ with $d_i \leqslant_L^{\tau_i} N_i (i = 1, \ldots, k)$. Then $[\![M]\!]\eta[\bar{d}/\bar{x}] \leqslant_L^\tau M[\bar{x} := \bar{N}]$ for every $\eta \in Env$.

This generalized assertion is proved by induction on the structure of M.

Case 1: M constant of type *loc*

Then $M \equiv l \in L$ and $[\![M]\!]\eta[\bar{d}/\bar{x}] = l \leqslant_L^{loc} l \equiv M[\bar{x}/\bar{N}]$.

Case 2: M constant of procedure type σ

It must be proved that $[\![M]\!] \leqslant_L^\sigma M$ for *all* $L \in W$. To this end it is sufficient to prove $[\![M]\!] \leqslant_\emptyset^\sigma M$, because $\leqslant_\emptyset^\sigma$ is the strongest relation among all $\leqslant_L^\sigma$. We consider the constants which are typical for an imperative language.

(i) $M \equiv cont : \; loc \to iexp$

Let $L \in W$, $d \in [\![loc]\!]_L$ and $l \in c\text{-}\mathrm{ALG}_L^{loc} = L$ with $d \leqslant_L^{loc} l$. Then $d = l$, hence we must prove $[\![cont]\!]\, l \leqslant_L^{iexp} cont\, l$. Let $L' \supseteq L$, $s \in Stores$ and $n \in \mathbb{Z}$ with $[\![cont]\!]\, l\, s = n$. Then $s\, l = n$, and as $l \in L'$ we obtain $(cont\, l, s \mid L') \to (n, s \mid L')$.

(ii) $M \equiv asgn : \; loc \to iexp \to cmd$

Let L, d and l be as in (i). Further let $L' \supseteq L$, $f \in [\![iexp]\!]_{L'}$ and $N \in c\text{-}\mathrm{ALG}_{L'}^{iexp}$ with $f \leqslant_{L'}^{iexp} N$. Then we must prove $[\![asgn]\!]\, l\, f \leqslant_{L'}^{cmd} asgn\, l\, N$. Let $L'' \supseteq L'$ and $s, s' \in Stores$ with $[\![asgn]\!]\, l\, f s = s'$. Then there is some $n \in \mathbb{Z}$ with $fs = n$ and $s' = s[n/l]$, and as $f \leqslant_{L'}^{iexp} N$ we know that $(N, s \mid L'') \xrightarrow{*} (n, ms)$ for some marked store ms. As $l \in L''$, this finally implies $(asgn\, l\, N, s \mid L'') \to (asgn\, l\, (N, s \mid L''), s \mid L'') \xrightarrow{*} (asgn\, l\, (n, ms), s \mid L'') \to (skip, s' \mid L'')$.

(iii) $M \equiv new_{cmd} : \; (loc \to cmd) \to cmd$

Let $L \in W$, $f \in [\![loc \to cmd]\!]_L$ and $N \in c\text{-}\mathrm{ALG}_L^{loc \to cmd}$ with $f \leqslant_L^{loc \to cmd} N$. We must prove $[\![new_{cmd}]\!]\, f \leqslant_L^{cmd} new_{cmd} N$. Let $L' \supseteq L$ and $s, s' \in Stores$ with $[\![new_{cmd}]\!]\, fs = s'$. Let $l = next(L')$, hence $l \notin L' \supseteq L \supseteq supp(f)$. Then there is some $s'' \in Stores$ with $f\, l\, (s[0/l]) = s''$ and $s' = s''[s\, l/l]$. As $f\, l \leqslant_{L \cup \{l\}}^{cmd} Nl$, we know that $(Nl, s[0/l] \mid L' \cup$

$\{l\}) \xrightarrow{*} (skip, s'' \mid L' \cup \{l\})$ and from this we obtain $(new_{cmd}N, s \mid L') \rightarrow dealloc_{cmd}\, l\,(Nl,$ $(s \mid L')[0/l]) \equiv dealloc_{cmd}\, l\,(Nl, s[0/l] \mid L' \cup \{l\}) \xrightarrow{*} dealloc_{cmd}(skip, s'' \mid L' \cup \{l\}) \rightarrow$ $(skip, s'' \mid L') \equiv (skip, s' \mid L')$.

The proofs for the remaining constants and for the remaining cases (application and λ-abstraction) follow well-known arguments from functional languages [39]. □

Theorem 6.4 (Computational adequacy). *For every program P and every* $n \in \mathbb{Z}$

$$n \in beh(P) \;\Leftrightarrow\; [\![P]\!]\, s_{init} = n$$

Proof.

$$\begin{array}{lll} \text{`}\Rightarrow\text{'}: \; n \in beh(P) & \Rightarrow (P, ms_{init}) \xrightarrow{*} (n, ms_{init}) & \text{per definition of } beh(P) \\ & \Rightarrow [\![(P, ms_{init})]\!] = [\![(n, ms_{init})]\!] & \text{by Lemma 6.1} \\ & \Rightarrow [\![P]\!]\, s_{init} = [\![n]\!]\, s_{init} & \text{because } \overline{ms_{init}} = s_{init} \\ & \Rightarrow [\![P]\!]\, s_{init} = n & \end{array}$$

'$\Leftarrow$': By Lemma 6.3 we have $[\![P]\!] \leqslant_{\emptyset}^{iexp} P$, hence

$$\begin{array}{lll} [\![P]\!]\, s_{init} = n & \Rightarrow \exists ms.\, (P, ms_{init}) \xrightarrow{*} (n, ms) & \text{because } ms_{init} = s_{init} \mid \emptyset \\ & \Rightarrow (P, ms_{init}) \xrightarrow{*} (n, ms_{init}) & \text{by Theorem 3.2 (ii)} \\ & \Rightarrow n \in beh(P) \quad \Box & \end{array}$$

Computational adequacy allows us to prove observational congruences. Here is the precise definition:

Definition 6.5. A *context* $C[\ ]$ is a term with a hole; $C[M]$ denotes the term which is obtained from $C[\ ]$ by placing M into the hole. $C[\ ]$ is a *program context* for M and N if both $C[M]$ and $C[N]$ are programs; M and N are *observationally congruent* (denoted $M \approx N$) if $beh(C[M]) = beh(C[N])$ for every program context $C[\ ]$.

Theorem 6.6 (Observational congruence). $[\![M]\!] = [\![N]\!] \;\Rightarrow\; M \approx N$

Proof. $[\![M]\!] = [\![N]\!]$ implies $[\![C[M]]\!] = [\![C[N]]\!]$ for every program context $C[\ ]$ by the compositionality of $[\![\ ]\!]$ and then $beh(C[M]) = beh(C[N])$ by Theorem 6.4. □

7. Observational congruences

In this section we will illustrate by a series of examples how to prove particular observational congruences with the aid of Theorem 6.6. Most of the examples have already appeared in the literature [8, 13, 22], some of them originally served as *counter*-examples for earlier denotational models of ALGOL-like languages, i.e. they were used to prove that these models are *not* fully abstract [8, 13].

We will no longer slavishly stick to the ALG-syntax, but freely use operators like $+, -, *, \mathbf{div}, abs, =, \geqslant, \neg, \wedge, \vee, \ldots$ with their standard interpretation, in particular each of them is assumed to be *strict* in all its arguments. These operators are of course definable by closed ALG-terms.

The intuitive idea behind all our examples is that a global procedure cannot have access to a local variable. The corresponding fo rmal argumentation in the denotational model works as follows: If f is the function which is bound to a global procedure identifier y^σ, then there is some $L \in W$ such that $f \in [\![\sigma]\!]_L$ and we may assume that all locations $l_1, \ldots, l_n$ which are bound to the local variables are *not* contained in L. The desired semantic equality then usually follows by applying Theorem 5.3 (in the case of a first order procedure) or by choosing some appropriate $\{l_1, \ldots, l_n\}$-definable ground relation R and exploiting the fact that f preserves (the logical relation induced by) R.

Example 7.1. $[\![\mathbf{new}\, x \textbf{ in } M \textbf{ end};\ y^\sigma]\!] = [\![\mathbf{new}\, x \textbf{ in } M;\ y^\sigma \textbf{ end}]\!]$

Proof. Let $\eta \in Env$, $s \in Stores$ and $L \in W$ with $\eta\, y \in [\![\sigma]\!]_L$. If $\sigma = cmd$, then

$$\begin{aligned}
[\![\mathbf{new}\, x \textbf{ in } M \textbf{ end}; y]\!]\,\eta\, s &= \eta\, y\,([\![M]\!]\,\eta[l/x]\,(s\,[0/l])\,[s\,l/l]) \\
&\qquad \text{for some new location } l \notin L \\
&= (\eta\, y\,([\![M]\!]\,\eta[l/x]\,(s\,[0/l]))\,[s\,l/l] \\
&\qquad \text{by Theorem 5.3 (iii}')\\
&= ([\![M; y]\!]\,\eta[l/x]\,(s\,[0/l]))\,[s\,l/l] \\
&= [\![\mathbf{new}\, x \textbf{ in } M; y \textbf{ end}]\!]\,\eta\, s
\end{aligned}$$

The proof for $\sigma = iexp$ is similar and the generalization to arbitrary procedure types σ is a routine calculation in the λ-calculus (remember that sequencing and *new*-operators for higher types have been introduced as syntactic sugar). □

The intuition for Example 7.1 is that the global procedure y cannot read on the local variable x, hence it does not matter whether y is inside or outside the scope of the local variable. As an immediate consequence of Example 7.1 we obtain

$$[\![\mathbf{new}\, x \textbf{ in } x := n;\ y \textbf{ end}]\!] = [\![\mathbf{new}\, x \textbf{ in } x := n \textbf{ end};\ y]\!] = [\![skip;\ y]\!] = [\![y]\!]$$

which is essentially Example 1 in [13].

Example 7.2. $[\![y^{cmd};\ \mathbf{new}\, x \textbf{ in } M \textbf{ end}]\!] = [\![\mathbf{new}\, x \textbf{ in } y^{cmd}; M \textbf{ end}]\!]$

Proof. We only consider the case $M \in \text{ALG}^{cmd}$. Let $\eta \in Env$, $s \in Stores$ and $L \in W$ with $\eta\, y \in [\![cmd]\!]_L$. Then

$$\begin{aligned}
[\![y;\ \mathbf{new}\, x\ \mathbf{in}\ M\ \mathbf{end}]\!]\,\eta\, s &= [\![\mathbf{new}\, x\ \mathbf{in}\ M\ \mathbf{end}]\!]\,\eta\,(\eta\, y\, s)\\
&= ([\![M]\!]\,\eta[l/x]\,((\eta\, y\, s)\,[0/l]))\,[\eta\, y\, s\, l/l]\\
&\qquad \text{for some new location } l \notin L\\
&= ([\![M]\!]\,\eta[l/x]\,(\eta\, y\,(s\,[0/l])))\,[s\, l/l]\\
&\qquad \text{by Theorem 5.3 (iii) and (iii}')\\
&= ([\![y; M]\!]\,\eta[l/x]\,(s\,[0/l]))\,[s\, l/l]\\
&= [\![\mathbf{new}\, x\ \mathbf{in}\ y; M\ \mathbf{end}]\!]\,\eta\, s \qquad \square
\end{aligned}$$

The intuition for Example 7.2 is that the global procedure y can neither read nor write on the local variable x, hence moving the procedure call of y into the scope of the local variable has no influence on the computation of y or M. As an immediate consequence of Example 7.2 we obtain

$$\begin{aligned}
[\![\mathbf{new}\, x\ \mathbf{in}\ y;\ \mathbf{if}\ !x = 0\ \mathbf{then}\ \Omega\ \mathbf{end}]\!] &= [\![y;\ \mathbf{new}\, x\ \mathbf{in}\ \mathbf{if}\ !x = 0\ \mathbf{then}\ \Omega\ \mathbf{end}]\!]\\
&= [\![\Omega]\!]
\end{aligned}$$

which is essentially Example 2 in [13].

Example 7.3. $[\![\mathbf{new}\, x,x'\ \mathbf{in}\ M\ \mathbf{end}]\!] \ = \ [\![\mathbf{new}\, x',x\ \mathbf{in}\ M\ \mathbf{end}]\!]$

Proof. For $M \in \text{ALG}^{cmd}$ we obtain

$$\begin{aligned}
[\![\mathbf{new}\, x,x'\ \mathbf{in}\ M\ \mathbf{end}]\!]\,\eta\, s &= [\![M]\!]\,\eta[l/x][l'/x']\,(s\,[0/l][0/l'])\,[s\, l/l][s\, l'/l']\\
&\qquad \text{where } l, l' \in Loc \setminus supp([\![\lambda x.M]\!]\,\eta) \text{ and } l \neq l'\\
&= [\![\mathbf{new}\, x',x\ \mathbf{in}\ M\ \mathbf{end}]\!]\,\eta\, s \quad \square
\end{aligned}$$

and similarly for $M \in \text{ALG}^{iexp}$. The generalization to $M \in \text{ALG}^{\sigma}$ is routine. $\square$

Example 7.3 is a generalization of Example 3 in [13]. Note that it is *not* an α-conversion, because we do not rename x and x' inside the block body M. The crucial point is that it does not matter which location we bind to the first local variable and which to the second.

Example 7.4. $[\![\mathbf{new}\, x\ \mathbf{in}\ y^{cmd \to cmd}(x := !x + 1);\ \mathbf{if}\ !x \geqslant 0\ \mathbf{then}\ \Omega\ \mathbf{end}]\!] \ = \ [\![\Omega]\!]$

Proof. Let $\eta \in Env$, $s \in Stores$ and $L \in W$ with $\eta\, y \in [\![cmd \to cmd]\!]_L$. We may assume that the location l which is bound to the local variable x is not contained in

L. If we now choose $R \in DEF_1^{\{l\}}$ with $R^{sto} = \{\bot\} \cup \{t \in Stores \mid t\,l \geqslant 0\}$, then we have $s\,[0/l] \in R^{sto}$ and $[\![x := !x + 1]\!]\,\eta\,[l/x] \in R^{cmd}$. Hence the store $t = [\![y\,(x := !x + 1)]\!]\,\eta\,[l/x]\,(s\,[0/l])$ is contained in $\eta\,y\,R^{cmd}R^{sto} \subseteq R^{sto}$, i.e. $t = \bot$ or $t\,l \geqslant 0$, and this easily implies the desired equality. □

Example 7.4 is similar to Example 4 in [13]. It illustrates "a form of *representational abstraction*, which is one of the main themes of modern programming methodology" [22], in particular of object oriented programming: The local variable x may be considered as the *instance variable* of a *counter object* which is initially set to 0 and which can only be accessed through a single *method*, namely through the parameterless procedure[6] $x := !x + 1$. Although we do not know how often this method is used by the *client* y, we can be sure that the *representation invariant* $!x \geqslant 0$ of the counter object will be preserved, and this finally implies that the whole block must diverge.

Example 7.5. For $i = 1,2$ let

$$M_i \;\equiv\; \textbf{new}\,x\ \textbf{in}\ y^{cmd \to iexp \to iexp}(x := !x + i)(!x\ \textbf{div}\ i)\ \textbf{end}$$

Then $[\![M_1]\!] = [\![M_2]\!]$.

Proof. Let $\eta \in Env$, $s \in Stores$ and $L \in W$ with $\eta\,y \in [\![cmd \to iexp \to iexp]\!]_L$. Again we may assume that the new location l is not contained in L. We choose $R \in DEF_2^{\{l\}}$ with $R^{sto} = \{\bot\}^2 \cup \{\vec{s} \in Stores^2 \mid s_2\,l = 2 * s_1\,l \wedge s_1 =_{Loc\setminus\{l\}} s_2\}$. Then we have $(s\,[0/l], s\,[0/l]) \in R^{sto}$, $([\![x := !x + 1]\!]\,\eta\,[l/x], [\![x := !x + 2]\!]\,\eta\,[l/x]) \in R^{cmd}$ and $([\![!x\ \textbf{div}\ 1]\!]\,\eta\,[l/x], [\![!x\ \textbf{div}\ 2]\!]\,\eta\,[l/x]) \in R^{iexp}$. Hence the pair (d_1, d_2) with $d_i = [\![y\,(x := !x + i)(!x\ \textbf{div}\ i)]\!]\,\eta\,[l/x]\,(s\,[0/l])$ is contained in $\eta\,y\,R^{cmd}R^{iexp}R^{sto} \subseteq R^{int} = \delta^2 D^{int}$, i.e. $d_1 = d_2$, and this proves the equality. □

Example 7.5 is essentially taken from [22]. It shows that "there is more to representational abstraction than preservation of invariants" [22]. Again, the local variable x may be considered as the instance variable of a counter object, which now has two methods, namely one which increases the counter and one which reads the counter. M_1 and M_2 use two different internal representations of such a counter object, and the observational congruence between M_1 and M_2 shows that the client y cannot distinguish between these different internal representations. This is an instance of *representation independence* [17,22].

Example 7.6. $[\![\textbf{new}\,x\ \textbf{in}\ x := 1;\ y^{iexp \to iexp}\,(!x)\ \textbf{end}]\!] \;=\; [\![y^{iexp \to iexp}\ 1]\!]$

Proof. Let $\eta \in Env$, $s \in Stores$ and let L and l as usual. Let $R \in DEF_2^{\{l\}}$ be defined by $R^{sto} = \{\bot\}^2 \cup \{\vec{s} \in Stores^2 \mid s_1\,l = 1 \wedge s_1 =_{Loc\setminus\{l\}} s_2\}$. Then $(s\,[1/l], s) \in R^{sto}$

[6] Note that ALG is a full fledged call-by-name λ-calculus, hence – in contrast to ALGOL 60 – there is no need to introduce a name for the procedure $x := !x + 1$ and – in contrast to the call-by-value language ML – there is no need to explicitly delay evaluation of $x := !x + 1$ with the aid of a λ-abstraction.

and $([\![!x]\!]\eta[l/x], [\![1]\!]\eta) \in R^{iexp}$. Hence the pair $(d_1,d_2) = ([\![y(!x)]\!]\eta[l/x](s[1/l]), [\![y\,1]\!]\eta s)$ is contained in $\eta\, y R^{iexp} R^{sto} \in R^{int} = \delta^2 D^{int}$, i.e. $d_1 = d_2$ and this easily implies the equality. □

Example 7.6 was presented in [8]. Note that the simpler term $(x := 1;\ y(!x))$ is *not* observationally congruent to $y\,1$, because $!x$ is a name parameter and the function procedure y may have a temporary side effect on the global variable x before it uses its parameter. Hence it is indeed necessary for the example that x is a *local* variable.

Example 7.7. $[\![y^{\sigma\to cmd} z^{\sigma}]\!] = [\![\textbf{new}\, x \textbf{ in } y(x := !x + 1;\ z) \textbf{ end}]\!]$

Proof. We only consider $\sigma = cmd$. Let $\eta \in Env$, $s \in Stores$ and $L \in W$ with $\eta\, y \in [\![cmd \to cmd]\!]_L$ and $\eta\, z \in [\![cmd]\!]_L$, and let $l \in Loc \setminus L$. We choose $R \in DEF_2^{\{l\}}$ with $R^{sto} = \{\bot\}^2 \cup \{\vec{s} \in Stores^2 \mid s_1 =_{Loc\setminus\{l\}} s_2\}$. Then we obtain $(s, s[0/l]) \in R^{sto}$ and $(\eta z, [\![x := !x+1;\ z]\!]\eta[l/x]) \in R^{cmd}$, because $s_1 =_{Loc\setminus\{l\}} s_2$ implies $\eta z s_1 =_{Loc\setminus\{l\}} \eta z s_2 =_{Loc\setminus\{l\}} [\![x := !x+1;\ z]\!]\eta[l/x]s_2$ by Theorem 5.3 (iii′). Hence the pair $(t_1,t_2) = ([\![yz]\!]\eta s, [\![y(x := !x+1;\ z)]\!]\eta[l/x](s[0/l]))$ is contained in $\eta\, y R^{cmd} R^{sto} \subseteq R^{cmd} R^{sto} \subseteq R^{sto}$, i.e. $t_1 =_{Loc\setminus\{l\}} t_2$. This implies $t_1 = t_2[s\,l/l]$ because $t_1 l = s\, l$ by Theorem 5.3 (iii), and thus the equality is proved. □

The intuition for Example 7.7 is that the local variable x counts the procedure calls of z during the computation of yz (occasionally the counter may snap back, namely when z is called inside an integer expression). The equivalence shows that adding such a counter has no influence on the procedure call yz. Example 7.7 will play a role in the full abstraction proof.

8. First and second order domains

As a preparation for the full abstraction proof in Section 9 we will now prove some further properties of our denotational semantics, in particular we will take a closer look at types of order $\leqslant 2$. The following theorem presents an alternative description of the domains $[\![\sigma]\!]_L$ with $ord(\sigma) = 1$. This description is more concrete than the original one in that it no longer refers to the signature Σ.

Theorem 8.1 (Concrete description of the first order domains). *Let $L \in W$. Then*

(i) $[\![iexp]\!]_L = \{f\colon D^{sto} \to D^{int} \mid f\bot = \bot \wedge \forall s,s' \in Stores.\, s =_L s' \Rightarrow fs = fs'\}$

(ii) $[\![cmd]\!]_L = \{f\colon D^{sto} \to D^{sto} \mid f\bot = \bot$
$\wedge\ \forall s \in Stores.\, fs \neq \bot \Rightarrow fs =_{Loc\setminus L} s$
$\wedge\ \forall s,s' \in Stores.\, s =_L s' \Rightarrow fs =_L fs'\}$

(iii) $[\![loc^m \to iexp]\!]_L = \{f\colon Loc \to \ldots \to Loc \to [\![iexp]\!] \mid$
$\forall l_1,\ldots,l_m \in Loc,\ s \in Stores,\ \varphi \in Fix(L).$
$f l_1 \ldots l_m \in [\![iexp]\!]_{L\cup\{l_1,\ldots,l_m\}}$
$\wedge\ f(\varphi l_1)\ldots(\varphi l_m)s = f l_1 \ldots l_m(s \circ \varphi)\}$

(iv) $[\![loc^m \to cmd]\!]_L = \{f\colon Loc \to \ldots \to Loc \to [\![cmd]\!] \mid$
$\forall l_1,\ldots,l_m \in Loc,\ s \in Stores,\ \varphi \in Fix(L).$
$f l_1 \ldots l_m \in [\![cmd]\!]_{L\cup\{l_1,\ldots,l_m\}}$
$\wedge\, f(\varphi l_1)\ldots(\varphi l_m)s = (f l_1 \ldots l_m(s \circ \varphi)) \circ \varphi^{-1}\,\}$

Proof. In each case only '$\supseteq$' must be proved, because '$\subseteq$' already follows from Theorem 5.3. We consider (ii) and (iv), the proofs for (i) and (iii) are similar.

(ii) Let f be in the set on the right-hand side. Per definition of $[\![cmd]\!]_L$ we must only show that f preserves all $R \in \Sigma^L$. Hence let $R \in \Sigma^L_n$ and $\vec{s} \in R^{sto}$. Because of the first and third condition for f there is some $M \in c\text{-}\mathrm{ALG}^{cmd}_L$ (consisting of tests and assignments over location constants in L) such that $s =_L s_i$ implies $[\![M]\!]s =_L f s_i$ for all $s \in Stores$ and $i \in \{1,\ldots,n\}$. This means in particular $[\![M]\!]s_i =_L f s_i$ for $i = 1,\ldots,n$, hence either $[\![M]\!]s_i = \bot = f s_i$ or $[\![M]\!]s_i =_{Loc\setminus L} s_i =_{Loc\setminus L} f s_i$ by the second condition for f, and this implies again $[\![M]\!]s_i = f s_i$. Thus we have proved $f\vec{s} = [\![M]\!]\vec{s} \in [\![M]\!]R^{sto} \subseteq R^{sto}$.

(iv) Let f be in the set on the right-hand side. Again it is sufficient to show that f preserves all $R \in \Sigma^L$ (local continuity is not an issue, because $[\![loc]\!]$ is ordered discretely). Hence let $R \in \Sigma^L_n$, $\vec{l}_1,\ldots,\vec{l}_m \in R^{loc}$ and $\vec{s} \in R^{sto}$. We define a relation $\sim$ on Loc^m by

$$(l_1,\ldots,l_m) \sim (l'_1,\ldots,l'_m) \Leftrightarrow \exists\, \varphi \in Fix(L).\forall i \in \{1,\ldots,m\}.\, \varphi\, l_i = l'_i$$

Obviously, $\sim$ is an equivalence relation on Loc^m (because $Fix(L)$ is a group with respect to function composition), and the equivalence class of a tuple $(l_1,\ldots,l_m)$ is uniquely determined by the two sets $\{(i,j) \in \{1,\ldots,m\}^2 \mid l_i = l_j\}$ and $\{(i,l) \in \{1,\ldots,m\} \times L \mid l_i = l\}$. Hence, with the aid of the term $EQ =_{def} \lambda x,x'.\, x := !x' + 1;\ !x = !x' \in \mathrm{ALG}^{loc^2\to iexp}_{\emptyset}$ which tests the equality of locations, it is easy to construct a term $CLASS \in c\text{-}\mathrm{ALG}^{loc^m\to iexp}_L$ which determines the $\sim$-equivalence class of a tuple in Loc^m, i.e.

$$[\![CLASS]\!]\, l_1 \ldots l_m s = [\![CLASS]\!]\, l'_1 \ldots l'_m s \Leftrightarrow (l_1,\ldots,l_m) \sim (l'_1,\ldots,l'_m)$$

for all $l_1,\ldots,l_m,l'_1,\ldots,l'_m \in Loc$ and $s \in Stores$.

Now let eq be one of the $\sim$-equivalence classes. We will first construct a term $N_{eq} \in c\text{-}\mathrm{ALG}^{loc^m\to cmd}_L$ such that

$$[\![N_{eq}]\!]\, l_{1i} \ldots l_{mi} s_i = f\, l_{1i} \ldots l_{mi} s_i \quad \text{whenever } (l_{1i},\ldots,l_{mi}) \in eq$$

Without loss of generality we may assume $eq = \{(l_{11},\ldots,l_{m1}),\ldots,(l_{1k},\ldots,l_{mk})\}$ for some $k \leqslant n$. Then there are functions $\varphi_i \in Fix(L)$ such that $(l_{1i},\ldots,l_{mi}) = (\varphi_i l_{11},\ldots,\varphi_i l_{m1})$ for $i = 1,\ldots,k$. Since $f l_{11} \ldots l_{m1} \in [\![cmd]\!]_{L\cup\{l_{11},\ldots,l_{m1}\}}$, we can first choose some $M \in c\text{-}\mathrm{ALG}^{cmd}_{L\cup\{l_{11},\ldots,l_{m1}\}}$ as in the proof of (ii), such that $[\![M]\!]$ and $f l_{11} \ldots l_{m1}$ coincide on the finitely many stores $s_i \circ \varphi_i$, $i = 1,\ldots,k$, and from M we can easily construct a term $N_{eq} \in c\text{-}\mathrm{ALG}_L^{loc^m \to cmd}$ with $[\![N_{eq}]\!]\, l_{11} \ldots l_{m1} = [\![M]\!]$. Thus we obtain indeed

$$\begin{aligned}[\![N_{eq}]\!]\, l_{1i} \ldots l_{mi} s_i &= ([\![N_{eq}]\!]\, l_{11} \ldots l_{m1}(s_i \circ \varphi_i)) \circ \varphi_i^{-1}\\ &= ([\![M]\!]\,(s_i \circ \varphi_i)) \circ \varphi_i^{-1}\\ &= (f\, l_{11} \ldots l_{m1}(s_i \circ \varphi_i)) \circ \varphi_i^{-1}\\ &= f\, l_{1i} \ldots l_{mi} s_i\end{aligned}$$

for $i = 1, \ldots, k$. Finally, we can use the term *CLASS* for branching between the various N_{eq} and thus obtain a term $N \in c\text{-}\mathrm{ALG}_L^{loc^m \to cmd}$ such that $[\![N]\!]\, l_{1i} \ldots l_{mi} s_i = f\, l_{1i} \ldots l_{mi} s_i$ for *all* $i \in \{1, \ldots, n\}$. This means $f\,\vec{l}\,\vec{s} = [\![N]\!]\,\vec{l}\,\vec{s} \in R^{sto}$. □

For second order types we do not have such a concrete description of all the domains $[\![\sigma]\!]_L$ as for the first order types. Instead, the following proposition presents only one particular example, namely the domain $[\![cmd \to cmd]\!]_\emptyset$. It is meant as a warm-up exercise for Section 9, because it anticipates certain techniques which will reappear in the full abstraction proof in a (much) more complicated form.

Proposition 8.2. *For* $m \geqslant n \geqslant 0$ *let* $f_{m,n} = [\![\lambda z^{cmd}.\,\mathbf{if}\ z^m; 0\ \mathbf{then}\ z^n]\!]$ *where* $z^0 \equiv skip$ *and* $z^k \equiv \underbrace{z; \ldots ; z}_{k}$ *if* $k > 0$. *Then* $[\![cmd \to cmd]\!]_\emptyset = \{\bot\} \cup \{f_{m,n} \mid m \geqslant n \geqslant 0\}$.

Proof. The proof is a refinement of Plotkin's argument for the Church numerals [27]. We choose some arbitrary $l \in Loc$ and define

$$\begin{aligned}&inc_{<i} = [\![\mathbf{if}\ !\,l < i\ \mathbf{then}\ l := !\,l + 1\ \mathbf{else}\ \Omega]\!] \quad \text{for every } i \in \mathbb{N}\\ &inc_{<\infty} = [\![l := !\,l + 1]\!]\end{aligned}$$

Then $inc_{<0} \sqsubset inc_{<1} \sqsubset \ldots$ is an ω-chain in $[\![cmd]\!]_{\{l\}}$ which has $inc_{<\infty}$ as its least upper bound. Now let $f \in [\![cmd \to cmd]\!]_\emptyset$. Define

$$\begin{aligned}m &= card\,\{i \in \mathbb{N} \mid f\, inc_{<i}\, s_0 = \bot\} && \in \mathbb{N} \cup \{\infty\}\\ n &= f\, inc_{<\infty}\, s_0\, l && \in \mathbb{Z}_\bot\end{aligned}$$

where s_0 is some arbitrary store with $s_0\, l = 0$. Note that m and n are independent of the particular choice of s_0 because $f\, inc_{<i} \in [\![cmd]\!]_{\{l\}}$ for every $i \in \mathbb{N} \cup \{\infty\}$. Moreover, $n \in \mathbb{Z}$ whenever $m \in \mathbb{N}$ because of the monotonicity of f. We will show that

$$\begin{aligned}&f = \bot && \text{if } m = \infty\\ &m \geqslant n \geqslant 0 \wedge f = f_{m,n} && \text{if } m \in \mathbb{N}\end{aligned}$$

To this end let $g \in [\![cmd]\!]$ and $s \in Stores$. Then there is some $L \in W$ with $l \in L$ and $g \in [\![cmd]\!]_L$. The intuition is that the computation of $f g s$ can be simulated by the computation of $f\, inc_{<k}\,(s\,[0/l])$ for some appropriate $k \in \mathbb{N} \cup \{\infty\}$, and that this simulation can be expressed by one of our logical relations. We choose $k = \sup\{i \in$

$\mathbb{N} \mid g^i s \neq \bot\}$ and we define $R \in DEF_2^L$ by

$$R^{sto} = \{\bot\}^2 \cup \{\vec{s} \in Stores^2 \mid \exists i \in \mathbb{N}.\, i \leqslant k \,\wedge\, s_1 l = i \,\wedge\, s_2 =_L g^i s \,\wedge\, s_1 =_{Loc \setminus L} s_2\}$$

Then $(s[0/l], s) = (s[0/l], g^0 s) \in R^{sto}$ and it is easy to see that $(inc_{<k}, g) \in R^{cmd}$. This implies $(f\, inc_{<k}(s[0/l]), f g s) \in f R^{cmd} R^{sto} \subseteq R^{sto}$. If $m = \infty$, then we have $f\, inc_{<k}(s[0/l]) = \bot$, hence also $f g s = \bot$. Thus we have proved $f = \bot$ in this case. If $m \in \mathbb{N}$, then

$$\begin{aligned} fgs = \bot &\Leftrightarrow f\, inc_{<k}(s[0/l]) = \bot \\ &\Leftrightarrow k < m \quad \text{per definition of } m \\ &\Leftrightarrow g^m s = \bot \quad \text{per definition of } k \end{aligned}$$

and

$$\begin{aligned} fgs \neq \bot &\Rightarrow \exists i \in \mathbb{N}.\, f\, inc_{<k}(s[0/l])\, l = i \,\wedge\, fgs =_L g^i s \\ &\Rightarrow n \in \mathbb{N} \,\wedge\, fgs =_L g^n s \quad \text{because } i \text{ can only be } n \\ &\Rightarrow n \in \mathbb{N} \,\wedge\, fgs = g^n s \quad \text{because } fg \text{ and } g^n \text{ are in } [\![cmd]\!]_L \end{aligned}$$

Thus we have proved

$$fgs = \begin{cases} \bot & \text{if } g^m s = \bot \\ g^n s \neq \bot & \text{if } g^m s \neq \bot \end{cases}$$

for all $g \in [\![cmd]\!]$ and $s \in Stores$. This implies $m \geqslant n$ and $f = f_{m,n}$. □

Note that by Proposition 8.2 the domain $[\![cmd \to cmd]\!]_\emptyset$ consists of infinitely many descending chains $f_{n,n} \sqsupset f_{n+1,n} \sqsupset \ldots$ which only meet in the bottom element[7]; the maximal elements of this domain are the 'Church numerals' $f_{n,n} = [\![\lambda z.z^n]\!]$. If we define $p_i : [\![cmd \to cmd]\!]_\emptyset \to [\![cmd \to cmd]\!]_\emptyset$ for all $i \in \mathbb{N}$ by

$$p_i f = \begin{cases} f & \text{if } f = f_{m,n} \text{ for some } m, n \leqslant i \\ \bot & \text{otherwise} \end{cases}$$

then $p_0 \sqsubseteq p_1 \sqsubseteq \ldots$ is an ω-chain of idempotent deflations [1] which have the identity as their least upper bound, i.e. $[\![cmd \to cmd]\!]_\emptyset$ is an SFP object [25]. But a different property of the functions p_i is more interesting for us: With the notation from the above proof we can reformulate the definition of p_i as

$$p_i f = \begin{cases} \bot & \text{if } f\, inc_{<i}\, s_0\, l = \bot \\ f_{m,n} & \text{if } f\, inc_{<i}\, s_0\, l \neq \bot, \\ & \quad m = card\,\{j < i \mid f\, inc_{<j}\, s_0\, l = \bot\} \text{ and } n = f\, inc_{<\infty}\, s_0\, l \end{cases}$$

[7] O'Hearn and Tennent, who obtained the same result for their model in [23], describe this domain as the smash product $\mathbb{N}_\bot \otimes VNat^{op}$ where $VNat^{op}$ is the partial order $(N \cup \{\infty\}, \geqslant)$.

This shows that $p_i f$ is uniquely determined by the finitely many values $f\, inc_{<j}\, s_0\, l$ with $j \in \{0,\ldots,i,\infty\}$. Such functions will be called 'finitely determined' in Section 9, and sequences of finitely determined functions which have the identity as their least upper bound will play a prominent role in the full abstraction proof.

We conclude this section with a technical lemma.

Lemma 8.3. *Let $\varphi : Loc \to Loc$ be a finite permutation and let $R_\varphi \in DEF_2^{Supp(\varphi)}$ as in Section 5. Then there is a term $SWAP_\varphi^\sigma \in c\text{-}\mathrm{ALG}_{Supp(\varphi)}^{\sigma\to\sigma}$ for every procedure type σ, such that R_φ^σ is the graph of $[\![SWAP_\varphi^\sigma]\!]$ and $[\![SWAP_\varphi^\sigma]\!] \circ [\![SWAP_{\varphi^{-1}}^\sigma]\!] = \mathsf{id}_\sigma$.*

Proof. It is easy to construct $SWAP_\varphi \in c\text{-}\mathrm{ALG}_{Supp(\varphi)}^{cmd}$ such that $[\![SWAP_\varphi]\!]\, s\, l = s\,(\varphi\, l)$ for all $s \in Stores$ and $l \in Loc$, i.e. such that R_φ^{sto} is the graph of $[\![SWAP_\varphi]\!]$. By induction on σ we then define $SWAP_\varphi^\sigma \in c\text{-}\mathrm{ALG}_{Supp(\varphi)}^{\sigma\to\sigma}$ by

$$SWAP_\varphi^{iexp} \equiv \lambda\, y^{iexp}.\ SWAP_{\varphi^{-1}};\ y$$

$$SWAP_\varphi^{cmd} \equiv \lambda\, y^{cmd}.\ SWAP_{\varphi^{-1}};\ y;\ SWAP_\varphi$$

$$\begin{aligned}SWAP_\varphi^{loc\to\sigma} \equiv\ & \lambda y^{loc\to\sigma}.\,\lambda x^{loc}.\\ & \textbf{if } EQ\,x\,l_1' \textbf{ then } SWAP_\varphi^\sigma(y\,l_1) \textbf{ else}\\ & \vdots\\ & \textbf{if } EQ\,x\,l_n' \textbf{ then } SWAP_\varphi^\sigma(y\,l_n) \textbf{ else } SWAP_\varphi^\sigma(y\,x)\end{aligned}$$

where $EQ =_{def} \lambda x,x'.x := !x' + 1;\ !x = !x'$ tests the equality on locations, $\{l_1,\ldots,l_n\} = Supp(\varphi)$ and $l_i' = \varphi\, l_i$ for $i = 1,\ldots,n$

$$SWAP_\varphi^{\sigma\to\sigma'} \equiv \lambda\, y^{\sigma\to\sigma'}.\,\lambda z^\sigma.SWAP_\varphi^{\sigma'}(y\,(SWAP_{\varphi^{-1}}^\sigma z))$$

A straightforward induction on σ shows that every $[\![SWAP_\varphi^\sigma]\!]$ has the desired properties. $\square$

9. Full abstraction

We will now present our full abstraction proof. The overall structure of the proof is the same as for PCF in [32]: In the first part we show that for every function $f \in [\![\sigma]\!]_L$ with $ord(\sigma) \leqslant 2$ and every finite set B of argument tuples for f there is a term $M \in c\text{-}\mathrm{ALG}_L^\sigma$ such that $[\![M]\!]$ and f coincide on B. As in [32] we will prove this result by using "logical relations which have large arity and are reminiscent of value tables" [6]. As a preparation we prove a technical lemma which allows us to 'fill up' a ground relation with a cofinite part of the diagonal $\delta^n Loc$.

Definition 9.1. Let R be an n-ary ground relation with $R^{sto}(\delta^n(Loc \setminus L)) \subseteq R^{int}$. Then the *$L$-closure* of R is defined to be the ground relation S with

- $S^{int} = R^{int}$,
- $S^{loc} = R^{loc} \cup \delta^n(Loc \setminus L)$
- $S^{sto} = \{\vec{s} \in (D^{sto})^n \mid \exists \vec{t} \in R^{sto}. \vec{s} =_L \vec{t} \wedge \vec{s}(\delta^n(Loc \setminus L)) \subseteq R^{int}\}$

Note that R is 'contained' in its L-closure S, i.e. $R^\gamma \subseteq S^\gamma$ for every $\gamma \in \Gamma$.

Lemma 9.2. *Let R be an n-ary ground relation with $R^{sto}(\delta^n(Loc \setminus L)) \subseteq R^{int}$ and $R^{loc} \subseteq L^n$. Then every function $f \in AUX$, which preserves R, also preserves the L-closure of R.*

Proof. Let $f \in AUX$ preserve R, and let S denote the L-closure of R. We show that f preserves S.

Case 1: $f = Const_m$

If $\vec{s} \in S^{sto}$ and $\vec{t} \in R^{sto}$ with $\vec{s} =_L \vec{t}$, then $Const_m\, \vec{s} = Const_m\, \vec{t} \in R^{int} = S^{int}$.

Case 2: $f \in \{Succ, Pred, Cond_{int}, Pcond\}$

Obvious, because $S^{int} = R^{int}$.

Case 3: $f = Cond_{sto}$

First note that $Cond_{sto}\, d\, s\, t\, l = Cond_{int}\, d\, (s\, l)(t\, l)$ for all $d \in D^{int}$, $s, t \in D^{sto}$ and $l \in Loc$. Now let $\vec{d} \in S^{int} = R^{int}$, $\vec{s}, \vec{t} \in S^{sto}$ and $\vec{u}, \vec{v} \in R^{sto}$ with $\vec{s} =_L \vec{u}$ and $\vec{t} =_L \vec{v}$. Then $Cond_{sto}\, \vec{d}\, \vec{s}\, \vec{t} =_L Cond_{sto}\, \vec{d}\, \vec{u}\, \vec{v} \in R^{sto}$ because $Cond_{int}\, \vec{d}\, (\vec{s}\, l)(\vec{t}\, l) = Cond_{int}\, \vec{d}\, (\vec{u}\, l)(\vec{v}\, l)$ for all $l \in L$, and $Cond_{sto}\, \vec{d}\, \vec{s}\, \vec{t}\, l = Cond_{int}\, \vec{d}\, (\vec{s}\, l)(\vec{t}\, l) \in Cond_{int} R^{int} R^{int} R^{int} \subseteq R^{int}$ for all $l \in Loc \setminus L$. This proves $Cond_{sto}\, \vec{d}\, \vec{s}\, \vec{t} \in S^{sto}$.

Case 4: $f = Cont$

Let $\vec{l} \in S^{loc}$, $\vec{s} \in S^{sto}$ and $\vec{t} \in R^{sto}$ with $\vec{s} =_L \vec{t}$. If $\vec{l} \in R^{loc} \subseteq L^n$, then $Cont\, \vec{l}\, \vec{s} = \vec{s}\, \vec{l} = \vec{t}\, \vec{l} = Cont\, \vec{l}\, \vec{t} \in R^{int} = S^{int}$. If $\vec{l} \in \delta^n(Loc \setminus L)$, then $Cont\, \vec{l}\, \vec{s} = \vec{s}\, \vec{l} \in R^{int} = S^{int}$ per definition of S^{sto}.

Case 5: $f = Asgn$

Let $\vec{l} \in S^{loc}$, $\vec{d} \in S^{int}$, $\vec{s} \in S^{sto}$, $\vec{t} \in R^{sto}$ with $\vec{s} =_L \vec{t}$ and let $\vec{u} = Asgn\, \vec{l}\, \vec{d}\, \vec{s}$. If $\vec{l} \in R^{loc} \subseteq L^n$, then $\vec{u} =_L Asgn\, \vec{l}\, \vec{d}\, \vec{t} \in R^{sto}$, because $Asgn\, \vec{l}\, \vec{d} \in ([\![cmd]\!]_L)^n$, and $\vec{u}\, l = \vec{s}\, l \in R^{int}$ for every $l \in Loc \setminus L$, hence $\vec{u} \in S^{sto}$. If $\vec{l} = (l, \ldots, l) \in \delta^n(Loc \setminus L)$, then $\vec{u} =_L \vec{s} =_L \vec{t} \in R^{sto}$, $\vec{u}\, l = \vec{d} \in S^{int} = R^{int}$ and $\vec{u}\, l' = \vec{s}\, l' \in R^{int}$ for all $l' \in Loc \setminus (L \cup \{l\})$, hence again $\vec{u} \in S^{sto}$. □

Notation. If $f \in [\![\tau_1 \to \cdots \to \tau_k \to sto \Rightarrow \gamma]\!]$, then we let f^d denote the *completely decurried version* of f, i.e.

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

Theorem 9.3 (Finite coincidence with a definable function). *Let $L \in W$ and $\sigma = \tau_1 \to \cdots \to \tau_k \to \theta$ ($k \geqslant 0$) with $ord(\sigma) \leqslant 2$. Let $f \in [\![\sigma]\!]_L$ and let $B \subseteq [\![\tau_1]\!] \times \cdots \times [\![\tau_k]\!] \times D^{sto}$ be finite. Then there is some $M \in c\text{-}\mathrm{ALG}_L^\sigma$ with $[\![M]\!]^d =_B f^d$.*

Proof. Let $B = \{(d_{11},\dots,d_{k1},s_1),\ \dots,\ (d_{1n},\dots,d_{kn},s_n)\}$, let $\vec{d}_j = (d_{j1},\dots,d_{jn})$ for $j = 1,\dots,k$, $\vec{s} = (s_1,\dots,s_n)$ and let R be the n-ary ground relation with

$$R^{loc} = \{\vec{d}_j \mid \tau_j = loc\} \cup \delta^n L$$

$$R^{\gamma} = \{[\![M]\!]\,\vec{d}_1\dots\vec{d}_k\,\vec{s} \mid M \in c\text{-}\mathrm{ALG}_L^{\tau_1\to\dots\to\tau_k\to sto\Rightarrow\gamma}\} \quad \text{for } \gamma = int, sto$$

Then we must prove that $f\vec{d}_1\dots\vec{d}_k\vec{s} \in R^{\gamma}$ (if $\theta = sto \Rightarrow \gamma$). As a first step we show that every $g \in AUX$ preserves R.

Case 1: $g = Const_m$

Let $\vec{t} \in R^{sto}$, i.e. $\vec{t} = [\![M]\!]\,\vec{d}_1\dots\vec{d}_k\,\vec{s}$ for some $M \in c\text{-}\mathrm{ALG}_L^{\tau_1\to\dots\to\tau_k\to cmd}$. Then $Const_m\vec{t} = [\![\lambda x_1,\dots,x_k.Mx_1\dots x_k;\ m]\!]\,\vec{d}_1\dots\vec{d}_k\,\vec{s} \in R^{int}$.

Case 2: $g = Succ$ (similarly for $Pred, Cond_\gamma$ and $Pcond$)

Let $\vec{e} \in R^{int}$, i.e. $\vec{e} = [\![M]\!]\,\vec{d}_1\dots\vec{d}_k\,\vec{s}$ for some $M \in c\text{-}\mathrm{ALG}_L^{\tau_1\to\dots\to\tau_k\to iexp}$. Then $Succ\,\vec{e} = [\![\lambda x_1,\dots,x_k.succ\,(Mx_1\dots x_k)]\!]\,\vec{d}_1\dots\vec{d}_k\,\vec{s} \in R^{int}$.

Case 3: $g = Cont$

Let $\vec{l} \in R^{loc}$ and $\vec{t} = [\![M]\!]\,\vec{d}_1\dots\vec{d}_k\,\vec{s} \in R^{sto}$. If $\vec{l} = \vec{d}_j$ with $\tau_j = loc$, then let $P \equiv \lambda x_1,\dots,x_k.Mx_1\dots x_k;\ !x_j$, and if $\vec{l} = (l,\dots,l) \in \delta^n L$, then let $P \equiv \lambda x_1,\dots,x_k.Mx_1\dots x_k;\ !\,l$. In both cases $Cont\,\vec{l}\,\vec{t} = [\![P]\!]\,\vec{d}_1\dots\vec{d}_k\,\vec{s} \in R^{int}$.

Case 4: $g = Asgn$

Let $\vec{l} \in R^{loc}$, $\vec{e} = [\![M]\!]\,\vec{d}_1\dots\vec{d}_k\,\vec{s} \in R^{int}$ and $\vec{t} = [\![N]\!]\,\vec{d}_1\dots\vec{d}_k\,\vec{s} \in R^{sto}$. If $\vec{l} = \vec{d}_j$ with $\tau_j = loc$, then $Asgn\,\vec{l}\,\vec{e}\,\vec{t} = [\![P]\!]\,\vec{d}_1\dots\vec{d}_k\,\vec{s}$, where

$$P \equiv \lambda x_1,\dots,x_k.\mathbf{new}\,x\ \mathbf{in}\ x := Mx_1\dots x_k;\ Nx_1\dots x_k;\ x_j := !x\ \mathbf{end}$$

Intuitively, P works as follows: First, $Mx_1\dots x_k$ is evaluated and the result $\vec{e}$ is stored into the local variable x. After evaluation of $Mx_1\dots x_k$ the computation snaps back to $\vec{s}$, and then $\vec{t}$ is computed by evaluating $Nx_1\dots x_k$. Finally, $\vec{t}$ is updated to $\vec{t}[\vec{e}/\vec{l}]$ by the assignment $x_j := !x$. The precise argumentation is as follows.

$$
\begin{aligned}
&[\![P]\!]\,\vec{d}_1\dots\vec{d}_k\,\vec{s}\\
&= ([\![x := Mx_1\dots x_k;\ Nx_1\dots x_k;\ x_j := !x]\!]\,\vec{\eta}\,[l/x]\,(\vec{s}\,[0/l]))\,[\vec{s}\,l/l]\\
&\qquad \text{where } l \text{ is some new location and } \vec{\eta} \in Env^n \text{ with } \vec{\eta}x_i = \vec{d}_i \text{ for } i = 1,\dots k\\
&= ([\![Nx_1\dots x_k;\ x_j := !x]\!]\,\vec{\eta}\,[l/x]\,(\vec{s}\,[\vec{e}/l]))\,[\vec{s}\,l/l]\\
&\qquad \text{because } l \text{ is new and hence } [\![M]\!]\,\vec{d}_1\dots\vec{d}_k\,(\vec{s}\,[0/l]) = [\![M]\!]\,\vec{d}_1\dots\vec{d}_k\,\vec{s} = \vec{e}\\
&= ([\![x_j := !x]\!]\,\vec{\eta}\,[l/x]\,(\vec{t}\,[\vec{e}/l]))\,[\vec{s}\,l/l]\\
&\qquad \text{because } l \text{ is new and hence } [\![N]\!]\,\vec{d}_1\dots\vec{d}_k\,(\vec{s}\,[\vec{e}/l]) = \vec{t}\,[\vec{e}/l]\\
&= \vec{t}\,[\vec{e}/l]\,[\vec{e}/\vec{l}]\,[\vec{s}\,l/l]\\
&= \vec{t}\,[\vec{e}/\vec{l}] \quad \text{because } l \text{ is new and hence } \vec{s}\,l = \vec{t}\,l\\
&= Asgn\,\vec{l}\,\vec{e}\,\vec{t}
\end{aligned}
$$

If $\vec{l} = (l,\dots,l) \in \delta^n L$, then we replace the assignment $x_j := !x$ in P by $l := !x$. Thus we obtain again $Asgn\,\vec{l}\,\vec{e}\,\vec{t} = [\![P]\!]\,\vec{d}_1\dots\vec{d}_k\,\vec{s}$, i.e. $Asgn\,\vec{l}\,\vec{e}\,\vec{t} \in R^{sto}$ in both cases.

So far we have shown that every $g \in AUX$ preserves R. Now let $L' \supseteq L$ be such that $\vec{d}_j \in ([\![\tau_j]\!]_{L'})^n$ for $j = 1,\dots,k$ and $\vec{s} \in (Stores_{L'})^n$. Then $R^{sto}(\delta^n(Loc \setminus L')) = Const_0 R^{sto} \subseteq R^{int}$, hence we can define the L'-closure S of R. By Lemma 9.2, every $g \in AUX$ preserves S, moreover $\delta^n(Loc \setminus (L' \setminus L)) \subseteq S^{loc}$ because $\delta^n L \subseteq R^{loc}$, and finally $(\bot,\dots,\bot) \in S^{sto}$ because $(\bot,\dots,\bot) = [\![\lambda x_1,\dots,x_k.\Omega]\!]\vec{d}_1\dots\vec{d}_k\vec{s} \in R^{sto}$. Altogether this proves $S \in \Sigma^L$ and hence f preserves S.

If we can now show that $\vec{d}_j \in S^{\tau_j}$ for $j = 1,\dots,k$, then we obtain $f\vec{d}_1\dots\vec{d}_k\vec{s} \in fS^{\tau_1}\dots S^{\tau_k}R^{sto} \subseteq fS^{\tau_1}\dots S^{\tau_k}S^{sto} \subseteq S^{\gamma}$. For $\gamma = int$ this already concludes the proof, because $S^{int} = R^{int}$. For $\gamma = sto$ we first obtain $f\vec{d}_1\dots\vec{d}_k\vec{s} =_{L'} \vec{t}$ for some $\vec{t} \in R^{sto}$, and then $f\vec{d}_1\dots\vec{d}_k\vec{s} =_{Loc\setminus L'} \vec{s} =_{Loc\setminus L'} \vec{t}$ implies $f\vec{d}_1\dots\vec{d}_k\vec{s} = \vec{t} \in R^{sto}$.

Hence let $j \in \{1,\dots,k\}$. If $\tau_j = loc$, then $\vec{d}_j \in R^{loc} \subseteq S^{loc}$. As τ_j is a type of order $\leqslant 1$, we are left with the case $\tau_j = loc^m \to \theta$ $(m \geqslant 0)$. In order to keep the notation simple, we consider only one particular case, namely $\tau_j = loc \to cmd$:

Let $\vec{l} \in S^{loc}, \vec{t} \in S^{sto}, \vec{u} \in R^{sto}$ with $\vec{t} =_{L'} \vec{u}$ and let $M \in c\text{-}\mathrm{ALG}_L^{\tau_1\to\dots\to\tau_k\to cmd}$ with $\vec{u} = [\![M]\!]\vec{d}_1\dots\vec{d}_k\vec{s}$. If $\vec{l} = \vec{d}_i$ with $\tau_i = loc$, then $\vec{d}_j\vec{l}\vec{t} =_{L'} \vec{d}_j\vec{l}\vec{u}$ because $\vec{d}_j\vec{l} \in ([\![cmd]\!]_{L'})^n$, hence $\vec{d}_j\vec{l}\vec{t} =_{L'} [\![\lambda x_1,\dots,x_k.Mx_1\dots x_k; x_j x_i]\!]\vec{d}_1\dots\vec{d}_k\vec{s} \in R^{sto}$, and moreover $\vec{d}_j\vec{l}\vec{t}\,l = \vec{t}\,l \in R^{int}$ for all $l \in Loc \setminus L'$. This proves $\vec{d}_j\vec{l}\vec{t} \in S^{sto}$. If $\vec{l} = (l,\dots,l) \in \delta^n L$, then we replace $x_j x_i$ by $x_j\, l$ in the above term and thus obtain again $\vec{d}_j\vec{l}\vec{t} \in S^{sto}$. Hence we are left with the case $\vec{l} = (l,\dots,l) \in \delta^n(Loc \setminus L')$:

As $\vec{t} \in S^{sto}$, we have $\vec{t}\,l \in R^{int}$, i.e. there is some $N \in c\text{-}\mathrm{ALG}_L^{\tau_1\to\dots\to\tau_k\to iexp}$ with $\vec{t}\,l = [\![N]\!]\vec{d}_1\dots\vec{d}_k\vec{s}$. Now we define

$$P \equiv \lambda x_1,\dots,x_k.\,\mathbf{new}\,x\ \mathbf{in}\ x := Nx_1\dots x_k;\ Mx_1\dots x_k;\ x_j x\ \mathbf{end}$$

$$Q \equiv \lambda x_1,\dots,x_k.\,\mathbf{new}\,x\ \mathbf{in}\ x := Nx_1\dots x_k;\ Mx_1\dots x_k;\ x_j x;\ !x\ \mathbf{end}$$

We may assume that our particular location $l \in Loc \setminus L'$ is bound to the local variable x, hence we obtain

$$\begin{aligned}
&[\![P]\!]\vec{d}_1\dots\vec{d}_k\vec{s}\\
&= ([\![x := Nx_1\dots x_k;\ Mx_1\dots x_k;\ x_j x]\!]\vec{\eta}\vec{s})\,[0/l]\\
&\qquad \text{where } \vec{\eta} \in Env^n \text{ with } \vec{\eta}x_i = \vec{d}_i \text{ for } i = 1,\dots k \text{ and } \vec{\eta}x = (l,\dots,l)\\
&=_{L'} [\![x := Nx_1\dots x_k;\ Mx_1\dots x_k;\ x_j x]\!]\vec{\eta}\vec{s}\\
&= [\![Mx_1\dots x_k;\ x_j x]\!]\vec{\eta}(\vec{s}[\vec{t}\,l/l])\\
&= [\![x_j x]\!]\vec{\eta}(\vec{u}[\vec{t}\,l/l])\\
&\qquad \text{because } [\![M]\!]\vec{d}_1\dots\vec{d}_k(\vec{s}[\vec{t}\,l/l]) = \vec{u}[\vec{t}\,l/l] \text{ by Theorem 5.3(iii}')\\
&= \vec{d}_j l(\vec{u}[\vec{t}\,l/l])\\
&=_L \vec{d}_j l\,\vec{t} \quad \text{because } d_j l \in ([\![cmd]\!]_{L'\cup\{l\}})^n \text{ and } \vec{t} =_{L'\cup\{l\}} \vec{u}[\vec{t}\,l/l]
\end{aligned}$$

This shows that $\vec{d}_j l\,\vec{t} =_{L'} [\![P]\!]\vec{d}_1\dots\vec{d}_k\vec{s} \in R^{sto}$ and similarly we can prove $\vec{d}_j l\,\vec{t}\,l = [\![Q]\!]\vec{d}_1\dots\vec{d}_k\vec{s} \in R^{int}$. Moreover, if $l' \in Loc\setminus(L' \cup \{l\})$, then $\vec{d}_j l\,\vec{t}\,l' = \vec{t}\,l' \in R^{int}$ because $\vec{d}_j l \in ([\![cmd]\!]_{L'\cup\{l\}})^n$. Thus we have shown that $\vec{d}_j l\,\vec{t} \in S^{sto}$, and this concludes the proof. □

From Theorem 9.3 we can obtain a sequence of definable functions $[\![M_1]\!], [\![M_2]\!], \ldots$ ($M_i \in c\text{-}\mathrm{ALG}_L^{\sigma}$) which 'converge' to the given function $f \in [\![\sigma]\!]_L$ in the sense that they coincide with f on more and more argument tuples. But for a typical full abstraction proof [26] we must know that f is the *least upper bound* of a sequence (or a directed set) of definable functions. In [32] we succeeded to close the gap between these two kinds of 'convergence' by showing that the meaning of each type is an SFP object, in which the identity is the lub of an ω-chain of *definable* idempotent deflations. But a closer look at the full abstraction proof in [32] reveals that we did not really need to know that these definable functions are idempotent or that they have finite image. Instead we only used the fact that they are 'finitely determined' in the sense of the following definition.

Definition 9.4. Let D_i, E_i ($i = 1,2$) be sets, let $F_i \subseteq (D_i \xrightarrow{t} E_i)$ and $p \in (F_1 \xrightarrow{t} F_2)$.

(1) $B \subseteq D_1$ is called a *determining set* for p, if $f =_B g$ implies $pf = pg$ for all $f, g \in F_1$. p is called *finitely determined* if it has a finite determining set.

(2) $\hat{p} \in (D_2 \xrightarrow{t} D_1)$ is called a *determining function* for p, if $f(\hat{p}d) = g(\hat{p}d)$ implies $pfd = pgd$ for all $f, g \in F_1$ and $d \in D_2$. p is called *locally determined* if it has a determining function.

Finitely determined functions are those which we need for the full abstraction proof, but sometimes it is very difficult to prove 'directly' that a particular function is finitely determined. Hence we use locally determined functions to construct new finitely determined functions from the given ones. This is possible by the following lemma.

Lemma 9.5. *Let D_i, E_i, F_i ($i = 0,1,2$) as before, $p \in (F_1 \xrightarrow{t} F_2)$ and $q \in (F_0 \xrightarrow{t} F_1)$.*

(i) *If p and q are locally determined, then $p \circ q$ is locally determined.*

(ii) *If q is finitely determined, then $p \circ q$ is finitely determined.*

(iii) *If p is finitely determined and q is locally determined, then $p \circ q$ is finitely determined.*

Proof. (i) Let $\hat{p}, \hat{q}$ be determining functions for p and q. Then $f(\hat{q}(\hat{p}d)) = g(\hat{q}(\hat{p}d)) \Rightarrow qf(\hat{p}d) = qg(\hat{p}d) \Rightarrow p(qf)d = p(qg)d$ for all $d \in D_2$ and $f, g \in F_0$. This shows that $\hat{q} \circ \hat{p}$ is a determining function for $p \circ q$.

(ii) Clearly every determining set for q is also a determining set for $p \circ q$.

(iii) Let B be a (finite) determining set for p and let $\hat{q}$ be a determining function for q. Then $f =_{\hat{q}B} g \Rightarrow f \circ \hat{q} =_B g \circ \hat{q} \Rightarrow qf =_B qg \Rightarrow p(qf) = p(qg)$ for all $f, g \in F_0$. This shows that $\hat{q}B$ is a (finite) determining set for $p \circ q$. $\square$

Notation. Let σ, σ' be procedure types and let $p \in ([\![\sigma]\!] \xrightarrow{t} [\![\sigma']\!])$. Then we let p^D denote the corresponding function on the completely decurried types, i.e.

$$p^D\colon [\![\sigma]\!]^d \to [\![\sigma']\!]^d, \qquad p^D f^d = (pf)^d$$

Note that $(p \circ q)^D = p^D \circ q^D$, if $q \in ([\![\sigma]\!] \xrightarrow{t} [\![\sigma']\!])$ and $p \in ([\![\sigma']\!] \xrightarrow{t} [\![\sigma'']\!])$.

Definition 9.6. Let σ, σ' be procedure types and let $L \in W$.

(1) An L-FD-*sequence* on σ is a sequence of terms $P_n \in c\text{-}\mathrm{ALG}_L^{\sigma\to\sigma}$ such that $([\![P_n]\!])_{n\in\mathbb{N}}$ is an ω-chain with $\bigsqcup_{n\in\mathbb{N}}[\![P_n]\!] = \mathsf{id}_\sigma$ and $[\![P_n]\!]^D \mid ([\![\sigma]\!]_L)^d$ is finitely determined for every $n \in \mathbb{N}$. σ is called an L-FD-*type* if there is an L-FD-sequence on σ.

(2) An L-*section-retraction-pair* (or L-SR-*pair*) between σ and σ' is a pair of terms $S \in c\text{-}\mathrm{ALG}_L^{\sigma\to\sigma'}$, $R \in c\text{-}\mathrm{ALG}_L^{\sigma'\to\sigma}$ such that $[\![R]\!] \circ [\![S]\!] = \mathsf{id}_\sigma$ and $[\![S]\!]^D$ is locally determined. σ is called an L-*retract* of σ' (notation: $\sigma \lhd_L \sigma'$) if there is an L-SR-pair between σ and σ'.

Note that L-retracts are closely related to ordinary retracts [1,4]: If (S,R) is an L-SR-pair between the types σ and σ', then $([\![S]\!] \mid [\![\sigma]\!]_L, [\![R]\!] \mid [\![\sigma']\!]_L)$ is an (ordinary) s-r-pair between the dcpo's $[\![\sigma]\!]_L$ and $[\![\sigma']\!]_L$.

Our ultimate goal is to prove that every procedure type σ with $ord(\sigma) \leqslant 2$ is an L-FD-type, whenever $L \neq \emptyset$. But – as mentioned before – it is sometimes very difficult to prove directly that particular functions are finitely determined. This is the point where L-retracts come in:

Theorem 9.7 (L-retracts of L-FD-types). *Every L-retract of an L-FD-type is an L-FD-type.*

Proof. Let $(P_n)_{n\in\mathbb{N}}$ be an L-FD-sequence on σ and let (S,R) be an L-SR-pair between σ' and σ. For every $n \in \mathbb{N}$ let $P'_n \equiv \lambda y^{\sigma'}.R(P_n(Sy)) \in c\text{-}\mathrm{ALG}_L^{\sigma'\to\sigma'}$. By monotonicity of $[\![R]\!]$, the functions $[\![P'_n]\!]$ form an ω-chain, and by its local continuity $\bigsqcup_{n\in\mathbb{N}}[\![P'_n]\!] = \bigsqcup_{n\in\mathbb{N}}[\![R]\!] \circ [\![P_n]\!] \circ [\![S]\!] = [\![R]\!] \circ (\bigsqcup_{n\in\mathbb{N}}[\![P_n]\!]) \circ [\![S]\!] = [\![R]\!] \circ \mathsf{id}_\sigma \circ [\![S]\!] = \mathsf{id}_{\sigma'}$. Moreover, by Lemma 9.5 (ii) and (iii), the functions $[\![P'_n]\!]^D \mid ([\![\sigma']\!]_L)^d = ([\![R]\!]^D) \mid ([\![\sigma]\!]_L)^d \circ [\![P_n]\!]^D \mid ([\![\sigma]\!]_L)^d \circ [\![S]\!]^D \mid ([\![\sigma']\!]_L)^d$ are finitely determined. This proves that $(P'_n)_{n\in\mathbb{N}}$ is an L-FD-sequence on σ'. $\square$

In order to make good use of Theorem 9.7 we will now provide some 'recipes' for obtaining L-retracts. First note that $\lhd_L$ is a preorder for every $L \in W$, because procedure types and L-SR-pairs form a category: The morphisms from σ to σ' are the L-SR-pairs between σ and σ'; the identity morphism on σ is $(\lambda y^\sigma.y, \lambda y^\sigma.y)$; the composition of two morphisms (S,R) from σ to σ' and (S',R') from σ' to σ'' is $(\lambda y^\sigma.S'(Sy), \lambda y^{\sigma''}.R(R'y))$. Note that $([\![S']\!] \circ [\![S]\!])^D = [\![S']\!]^D \circ [\![S]\!]^D$ is indeed locally determined by Lemma 9.5 (i). In order to obtain some more interesting facts about the relations $\lhd_L$ we need the following technical lemma.

Lemma 9.8. (i) *Let $\sigma = \tau_1 \to \cdots \to \tau_k \to \theta$ and $\sigma' = \tau_{m+1} \to \cdots \to \tau_k \to \theta$ for some $m \geqslant 0$, let $p_i \in ([\![\tau]\!] \xrightarrow{l} [\![\tau_i]\!])$ for $i = 1,\ldots,m$ and let $p \in ([\![\sigma]\!] \xrightarrow{l} [\![\tau \to \sigma']\!])$ with*

$$pfd = f(p_1 d)\ldots(p_m d) \quad \textit{for all } f \in [\![\sigma]\!], d \in [\![\tau]\!]$$

Then p^D is locally determined.

(ii) *Let $q \in ([\![\sigma']\!] \xrightarrow{t} [\![\sigma]\!])$ and let $p \in ([\![\tau \to \sigma']\!] \xrightarrow{t} [\![\tau \to \sigma]\!])$ with*

$$pf = q \circ f \quad \textit{for all } f \in [\![\tau \to \sigma']\!]$$

Then p^D is locally determined, if q^D is locally determined.

Proof. (i) For all $d \in [\![\tau]\!]$ and all $(d_{m+1},\dots,d_k,s) \in [\![\tau_{m+1}]\!] \times \dots \times [\![\tau_k]\!] \times D^{sto}$ we have $p^D f^d(d,d_{m+1},\dots,d_k,s) = pfd\,d_{m+1}\dots d_k\,s = f^d(p_1 d,\dots,p_m d,d_{m+1},\dots,d_k,s)$. This shows that $\langle\langle p_1,\dots,p_m\rangle \circ fst, snd\rangle$ is a determining function for p^D.

(ii) For all $d \in [\![\tau]\!]$ and all $(d_1,\cdots,d_k,s) \in [\![\tau_1]\!] \times \cdots \times [\![\tau_k]\!] \times D^{sto}$ we have $p^D f^d(d,d_1,\dots,d_k,s) = pfd\,d_1\dots d_k\,s = q(fd)d_1\dots d_k\,s = q^D(fd)^d(d_1,\dots,d_k,s)$. Now let $\hat{q}$ be a determining function for q^D. Then we obtain

$$\begin{aligned}
&f^d(d,\hat{q}(d_1,\dots,d_k,s)) = g^d(d,\hat{q}(d_1,\dots,d_k,s))\\
&\Rightarrow (fd)^d(\hat{q}(d_1,\dots,d_k,s)) = (gd)^d(\hat{q}(d_1,\dots,d_k,s))\\
&\Rightarrow q^D(fd)^d(d_1,\dots,d_k,s) = q^D(gd)^d(d_1,\dots,d_k,s)\\
&\Rightarrow p^D f^d(d,d_1,\dots,d_k,s) = p^D g^d(d,d_1,\dots,d_k,s)
\end{aligned}$$

This shows that $\langle fst, \hat{q} \circ snd\rangle$ is a determining function for p^D. □

Lemma 9.9. *Let $L \in W$. Then*[8]

(i) $\sigma \lhd_L \sigma'$ *implies* $\tau \to \sigma \lhd_L \tau \to \sigma'$ *and* $\sigma \to \sigma'' \lhd_L \sigma' \to \sigma''$
(ii) $L \neq \emptyset$ *implies* $iexp \lhd_L cmd$
(iii) $\sigma \lhd_L loc \to \sigma$
(iv) $\sigma \to \sigma \to \sigma' \lhd_L (loc \to \sigma) \to \sigma'$
(v) $\tau \to \tau' \to \sigma \lhd_L \tau' \to \tau \to \sigma$

The condition $L \neq \emptyset$ is necessary in (ii), because – by Theorem 8.1 – $[\![iexp]\!]_\emptyset$ is isomorphic to $\mathbb{Z}_\bot$ and $[\![cmd]\!]_\emptyset$ is isomorphic to $\{\bot,\top\}$, hence $[\![iexp]\!]_\emptyset$ cannot be a retract of $[\![cmd]\!]_\emptyset$.

Proof. (i) Let (S,R) be an L-SR-pair between σ and σ' and define

$$\bar{R} \equiv \lambda y^{\tau\to\sigma'}.\lambda z^\tau.R(yz), \qquad \bar{S} \equiv \lambda y^{\tau\to\sigma}.\lambda z^\tau.S(yz)$$

$$\tilde{R} \equiv \lambda y^{\sigma'\to\sigma''}.\lambda z^\sigma.y(Sz), \qquad \tilde{S} \equiv \lambda y^{\sigma\to\sigma''}.\lambda z^{\sigma'}.y(Rz)$$

$[\![\bar{S}]\!]^D$ is locally determined by Lemma 9.8(ii), because $[\![S]\!]^D$ is locally determined and $[\![\bar{S}]\!]f = [\![S]\!]\circ f$ for all $f \in [\![\tau \to \sigma]\!]$. $[\![\tilde{S}]\!]^D$ is locally determined by Lemma 9.8 (i), because $[\![\tilde{S}]\!]fg = f([\![R]\!]g)$ for all $f \in [\![\sigma \to \sigma'']\!]$ and $g \in [\![\sigma']\!]$. Moreover, $[\![\bar{R}(\bar{S}y)z]\!] = [\![R(\bar{S}yz)]\!] = [\![R(S(yz))]\!] = [\![yz]\!]$ shows that $[\![\bar{R}(\bar{S}y)]\!] = [\![y]\!]$, and $[\![\tilde{R}(\tilde{S}y)z]\!] = [\![\tilde{S}y(Sz)]\!] = [\![y(R(Sz))]\!] = [\![yz]\!]$ shows that $[\![\tilde{R}(\tilde{S}y)]\!] = [\![y]\!]$.

[8] If we had product types in ALG, then we could replace (iv) by $\sigma \times \sigma \lhd_L loc \to \sigma$ and (v) by $\tau \times \tau' \lhd_L \tau' \times \tau$.

(ii) Let $l \in L$, let $R \equiv \lambda y^{cmd}.y;\ !l$ and $S \equiv \lambda y^{iexp}.l := y$. For all $f \in [\![iexp]\!]$ and $s \in D^{sto}$ we have $[\![S]\!]fs = s[f s/l]$, hence $[\![S]\!]^D = [\![S]\!]$ is locally determined with determining function id_{sto}. Moreover, $[\![R(Sy)]\!] = [\![Sy;\ !l]\!] = [\![l := y;\ !l]\!] = [\![y]\!]$.

(iii) Let $R \equiv \lambda y^{loc \to \sigma}.\mathbf{new}\,x\ \mathbf{in}\ yx\ \mathbf{end}$ and $S \equiv \lambda y^{\sigma}.\lambda x^{loc}.y$. Then $[\![S]\!]^D$ is locally determined by Lemma 9.8 (i), because $[\![S]\!]fl = f$ for all $f \in [\![\sigma]\!]$ and $l \in [\![loc]\!]$. Moreover, $[\![R(Sy)]\!] = [\![\mathbf{new}\,x\ \mathbf{in}\ Syx\ \mathbf{end}]\!] = [\![\mathbf{new}\,x\ \mathbf{in}\ y\ \mathbf{end}]\!] = [\![y]\!]$.

(iv) Let $R \equiv \lambda y^{(loc \to \sigma) \to \sigma'}.\lambda z_1^{\sigma}, z_2^{\sigma}.y(\lambda x^{loc}.\mathbf{if}\ !x = 0\ \mathbf{then}\ z_1\ \mathbf{else}\ z_2)$ and $S \equiv \lambda y^{\sigma \to \sigma \to \sigma'}.\lambda z^{loc \to \sigma}.y(\mathbf{new}\,x\ \mathbf{in}\ x := 0;\ zx\ \mathbf{end})(\mathbf{new}\,x\ \mathbf{in}\ x := 1;\ zx\ \mathbf{end})$. Then $[\![S]\!]^D$ is locally determined by Lemma 9.8 (i) and

$$\begin{aligned}
[\![R(Sy)z_1z_2]\!] &= [\![Sy(\lambda x.\mathbf{if}\ !x = 0\ \mathbf{then}\ z_1\ \mathbf{else}\ z_2)]\!]\\
&= [\![y\ (\mathbf{new}\,x\ \mathbf{in}\ x := 0;\ \mathbf{if}\ !x = 0\ \mathbf{then}\ z_1\ \mathbf{else}\ z_2\ \mathbf{end})\\
&\qquad (\mathbf{new}\,x\ \mathbf{in}\ x := 1;\ \mathbf{if}\ !x = 0\ \mathbf{then}\ z_1\ \mathbf{else}\ z_2\ \mathbf{end})]\!]\\
&= [\![y(\mathbf{new}\,x\ \mathbf{in}\ x := 0;\ z_1\ \mathbf{end})(\mathbf{new}\,x\ \mathbf{in}\ x := 1;\ z_2\ \mathbf{end})]\!]\\
&= [\![yz_1z_2]\!] \quad \text{by the remark after Example 7.1}
\end{aligned}$$

hence $[\![R(Sy)]\!] = [\![y]\!]$.

(v) $R \equiv \lambda y^{\tau' \to \tau \to \sigma}.\lambda z_1^{\tau}, z_2^{\tau'}.yz_2z_1$ and $S \equiv \lambda y^{\tau \to \tau' \to \sigma}.\lambda z_1^{\tau'}, z_2^{\tau}.yz_2z_1$ define (even) an isomorphism, and obviously $[\![S]\!]^D$ is locally determined. □

Theorem 9.10 (L-FD-types). *Let σ be a procedure type with $ord(\sigma) \leqslant 2$ and let $L \neq \emptyset$. Then σ is an L-FD-type.*

Proof. We will prove that

(1) $loc^m \to cmd$ is an L-FD-type for every $m \geqslant 0$, $L \neq \emptyset$

(2) $(loc^m \to cmd) \to cmd$ is an L-FD-type for every $m \geqslant 0$, $L \neq \emptyset$

(3) $(loc \to \sigma)$ is an L-FD-type for all $L \neq \emptyset$, if σ is an L-FD-type for all $L \neq \emptyset$

From (1) we obtain by Theorem 9.7 that all first order types are L-FD-types, because $loc^m \to iexp \lhd_L loc^m \to cmd$ by Lemma 9.9(i) and (ii). From (2) we obtain, again by Theorem 9.7, that $\sigma_1 \to \cdots \to \sigma_k \to \theta$ is an L-FD-type, provided that all σ_i are first order types, namely: If $\sigma_i = loc^{m_i} \to \theta_i$ and $m = max\{m_1, \ldots, m_k\}$, then $\sigma_1 \to \cdots \to \sigma_k \to \theta \lhd_L (loc^m \to cmd)^k \to \theta \lhd_L (loc^{m+k-1} \to cmd) \to cmd$ by Lemma 9.9(i)–(iv). Together with (3) this implies that all types of the form $loc^m \to \sigma_1 \to \cdots \to \sigma_k \to \theta$ with first order types σ_i are L-FD-types. But from these we obtain any arbitrary second order type by a permutation of the parameter types, hence another application of Theorem 9.7 combined with Lemma 9.9 (v) shows that all second order types are L-FD-types.

Proof of (1): Let $\sigma = loc^m \to cmd$ and $L \in W$. We will show that $(P^{\sigma}_{n,L})_{n \in \mathbb{N}}$ with

$$P^{\sigma}_{n,L} \equiv \lambda y^{\sigma}.\lambda x_1^{loc}, \ldots, x_m^{loc}.$$

$$\mathbf{proc}\ z\colon \mathbf{if}\ \bigwedge_{i=1}^{m} abs(!x_i) \leqslant n \wedge \bigwedge_{l \in L} abs(!l) \leqslant n\ \mathbf{then}\ skip\ \mathbf{else}\ \Omega$$
$$\mathbf{in}\ z;\ yx_1 \ldots x_m;\ z\ \mathbf{end}$$

is an L-FD-sequence on σ with the additional property that, for every $n \in \mathbb{N}$, $[\![P^{\sigma}_{n,L}]\!]$ is idempotent and $[\![P^{\sigma}_{n,L}]\!]([\![\sigma]\!]_L)$ is finite (hence $[\![\sigma]\!]_L$ is an SFP object). To this end we define $p^{sto}_{n,L'} \in [\![cmd]\!]_{L'}$ for every $n \in \mathbb{N}$ and $L' \in W$ by

$$p^{sto}_{n,L'}\, s = \begin{cases} s & \text{if } sL' \subseteq \{-n,\dots,n\} \\ \bot & \text{otherwise} \end{cases}$$

Clearly, for every $L' \in W$, $(p^{sto}_{n,L'})_{n\in\mathbb{N}}$ is an ω-chain of idempotent functions in $[\![cmd]\!]_{L'}$ such that $\bigsqcup_{n\in\mathbb{N}} p^{sto}_{n,L'} = [\![skip]\!]$ and $p^{sto}_{n,L'}(Stores_{L'})$ is finite for every $n \in \mathbb{N}$. Now note that

$$[\![P^{\sigma}_{n,L}]\!] f l_1 \dots l_m = p^{sto}_{n,L\cup\{l_1,\dots,l_m\}} \circ (f l_1 \dots l_m) \circ p^{sto}_{n,L\cup\{l_1,\dots,l_m\}}$$

for all $f \in [\![\sigma]\!]$ and $l_1,\dots,l_m \in Loc$. This implies immediately that $([\![P^{\sigma}_{n,L}]\!])_{n\in\mathbb{N}}$ is an ω-chain of idempotent functions with $\bigsqcup_{n\in\mathbb{N}}[\![P^{\sigma}_{n,L}]\!] = \mathrm{id}_{\sigma}$. It remains to be shown that, for every $n \in \mathbb{N}$, $[\![P^{\sigma}_{n,L}]\!]^D \,|\, ([\![\sigma]\!]_L)^d$ is finitely determined and that $[\![P^{\sigma}_{n,L}]\!]([\![\sigma]\!]_L)$ is finite.

To this end let $f \in [\![\sigma]\!]_L$ and $l_1,\dots,l_m \in Loc$. By Theorem 5.3, $[\![P^{\sigma}_{n,L}]\!] f l_1 \dots l_m \in [\![cmd]\!]_{L\cup\{l_1,\dots,l_m\}}$ is uniquely determined by its restriction to $Stores_{L\cup\{l_1,\dots,l_m\}}$ and hence also by the restriction of $f l_1 \dots l_m$ to $p^{sto}_{n,L\cup\{l_1,\dots,l_m\}}(Stores_{L\cup\{l_1,\dots,l_m\}}) \setminus \{\bot\}$. This means that

$$A = \{(l_1,\dots,l_m,s) \,|\, l_1,\dots,l_m \in Loc \wedge s \in p^{sto}_{n,L\cup\{l_1,\dots,l_m\}}(Stores_{L\cup\{l_1,\dots,l_m\}}) \setminus \{\bot\}\}$$

is a determining set for $[\![P^{\sigma}_{n,L}]\!]^D \,|\, ([\![\sigma]\!]_L)^d$. In order to obtain a *finite* determining set we define an equivalence relation $\sim$ on A by

$$(l_1,\dots,l_m,s) \sim (l'_1,\dots,l'_m,s') \Leftrightarrow \exists\, \varphi \in Fix(L). \forall i \in \{1,\dots,m\}. \varphi\, l_i = l'_i \wedge s = s' \circ \varphi$$

This equivalence relation has finite index, because the equivalence class of an element $(l_1,\dots,l_m,s)$ is uniquely determined by the sets $\{(i,j) \in \{1,\dots,m\}^2 \,|\, l_i = l_j\}$ and $\{(l,i) \in L \times \{1,\dots,m\} \,|\, l = l_i\}$, the function $(s\,|\,L) \in (L \xrightarrow{t} \{-n,\dots,n\})$ and the tuple $(sl_1,\dots,sl_m) \in \{-n,\dots,n\}^m$. Moreover, the restriction of f^d to any equivalence class is uniquely determined by its value for one representative of the class, because $f^d(\varphi\, l_1,\dots \varphi\, l_m, s\circ\varphi^{-1}) = f(\varphi\, l_1)\dots(\varphi\, l_m)(s\circ\varphi^{-1}) = (f l_1 \dots l_m s)\circ\varphi^{-1} = f^d(l_1,\dots,l_m,s)\circ\varphi^{-1}$ by Theorem 5.3 (v). Thus, every representation system B for $\sim$ is a finite determining set for $[\![P^{\sigma}_{n,L}]\!]^D \,|\, ([\![\sigma]\!]_L)^d$. Finally note that $([\![P^{\sigma}_{n,L}]\!] f)^d =_B ([\![P^{\sigma}_{n,L}]\!] g)^d$ implies $[\![P^{\sigma}_{n,L}]\!] f = [\![P^{\sigma}_{n,L}]\!]([\![P^{\sigma}_{n,L}]\!] f) = [\![P^{\sigma}_{n,L}]\!]([\![P^{\sigma}_{n,L}]\!] g) = [\![P^{\sigma}_{n,L}]\!] g$ for all $f,g \in [\![\sigma]\!]_L$, i.e. $[\![P^{\sigma}_{n,L}]\!] f$ is uniquely determined by the finite value table

$$([\![P^{\sigma}_{n,L}]\!] f l_1 \dots l_m s)_{(l_1,\dots,l_m,s)\in B}$$

The set of all such value tables is finite, because the possible entries for any element $(l_1,\dots,l_m,s) \in B$ can only range over the finite set $p^{sto}_{n,L\cup\{l_1,\dots,l_m\}}(Stores_{L\cup\{l_1,\dots,l_m\}})$. This shows that $[\![P^{\sigma}_{n,L}]\!]([\![\sigma]\!]_L)$ is finite.

Proof of (2): Let $\sigma' = (loc^m \to cmd) \to cmd$ and $L \in W$. We continue to use the notation from the proof of (1), in particular σ stands for $loc^m \to cmd$.

The first idea which comes to mind for defining an L-FD-sequence on σ' is to imitate the definition of idempotent deflations [1] for the functional language PCF [14, 20, 32], i.e. to define

$$P^{\sigma'}_{n,L} \overset{?}{\equiv} \lambda y^{\sigma'}.\lambda z^{\sigma}.P^{cmd}_{n,L}(y\,(P^{\sigma}_{n,L}z))$$

Unfortunately this idea is too naive: As the elements of $[\![\sigma']\!]_L$ cannot be considered as functions from $[\![\sigma]\!]_L$ to $[\![cmd]\!]_L$ (but also map $[\![\sigma]\!]_{L'}$ to $[\![cmd]\!]_{L'}$ for all $L' \supseteq L$), we cannot really expect that this simple PCF-approach carries over to ALG, and indeed it turns out that the functions $[\![P^{\sigma'}_{n,L}]\!]^D \,|\, ([\![\sigma']\!]_L)^d$ are *not* finitely determined (nor do they have finite image). Somewhat to our own surprise, this problem can already be solved by using a slightly modified definition, namely

$$P^{\sigma'}_{n,L} \equiv \lambda y^{\sigma'}.\lambda z^{\sigma}.\mathbf{new}\,x\,\mathbf{in}\,P^{cmd}_{n,L}(y\,(\mathbf{if}\ !x < n\ \mathbf{then}\ x := !x+1; P^{\sigma}_{n,L}z\ \mathbf{else}\ \Omega))\,\mathbf{end}$$

The difference to the first definition is that we now use a local variable x to count the procedure calls of z (as in Example 7.7) and that we let $P^{\sigma'}_{n,L}y\,z$ diverge as soon as the number of these procedure calls exceeds n. We will show that these new terms $P^{\sigma'}_{n,L}$ indeed define an L-FD-sequence on σ'.

Clearly, $P^{\sigma'}_{n,L} \in c\text{-ALG}^{\sigma'\to\sigma'}_L$ and $([\![P^{\sigma'}_{n,L}]\!])_{n\in\mathbb{N}}$ is an ω-chain with $\bigsqcup_{n\in\mathbb{N}}[\![P^{\sigma'}_{n,L}]\!] = [\![\lambda y.\lambda z.\mathbf{new}\,x\ \mathbf{in}\ y\,(x := !x+1;\,z)\ \mathbf{end}]\!] = [\![\lambda y.\lambda z.\,y\,z]\!] = \mathrm{id}_{\sigma'}$, where the second equality holds by Example 7.7. The hard part is to show that $[\![P^{\sigma'}_{n,L}]\!]^D \,|\, ([\![\sigma']\!]_L)^d$ is finitely determined for every $n \in \mathbb{N}$:

Let A and $\sim$ be defined as before. As the set $A/\!\sim$ of all $\sim$-equivalence classes eq is finite, we may encode a word $w \in (A/\!\sim)^*$ as an integer and thus store it into a location. We use $eq.w$ to denote the concatenation of eq and w and $|w|$ to denote the length of w and – in order to simplify notation – we do not explicitly distinguish between a word w and its code. Below we will use an element $eq \in A/\!\sim$ as a(n incomplete) description of a procedure call of some fixed procedure $g \in [\![\sigma]\!]$, hence a word $w \in (A/\!\sim)^*$ stands for a sequence of such procedure calls.

Now let $Seq_{<n} = \{w \in (A/\!\sim)^* \mid |w| < n\}$ and let $l \in Loc \setminus L$. For every function $\Phi : Seq_{<n} \to [\![P^{\sigma}_{n,L}]\!]([\![\sigma]\!]_L)$ we define $c_\Phi \in [\![\sigma]\!]_{L\cup\{l\}}$ by

$$c_\Phi l_1 \ldots l_m s = \begin{cases} \Phi(sl)\, l_1 \ldots l_m(s\,[class\, l_1 \ldots l_m s\,.\,sl/l]) & \text{if } sl \in Seq_{<n} \\ \bot & \text{otherwise} \end{cases}$$

where $class \in [\![loc^m \to iexp]\!]_L$ is such that

$$class\, l_1 \ldots l_m\, s = \begin{cases} eq \text{ if } (l_1,\ldots,l_m,s) \in eq \\ \bot \text{ if } (l_1,\ldots,l_m,s) \notin A \end{cases}$$

for all $l_1,\ldots,l_m \in Loc$, $s \in Stores_{L\cup\{l_1,\ldots,l_m\}}$. Note that by Theorem 8.1 (i) and (iii) such a function $class$ exists and is uniquely determined. Moreover, each c_Φ is indeed in $[\![\sigma]\!]_{L\cup\{l\}}$, because it is defined by a finite case distinction on the contents of l from the function $class$ and the functions $\Phi(w) \in [\![\sigma]\!]_L$ ($w \in Seq_{<n}$). To obtain some intuition for these functions, note that each c_Φ uses the location l to keep a record of

its own history of procedure calls and diverges as soon as the recorded history becomes longer than n. The role of the index Φ is to describe how a call of c_Φ depends on the previously recorded history.

From the proof of (1) we know that $[\![P^\sigma_{n,L}]\!]([\![\sigma]\!]_L)$ and $p^{sto}_{n,L}(Stores_L)$ are finite, hence the set

$$C = \{(c_\Phi, s[\varepsilon/l]) \mid \Phi : Seq_{<n} \to [\![P^\sigma_{n,L}]\!]([\![\sigma]\!]_L) \wedge s \in p^{sto}_{n,L}(Stores_L) \setminus \{\bot\}\}$$

is also finite, and we will prove that it is a determining set for $[\![P^{\sigma'}_{n,L}]\!]^D \mid ([\![\sigma']\!]_L)^d$. More precisely, we will show that for every $(g,s) \in [\![\sigma]\!] \times Stores$ there is some $(c_\Phi, s'[\varepsilon/l]) \in C$ such that $f c_\Phi(s'[\varepsilon/l]) = f' c_\Phi(s'[\varepsilon/l]) \Rightarrow [\![P^{\sigma'}_{n,L}]\!] f g s = [\![P^{\sigma'}_{n,L}]\!] f' g s$ whenever $f, f' \in [\![\sigma']\!]_L$ (i.e. $[\![P^{\sigma'}_{n,L}]\!]^D \mid ([\![\sigma']\!]_L)^d$ is not only finitely determined but also locally determined). The intuition for the proof is that the computation of $[\![P^{\sigma'}_{n,L}]\!] f g s$ can be simulated by the computation of $f c_\Phi(s'[\varepsilon/l])$ for some appropriate Φ and s'. *This is a surprising fact* and we consider it *as one of the central points of the whole full abstraction proof.* Note that, although every single function $g \in [\![\sigma]\!]$ can only have access to finitely many locations $l_1, \ldots, l_k \in Loc \setminus L$, there is *no upper bound* on the number k of these locations. Moreover, the values which are stored in $l_1, \ldots, l_k$ during the computation of $[\![P^{\sigma'}_{n,L}]\!] f g s$ can in no way be controlled by $[\![P^{\sigma'}_{n,L}]\!] f$, i.e. the simulation must cope with *arbitrarily large* values in $l_1, \ldots, l_k$. On the other hand, there are only *finitely many* functions c_Φ, which only use a *single* location $l \in Loc \setminus L$ and, by their very definition, can only store *finitely many* different values in l, namely the words w with $|w| \leqslant n$. Altogether, this means that there is no hope for a naive simulation, which uses some direct encoding of the values in $l_1, \ldots, l_k$ into the contents of l. The trick is to use an 'indirect encoding' which is based on histories of procedure calls. This encoding will now be defined.

Let $g \in [\![\sigma]\!]$ and $s \in Stores$. The idempotence of $[\![P^\sigma_{n,L}]\!]$ implies that $[\![P^{\sigma'}_{n,L}]\!] f g s = [\![P^{\sigma'}_{n,L}]\!] f ([\![P^\sigma_{n,L}]\!] g) s$ for all $f \in [\![\sigma']\!]$, hence we may assume that g itself is already contained in $[\![P^\sigma_{n,L}]\!] [\![\sigma]\!]$. Now let $l_1, \ldots, l_k \in Loc \setminus L$ be such that $g \in [\![\sigma]\!]_{L \cup \{l_1, \ldots, l_k\}}$, $s \in Stores_{L \cup \{l_1, \ldots, l_k\}}$ and $l \in \{l_1, \ldots, l_k\}$. For every equivalence class $eq \in A/\!\sim$ and every $i \in \{1, \ldots, k\}$ we define $g^{eq}_i : \mathbb{Z}^k_\bot \to \mathbb{Z}_\bot$ by

$$g^{eq}_i(d_1, \ldots, d_k) = g\, l'_1 \ldots l'_m(t[d_1/l_1] \ldots [d_k/l_k])\, l_i$$
$$\text{where } (l'_1, \ldots, l'_m, t) \in eq \text{ and } l'_1, \ldots, l'_m \notin \{l_1, \ldots, l_k\}$$

In order to see that the functions g^{eq}_i are well-defined, first note that every eq does indeed contain a tuple $(l'_1, \ldots, l'_m, t)$ with $l'_1, \ldots, l'_m \notin \{l_1, \ldots, l_k\}$. Moreover, if we have two such tuples in the same equivalence class eq, then we may assume that the corresponding permutation φ in the definition of $\sim$ is contained in $Fix(L \cup \{l_1, \ldots, l_k\})$ and then Theorem 5.3 (v) implies $g(\varphi l'_1) \ldots (\varphi l'_m)((t \circ \varphi^{-1})[d_1/l_1] \ldots [d_k/l_k])\, l_i = g(\varphi l'_1) \ldots (\varphi l'_m)((t[d_1/l_1] \ldots [d_k/l_k]) \circ \varphi^{-1})\, l_i = g\, l'_1 \ldots l'_m(t[d_1/l_1] \ldots [d_k/l_k])(\varphi^{-1} l_i)$ $= g\, l'_1 \ldots l'_m(t[d_1/l_1] \ldots [d_k/l_k])\, l_i$, i.e. both tuples lead to the same result.

Finally we define a value $d_i^w \in \mathbb{Z}_\perp$ for every $w \in (A/\sim)^*$ and $i \in \{1,\dots,k\}$ by

$$d_i^\varepsilon = sl_i, \qquad d_i^{eq.w} = g_i^{eq}(d_1^w,\dots,d_k^w)$$

Intuitively, d_i^w is the current contents of the location l_i, if w describes the history of procedure calls of g (and if the initial contents of $l_1,\dots,l_k$ is given by s). The above argumentation has shown that the values d_i^w are uniquely determined by the (incomplete) description w, hence w is indeed an 'indirect encoding' of the current contents of $l_1,\dots,l_k$. Moreover, the counter x in the definition of $P_{n,L}^{\sigma'}$ will guarantee that the sequence of procedure calls of g will never become longer than n, hence we will need only words w of length $\leqslant n$ for the encoding.

Now we are ready to choose the appropriate function c_Φ which will make our simulation work. Let $\Phi : Seq_{<n} \to [\![P_{n,L}^\sigma]\!]([\![\sigma]\!]_L)$ be such that

$$\Phi(w)\, l'_1 \dots l'_m\, t =_{L\cup\{l'_1,\dots,l'_m\}} g\, l'_1 \dots l'_m(t\,[d_1^w/l_1]\dots[d_k^w/l_k])$$

for all $l'_1,\dots,l'_m \in Loc \setminus \{l_1,\dots,l_k\}$ and $t \in Stores$. It follows easily from Theorem 8.1 (ii) and (iv) that the function Φ exists and is uniquely determined. To obtain some intuition for Φ, note that the procedure calls $\Phi(w)\, l'_1 \dots l'_m$ and $g\, l'_1 \dots l'_m$ have the same effect on $L \cup \{l'_1,\dots,l'_m\}$, provided that w describes the history of procedure calls of g. Hence $c_\Phi l'_1 \dots l'_m$ indeed simulates $g\, l'_1 \dots l'_m$ in the following sense: First it reads the word w from the location l, then it behaves like $\Phi(w)\, l'_1 \dots l'_m$, i.e. it acts like $g\, l'_1 \dots l'_m$ on $L \cup \{l'_1,\dots,l'_m\}$, and finally it extends the contents w of l by the description eq of the current procedure call. The last step guarantees that the contents $eq.w$ of l after the call of c_Φ encodes the contents $d_1^{eq.w},\dots,d_k^{eq.w}$ of $l_1,\dots,l_k$ after the call of g. Of course, all this is only a vague intuition, which will now be replaced by a mathematically rigorous proof. In particular, our intuitive understanding of a 'simulation' will be expressed by an appropriate logical relation.

To this end let $s' = p_{n,L}^{sto}(s\,[0/l_1]\dots[0/l_k]) \in p_{n,L}^{sto}(Stores_L)$. We must show that $[\![P_{n,L}^{\sigma'}]\!] f g s$ is uniquely determined by $f c_\Phi(s'[\varepsilon/l])$. First note that $[\![P_{n,L}^{\sigma'}]\!] f g s = ([\![P_{n,L}^{cmd}]\!](f g')(s[0/l']))[sl'/l']$, where l' is new, say $l' \notin L \cup \{l_1,\dots,l_k\}$, and $g' \in [\![\sigma]\!]_{L\cup\{l_1,\dots,l_k,l'\}}$ is defined by

$$g'\, l'_1 \dots l'_m t = \begin{cases} g\, l'_1 \dots l'_m(t\,[t\,l' + 1/l']) & \text{if } t\,l' < n \\ \perp & \text{otherwise} \end{cases}$$

Now let $S \in DEF_2^{\{l_1,\dots,l_k,l'\}} \subseteq OUT_2^L$ be defined by

$$S^{sto} = \{\perp\}^2 \cup \{\vec{t} \in Stores^2 \mid \exists w \in (A/\sim)^*.\, t_1 l = w \wedge t_2 l' = |w| \leqslant n \wedge$$
$$\forall i \in \{1,\dots,k\}.\, t_2 l_i = d_i^w \wedge t_1 =_{Loc\setminus\{l_1,\dots,l_k,l'\}} t_2\}$$

Clearly, $(s\,[0/l_1]\dots[0/l_k]\,[\varepsilon/l], s\,[0/l']) \in S^{sto}$, and if we can prove $(c_\Phi, g') \in S^\sigma$, then we obtain

$$([\![P_{n,L}^{cmd}]\!](f c_\Phi)(s\,[0/l_1]\dots[0/l_k]\,[\varepsilon/l]),\ [\![P_{n,L}^{cmd}]\!](f g')(s\,[0/l'])) \in S^{sto}$$

for all $f \in [\![\sigma]\!]_L$, because $[\![P^{cmd}_{n,L}]\!] \circ f$ is also in $[\![\sigma]\!]_L$. The left-hand side of this pair equals $p^{sto}_{n,L}(f c_\Phi(s'[\varepsilon/l]))$ and the right-hand side uniquely determines $[\![P^{\sigma'}_{n,L}]\!] f g s$. As S^{sto} is a partial function, this implies that $[\![P^{\sigma'}_{n,L}]\!] f g s$ is indeed uniquely determined by $f c_\Phi(s'[\varepsilon/l])$.

It remains to be shown that $(c_\Phi, g') \in S^\sigma$. It can be easily seen that $S^{loc} = \delta^2(Loc \setminus \{l_1, \ldots, l_k, l'\})$, hence we must prove $(c_\Phi l'_1 \ldots l'_m, g' l'_1 \ldots l'_m) \in S^{cmd}$ for all $l'_1, \ldots, l'_m \in Loc \setminus \{l_1, \ldots, l_k, l'\}$. To this end let $(t_1, t_2) \in S^{sto} \setminus \{\bot\}^2$. Then there is some $w \in (A/\!\sim)^*$ with $t_1 l = w$, $t_2 l' = |w| \leqslant n$, $t_2 l_i = d_i^w$ for $i = 1, \ldots, k$ and $t_1 =_{Loc \setminus \{l_1, \ldots, l_k, l'\}} t_2$. If $t_1, t_2 \notin p^{sto}_{n, L \cup \{l'_1, \ldots, l'_m\}}(Stores)$ then $c_\Phi l'_1 \ldots l'_m t_1 = \bot$ because $class\, l'_1 \ldots l'_m t_1 = \bot$, and $g' l'_1 \ldots l'_m t_2 = \bot$ because $g \in [\![P^\sigma_{n,L}]\!][\![\sigma]\!]$. If $|w| = n$ then $c_\Phi l'_1 \ldots l'_m t_1 = \bot$ because $t_1 l = w \notin Seq_{<n}$, and $g' l'_1 \ldots l'_m t_2 = \bot$ because $t_2 l' = |w| = n$. Hence we have $(c_\Phi l'_1 \ldots l'_m t_1, g' l'_1 \ldots l'_m t_2) = (\bot, \bot) \in S^{sto}$ in both cases. The only remaining case is $t_1, t_2 \in p^{sto}_{n, L \cup \{l'_1, \ldots, l'_m\}}(Stores) \wedge |w| < n$. Then there is some $eq \in A/\!\sim$ with $class\, l'_1 \ldots l'_m t_1 = eq$ and we obtain

$$\begin{aligned}
g' l'_1 \ldots l'_m t_2 &= g\, l'_1 \ldots l'_m (t_2[t_2 l' + 1/l']) \quad \text{because } t_2\, l' = |w| < n \\
&=_{L \cup \{l_1, \ldots, l_k, l'_1, \ldots, l'_m\}} g\, l'_1 \ldots l'_m (t_1[d_1^w/l_1, \ldots, d_k^w/l_k]) \\
&\qquad\qquad \text{because } t_2[t_2 l' + 1/l'] =_{Loc \setminus \{l'\}} t_1[d_1^w/l_1, \ldots, d_k^w/l_k] \\
&=_{L \cup \{l'_1, \ldots, l'_m\}} \Phi(w)\, l'_1 \ldots l'_m t_1 \quad \text{per definition of } \Phi \\
&=_{L \cup \{l'_1, \ldots, l'_m\}} \Phi(w)\, l'_1 \ldots l'_m (t_1[eq.w/l]) \quad \text{because } t_1 =_{L \cup \{l'_1, \ldots, l'_m\}} t_1[eq.w/l] \\
&= c_\Phi l'_1 \ldots l'_m t_1 \quad \text{because } class\, l'_1 \ldots l'_m t_1 = eq \wedge t_1\, l = w
\end{aligned}$$

If $g' l'_1 \ldots l'_m t_2 = c_\Phi l'_1 \ldots l'_m t_1 = \bot$ then we are done. Otherwise note that

$$c_\Phi l'_1 \ldots l'_m t_1\, l = eq.w$$

$$g' l'_1 \ldots l'_m t_2\, l' = t_2\, l' + 1 = |eq.w|$$

$$g' l'_1 \ldots l'_m t_2\, l_i = g\, l'_1 \ldots l'_m (t_1[d_1^w/l_1] \ldots [d_k^w/l_k])\, l_i = g_i^{eq}(d_1^w, \ldots, d_k^w) = d_i^{eq.w}$$
$$\text{for every } i \in \{1, \ldots, k\}$$

$$g' l'_1 \ldots l'_m t_2 \text{and } c_\Phi l'_1 \ldots l'_m t_1 \text{coincide on } Loc \setminus \{l_1, \ldots, l_k, l'_1, \ldots, l'_m, l'\}$$

The latter observation implies that $g' l'_1 \ldots l'_m t_2$ and $c_\Phi l'_1 \ldots l'_m t_1$ even coincide on $Loc \setminus \{l_1, \ldots, l_k, l'\}$. Altogether this proves $(c_\Phi l'_1 \ldots l'_m t_1, g' l'_1 \ldots l'_m t_2) \in S^{sto}$ and thus concludes the proof of (2).

Proof of (3): Let $\sigma' = loc \to \sigma$, $L \in W$ and $L \neq \emptyset$.

We choose some location $l \in Loc \setminus L$. By assumption, there is an L-FD-sequence $(P^\sigma_{n,L})_{n \in \mathbb{N}}$ and an $(L \cup \{l\})$-FD-sequence $(P^\sigma_{n, L \cup \{l\}})_{n \in \mathbb{N}}$ on σ. For every $n \in \mathbb{N}$ let $P^\sigma_{n, L \cup \{x\}} \in \mathrm{ALG}_L^{\sigma \to \sigma}$ be the term which is obtained from $P^\sigma_{n, L \cup \{l\}}$ by substituting a fresh variable x for the location constant l, and let $P^{\sigma'}_{n,L} \in c\text{-}\mathrm{ALG}_L^{\sigma' \to \sigma'}$ be defined by

$$P^{\sigma'}_{n,L} \equiv \lambda y^{\sigma'}.\lambda x^{loc}.\textbf{if} \bigvee_{l' \in L} EQ\, x\, l' \textbf{ then } P^\sigma_{n,L}(yx) \textbf{ else } P^\sigma_{n, L \cup \{x\}}(yx)$$

where $EQ \in c\text{-}\mathrm{ALG}_{\emptyset}^{loc^2 \to iexp}$ is the equality on locations and $y^{\sigma'}$ is some fresh identifier. Let $f \in [\![\sigma']\!]$. Then

$$[\![P_{n,L}^{\sigma'}]\!] f\, l' = \begin{cases} [\![P_{n,L}^{\sigma}]\!](f\, l') & \text{if } l' \in L \\ [\![P_{n,L\cup\{l\}}^{\sigma}]\!](f\, l) & \text{if } l' = l \end{cases}$$

and for $l' \notin L \cup \{l\}$ we conclude by Lemma 8.3 that

$$\begin{aligned} [\![P_{n,L}^{\sigma'}]\!] f\, l' &= [\![SWAP_{(l\,l')}^{\sigma}]\!]([\![P_{n,L}^{\sigma'}]\!]([\![SWAP_{(l\,l')}^{\sigma'}]\!] f)\, l) \\ &= [\![SWAP_{(l\,l')}^{\sigma}]\!]([\![P_{n,L\cup\{l\}}^{\sigma}]\!]([\![SWAP_{(l\,l')}^{\sigma'}]\!] f\, l)) \end{aligned}$$

where $(l\,l')$ denotes the transposition of l and l'. From these equations it follows easily that $([\![P_{n,L}^{\sigma'}]\!])_{n\in\mathbb{N}}$ is an ω-chain with $\bigsqcup_{n\in\mathbb{N}}[\![P_{n,L}^{\sigma'}]\!] = \mathsf{id}_{\sigma'}$.

It remains to be shown that $[\![P_{n,L}^{\sigma'}]\!]^D \,|\, ([\![\sigma']\!]_L)^d$ is finitely determined for all $n \in \mathbb{N}$. Let $f \in [\![\sigma']\!]_L$. Then $[\![SWAP_{(l\,l')}^{\sigma'}]\!] f = f$ whenever $l' \in Loc \setminus L$ and thus we obtain from the above equations

$$[\![P_{n,L}^{\sigma'}]\!] f\, l' = \begin{cases} [\![P_{n,L}^{\sigma}]\!](f\, l') & \text{if } l' \in L \\ [\![SWAP_{(l\,l')}^{\sigma}]\!]([\![P_{n,L\cup\{l\}}^{\sigma}]\!](f\, l)) & \text{if } l' \notin L \end{cases}$$

Now let B and B' be finite determining sets for the functions $[\![P_{n,L}^{\sigma}]\!]^D \,|\, ([\![\sigma]\!]_L)^d$ and $[\![P_{n,L\cup\{l\}}^{\sigma}]\!]^D \,|\, ([\![\sigma]\!]_{L\cup\{l\}})^d$. Then, for every $l' \in L$, $([\![P_{n,L}^{\sigma}]\!](f\, l'))^d = [\![P_{n,L}^{\sigma}]\!]^D (f\, l')^d$ is uniquely determined by $(f\, l')^d \,|\, B$ and $([\![P_{n,L\cup\{l\}}^{\sigma}]\!](f\, l))^d = [\![P_{n,L\cup\{l\}}^{\sigma}]\!]^D (f\, l)^d$ is uniquely determined by $(f\, l)^d \,|\, B'$. Thus it follows from the above equation that $[\![P_{n,L}^{\sigma'}]\!]^D f^d = ([\![P_{n,L}^{\sigma'}]\!] f)^d$ is uniquely determined by $f^d \,|\, (L \times B) \cup (\{l\} \times B')$, i.e. $[\![P_{n,L}^{\sigma'}]\!]^D$ is indeed finitely determined. □

We conjecture that the restriction $L \neq \emptyset$ can be dropped from Theorem 9.10 and that *all* the domains $[\![\sigma]\!]_L$ with $ord(\sigma) \leqslant 2$ are SFP objects (as we already know for $\sigma = loc^m \to cmd$). But the only way to prove this would be to *directly* construct L-FD-sequences for all types of order $\leqslant 2$ in order to avoid applications of Lemma 9.9 (where the restriction $L \neq \emptyset$ comes from) and Theorem 9.7 (where the idempotence of functions gets lost). We preferred to take the *indirect* way via L-retracts, because it allowed us to restrict the tricky encoding in the proof of Theorem 9.10 to types of the form $(loc^m \to cmd) \to cmd$, and because we consider the use of L-retracts as an interesting technique in its own right.

We are now ready to prove full abstraction.

Theorem 9.11 (Approximation by definable functions). *Let σ be a procedure type of order 1 or 2 and let $L \neq \emptyset$. Then every $f \in [\![\sigma]\!]_L$ is the least upper bound of an ω-chain of definable functions $[\![M]\!]$ with $M \in c\text{-}\mathrm{ALG}_L^{\sigma}$.*

Proof. Let $(P_n)_{n\in\mathbb{N}}$ be an L-FD-sequence on σ, and for every $n \in \mathbb{N}$ let B_n be a finite determining set for $([\![P_n]\!])^D \,|\, ([\![\sigma]\!]_L)^d$. By Theorem 9.3 there are terms $M_n \in c\text{-}\mathrm{ALG}_L^{\sigma}$

with $[\![M_n]\!]^d =_{B_n} f^d$, hence $[\![P_nM_n]\!]^d = [\![P_n]\!]^D[\![M_n]\!]^d = [\![P_n]\!]^D f^d = ([\![P_n]\!]f)^d$ for every $n \in \mathbb{N}$. This implies $f = \bigsqcup_{n\in\mathbb{N}}[\![P_n]\!]f = \bigsqcup_{n\in\mathbb{N}}[\![P_nM_n]\!]$. $\square$

Now we obtain our full abstraction result by the usual argumentation [26]:

Theorem 9.12 (Full abstraction). *Let σ be a type of order $\leqslant 3$ and let $M_1, M_2 \in c\text{-}\mathrm{ALG}^{\sigma}_{\emptyset}$. Then*

$$[\![M_1]\!] = [\![M_2]\!] \Leftrightarrow M_1 \approx M_2$$

Proof. Only '$\Leftarrow$' remains to be proved. Let $\sigma = \tau_1 \to \cdots \to \tau_k \to \theta$ $(k \geqslant 0)$ and assume that $[\![M_1]\!] \neq [\![M_2]\!]$, i.e. there are $d_j \in [\![\tau_j]\!]$ for $j = 1,\dots,k$ and $s \in \mathit{Stores}$ such that

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

Let $L \neq \emptyset$ be such that $d_j \in [\![\tau_j]\!]_L$ for $j = 1,\dots,k$. By Theorem 9.11, every d_j is the least upper bound of a directed set of definable elements in $[\![\tau_j]\!]_L$ (for $\tau_j = loc$, d_j is itself definable), hence the local continuity of $[\![M_1]\!]$ and $[\![M_2]\!]$ implies that there are $N_j \in c\text{-}\mathrm{ALG}^{\tau_j}_L$ with

$$[\![M_1N_1 \dots N_k]\!]\, s \neq [\![M_2N_1 \dots N_k]\!]\, s$$

From this, it is easy to construct a program context $C[\;]$ with $[\![C[M_1]]\!] \neq [\![C[M_2]]\!]$. Hence $M_1 \not\approx M_2$. $\square$

We have formulated Theorem 9.12 for *closed* terms of order $\leqslant 3$. Instead, we could have used *open* terms of order $\leqslant 2$ whose only free identifiers are of order $\leqslant 2$. In any case the main role is played by procedure identifiers of order $\leqslant 2$; that is why we speak of 'full abstraction for the second order subset'.

10. Variants of the language ALG

We have included some features in our language ALG which are not typical for an ALGOL-like language, hence it seems worthy to discuss whether they can be removed or whether some further ones can be added.

10.1. Removing the parallel conditional

The observant reader may have realized that the parallel conditional did not really play a role in our full abstraction proof, and indeed it can be removed from the language ALG without any difficulties:

Let ALG^{seq} be the *sequential subset* of ALG, which is obtained from ALG by removing the constant *pcond*. If we replace AUX by $AUX \setminus \{Pcond\}$ in the definition of the signature Σ, then we obtain a new signature Σ^{seq} which is strictly greater than Σ, in particular it contains ground relations which are similar to the *sequentiality relations* of [32]. One such example is the ground relation R with

$$R^{loc} = \delta^3 Loc$$
$$R^{\gamma} = \{\vec{d} \in (D^{\gamma})^3 \mid d_1 = \bot \vee d_2 = \bot \vee d_1 = d_2 = d_3\} \quad for \gamma = int, sto$$

which is contained in $(\Sigma^{seq})^L$ for all $L \in W$ (as can be easily checked). If we replace Σ by Σ^{seq} in the construction of our denotational model, then we obtain a 'smaller' model which is computationally adequate and fully abstract for the language ALG^{seq}. This can be proved exactly as before, because we have never made any real use of the parallel conditional in the proofs of computational adequacy and full abstraction. In the new model we can validate additional denotational equivalences like

$$[\![y\, skip\, \Omega + y\, \Omega\, skip]\!] = [\![2 * y\, \Omega\, \Omega]\!]$$

where $y : cmd \to cmd \to iexp$. This is a variant of the famous PCF-equivalence [26, 32], and it can be validated similarly as in [32]: Let $\eta \in Env$, $s \in Stores$ and let R be defined as above. Then $([\![skip]\!], [\![\Omega]\!], [\![\Omega]\!]) \in R^{cmd}$, $([\![\Omega]\!], [\![skip]\!], [\![\Omega]\!]) \in R^{cmd}$ and $(s, s, s) \in R^{sto}$, hence $([\![y\, skip\, \Omega]\!]\eta s, [\![y\, \Omega\, skip]\!]\eta s, [\![y\, \Omega\, \Omega]\!]\eta s) \in \eta\, y R^{cmd} R^{cmd} R^{sto} \subseteq R^{int}$. This means that one of the first two components must be $\bot$ or all three must be equal, and in both cases the above equality follows easily.

Note that – for the first time in this paper – we have used a relation of arity greater than 2 for proving an observational congruence. Indeed, it can be shown that *no binary relation* works for this example, and similar examples show that there is *no upper bound at all* on the arity of relations which are needed for proving observational congruences in ALG^{seq}. This is in contrast to ALG itself, where binary relations seem to be sufficient (cf. Sections 7 and 11).

10.2. Removing the snap back effect

In contrast to the parallel conditional, the snap back effect does play an important role in our full abstraction proof, and it is not (yet) clear whether we can obtain a fully abstract model without it.

First note that there are at least two (significantly) different options for a language without snap back, namely

- a language ALG^{-se} in which integer expressions have *no* side effects at all [8, 9], not even temporary ones,
- a language ALG^{+se} in which integer expressions may have *permanent* side effects [38].

ALG^{-se} can be defined by removing the constant seq_{iexp} from ALG. As it is a subset of ALG, its observational congruence relation can only be coarser than the ALG-congruence, and indeed it is *strictly* coarser, as is illustrated by the terms

$$M_i \;\equiv\; \mathbf{new}\, x \;\mathbf{in}\; y^{cmd\to cmd}(x := 1;\, x' := i);\; \mathbf{if}\; !x = 1 \;\mathbf{then}\; \Omega \;\mathbf{end} \quad (i = 1,2)$$

In ALG^{-se} we have $M_1 \approx M_2$ by the following (somewhat informal) argumentation: If we start M_1 and M_2 in the same initial store, then it only depends on this store (and not on the particular parameter) whether the procedure y ignores its parameter or whether it calls its parameter at least once. In the first case it is obvious that M_1 and M_2 either both diverge or terminate with the same result. In the second case the local variable x contains 1 after $y(x := 1;\, x' := 1)$ and also after $y(x := 1;\, x' := 2)$, because the global procedure y has no access to x *and* because the contents of x cannot snap back to 0. Hence M_1 and M_2 both diverge in this case. In ALG itself we have $M_1 \not\approx M_2$ because M_1 and M_2 can be distinguished by the program context $C[\;] \equiv \mathbf{new}\, x' \;\mathbf{in}\; \mathbf{proc}\; y^{cmd\to cmd}\colon \lambda z^{cmd}.\, x' := (z;\, !x')\; \mathbf{in}\; [\;];\; !x' \;\mathbf{end}\; \mathbf{end}$.

ALG^{+se} can be defined to have the same syntax as ALG^{seq} (the parallel conditional does not make sense if integer expressions can have permanent side effects) but of course it must have a rather different operational and denotational semantics, in particular $[\![iexp]\!]$ must consist of functions from D^{sto} to $D^{sto} \times D^{int}$. The two observational congruence relations of ALG^{seq} and ALG^{+se} are incomparable: On the one hand, the above terms M_1 and M_2 are observationally congruent in ALG^{+se} (with the same argumentation as before) but not in ALG^{seq}. On the other hand there are trivial examples of ALG^{seq}-congruences which do not hold in ALG^{+se}, e.g. $(x := 0;\, 1) \approx 1$.

As to finding fully abstract semantics, both ALG^{-se} and ALG^{+se} seem to create new problems. Although ALG^{-se} is just a syntactic restriction of ALG , we cannot use the same trick as for ALG^{seq} in order to obtain a larger signature (and thus a 'smaller' model), because AUX does not contain an auxiliary function which corresponds to the constant seq_{iexp}. Hence, if there is an appropriate signature at all for ALG^{-se}, new ideas seem to be necessary for defining it. For ALG^{+se} it is of course necessary to restructure the whole denotational model before searching for an appropriate signature. Some first steps which we have made into this direction seem to suggest that ALG^{+se} is more promising than ALG^{-se}.

10.3. Removing reference parameters

We have included parameters of type *loc* as a matter of convenience, but they are not important for our full abstraction result. Only some minor changes are necessary if we want to remove them from ALG: Of course '**new** x **in** … **end**' can no longer be considered as syntactic sugar; it must be introduced as an extra binding mechanism. Besides that we must only insist that environments η are injective on the set Id^{loc} because sharing between location identifiers is no longer possible in the restricted language. At some points the full abstraction proof must be carried out with more care in order to avoid redundant λ-abstractions (with reference parameters) in the distinguishing contexts (cf. [34]), but all in all it will even become simpler because some nasty case distinctions will disappear (especially in the proof of Theorem 9.3).

10.4. Adding value parameters

It should be no problem to add parameters of type *int* to ALG, i.e. to introduce call-by-value as an additional parameter passing mechanism (at ground type level). We conjecture that they can be handled similarly as the parameters of type *loc*.

11. Conclusions and open questions

We have defined a denotational semantics for an ALGOL-like language, and we have proved that it is fully abstract for the second order subset of that language. Our denotational model satisfies the usual 'goodness' criteria, namely, it is defined in a cartesian closed category, it is syntax-independent and – despite its rather technical definition – it allows us to give rigorous and simple proofs for all the test equivalences which have been proposed in the literature. The simplicity of these equivalence proofs is partially due to the fact that they are all based on relations of arity $\leqslant 2$ from the sub-signature OUT. This leads us to

Conjecture 11.1. *Theorem 9.12 remains valid, if we use a smaller signature for our model construction, namely the signature $\bar{\Sigma}$ with*

$$\bar{\Sigma}_1^L = OUT_1^L \cup \{(\{\bot_{int}\}, \{\bot_{sto}\}, Loc)\}, \qquad \bar{\Sigma}_2^L = OUT_2^L, \qquad \bar{\Sigma}_n^L = \emptyset \quad \text{for } n > 2$$

From our efforts to construct counter-examples, we have already gained some evidence that Conjecture 11 really holds. This would increase the 'tastefulness' of our model, because OUT (and hence $\bar{\Sigma}$) is defined more concretely than the original signature Σ. Moreover, we would come closer to O'Hearn and Tennent's parametric functor model [22], and so it could finally turn out that their model is also fully abstract for the second order subset of ALG. Therefore we consider Conjecture 11 as a worthwhile subject of further research. Since the only place where we needed relations $R \notin \bar{\Sigma}$ was the proof of Theorem 9.3, it would be sufficient to redo this proof with relations $R \in \bar{\Sigma}$.

The most obvious open question is, of course, whether our model is fully abstract for the full language ALG and not only for the second order subset. We believe that the answer is negative: Our intuition is that a global procedure acts on a local variable like a pure λ-term and hence the full abstraction problem for ALG should be closely related to the definability problem in the pure (simply typed) λ-calculus. From [10] it follows that (at least for a finite ground type) the λ-definable functions of order 3 cannot be characterized by logical relations, and so we expect that full abstraction for our ALG-model also fails (already) at order 3. In order to repair this, one might try to use 'Kripke logical relations of varying arity' [6,20] instead of our finitary logical relations, but this would certainly lead to a terribly difficult model construction and it is questionable whether such a model would provide any new insights into the nature of local variables. Hence we think that our ‘full abstraction for the second order subset’ is indeed the best result which one may expect at the current state of the art.

One may finally wonder whether our techniques can be transferred to call-by-value (i.e. ML-like as opposed to ALGOL-like) languages [24]. This is a question which we have not yet investigated. Although the observations in [24] indicate that additional problems might come up in the call-by-value setting, we are confident that at least our main ideas will be helpful.

Acknowledgements

I am deeply indebted to Albert Meyer with whom I began my work on the full abstraction problem for local variables in [13], and from whom I learnt to ask the right questions about programming languages (and I am glad to have found an answer to one of these questions). I am grateful to Peter O’Hearn and Bob Tennent for discussions about the relationship between our different approaches, to Jörg Zeyer for pointing out unclarities in an earlier version of the paper and to the referee who provided many valuable suggestions.

References

[1] S. Abramsky and A. Jung, Domain theory, in: S. Abramsky and D.M. Gabbay and T.S.E. Maibaum, ed., *Handbook of Logic in Computer Science*, Vol. III (Oxford University Press, Oxford, 1995), 1–168.

[2] J. de Bakker, *Mathematical Theory of Program Correctness*, International Series in Computer Science (Prentice Hall, Englewood Cliffs, NJ, 1980).

[3] M.J.C. Gordon, *The Denotational Description of Programming Languages* (Springer, Berlin, 1979).

[4] C.A. Gunter, *Semantics of Programming Languages: Structures and Techniques*, Foundations of Computing Series (MIT Press, Cambridge, MA, 1992).

[5] J.Y. Halpern, A.R. Meyer and B.A. Trakhtenbrot, The semantics of local storage, or what makes the free-list free?, in: *11th Annual ACM Symp. on Principles of Programming Languages* (1984) 245–257.

[6] A. Jung and J. Tiuryn, A new characterization of lambda definability, in: *Proc. Int'l Conf. on Typed Lambda Calculi and Applications (TLCA)* (Utrecht, The Netherlands, 1993) Lecture Notes in Computer Science, Vol. 664 (Springer, Berlin, 1993) 230–244.

[7] P.J. Landin, A correspondence between Algol 60 and Church’s lambda notation, *Comm. ACM* **8** (1965) 89–101,158–165.

[8] A.F. Lent, The category of functors from state shapes to bottomless cpos is adequate for block structure, Master’s thesis, M.I.T., Cambridge, 1992

[9] A.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* (available as Technical Report YALEU/DCS/RR-968, Yale University) Copenhagen, Denmark, 1993, 101–119.

[10] R. Loader, The undecidability of λ-definability, in: M. Zeleny, ed., *The Church Festschrift* (CSLI/University of Chicago Press, Chicago, 1994)

[11] Q.M. Ma and J.C. Reynolds, Types, abstraction and parametric polymorphism, in: S. Brookes, M. Main, A. Melton, M. Mislove and D. Schmidt, ed., Proc. 7th Conf. on Mathematical Foundations of Programming Semantics (Pittsburgh, 1991), Lecture Notes in Computer Science, Vol. 568 (Springer, Berlin, 1991) 1–40.

[12] A.R. Meyer, Semantical paradigms: notes for an invited lecture, with two appendices by Stavros Cosmadakis, in: *Proc. 3rd IEEE Symp. on Logic in Computer Science* (Edinburgh, 1988) 236–253.

[13] A.R. Meyer and K. Sieber, Towards fully abstract semantics for local variables: Preliminary report, in: *Proc. 15th Annual ACM Symp. on Principles of Programming Languages* (San Diego, 1988) 191–203.

[14] R. Milner, Fully abstract models for typed λ-calculi, *Theoret. Comput. Sci.* 4 (1977) 1–22.

[15] R. Milner and M. Tofte, *Commentary on Standard ML* (MIT Press, Cambridge, MA, 1991).

[16] R. Milner, M. Tofte and R. Harper, *The Definition of Standard ML* (MIT Press, Cambridge, MA, 1990).

[17] J.C. Mitchell, Representation independence and data abstraction (Preliminary version), in: *Proc. 13th Annual ACM Symp. on Principles of Programming Languages* (St. Petersburg, Florida, 1986) 263–276.

[18] J.C. Mitchell, Type systems for programming languages, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science*, Vol. B, (North-Holland, Amsterdam, 1990) chapter 8, 365–458.

[19] H.R. Nielson and F. Nielson, *Semantics with Applications: A Formal Introduction* (Wiley Professional Computing, Wiley, New York, 1992).

[20] P.W. O'Hearn and J.G. Riecke, Kripke logical relations and PCF, Inform. and Comput. **120**(1) (1995) 107–116.

[21] P.W. O'Hearn and R.D. Tennent, Semantics of local variables, in: M.P. Fourman, P.T. Johnstone and A.M. Pitts, eds., *Proc. LMS Symp. on Applications of Categories in Computer Science, Durham* 1991, LMS Lecture Note Series, Vol. 177 (Cambridge University Press, Cambridge, 1992). 217–238.

[22] P.W. O'Hearn and R.D. Tennent, Relational parametricity and local variables, in: *Proc. 20th Annual ACM Symp. on Principles of Programming Languages* (Charleston, 1993) 171–184.

[23] P.W. O'Hearn and R.D. Tennent, Parametricity and local variables, *J. ACM* **42**(3) (1995) 658–709.

[24] A.M. Pitts and I.D.B. Stark, Observable properties of higher order functions that dynamically create local names, or: What's *new*?, in: A.M. Borzyszkowski and S. Sokołowski, eds., *Proc. 18th Internat. Symp. on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, Vol. 711 (Springer, Berlin, 1993) 122–141.

[25] G.D. Plotkin, A powerdomain construction, *SIAM J. on Comput.* **5**(3) (1976) 452–487.

[26] G.D. Plotkin, LCF considered as a programming language, *Theoret. Comput. Sci.* **5** (1977) 223–256.

[27] G.D. Plotkin, Lambda-definability in the full type hierarchy, in: J.P. Seldin and J.R. Hindley, eds., *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism* (Academic Press, NewYork, 1980) 363–374.

[28] G.D. Plotkin, A structural approach to operational semantics, Technical Report DAIMI FN-19, Aarhus University, Computer Science Department, Denmark, 1981.

[29] J.C. Reynolds, *The Craft of Programming*, International Series in Computer Science (Prentice-Hall, Englewood Cliffs, NJ, 1981).

[30] J.C. Reynolds, The essence of ALGOL, in: J. deBakker, and van Vliet, eds., *Internat Symp. Algorithmic Languages*, (IFIP, North-Holland, Amsterdam, 1981) 345–372.

[31] K. Sieber, A partial correctness logic for procedures (in an ALGOL-like language), in: R. Parikh, ed., *Proc. Logics of Programs* (Brooklyn, 1985), Lecture Notes in Computer Science, Vol. 193 (Springer, Berlin, 1985) 320–342.

[32] K. Sieber, Reasoning about sequential functions via logical relations, in: M.P. Fourman, P.T. Johnstone and A.M. Pitts, eds., *Proc. LMS Symp. on Applications of Categories in Computer Science* (Durham 1991) LMS Lecture Note Series 177 (Cambridge University Press, Cambridge, 1992) 258–269.

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

[34] A. Stoughton, Mechanizing logical relations, in: S. Brookes et al., eds., *Proc. 9th Internat Conf. on Mathematical Foundations of Programming Semantics* (New Orleans, 1993) Lecture Notes in Computer Science, Vol. 802 (Springer, Berlin, 1993) 359–377.

[35] R.D. Tennent, Semantical analysis of specification logic, *Inform. and Comput.* **85** (1990) 135–162.

[36] R.D. Tennent, *Semantics of Programming Languages*, International Series in Computer Science (Prentice Hall, Englewood Cliffs, NJ, 1991).

[37] B.A. Trakhtenbrot, J.Y. Halpern and A.R. Meyer, From denotational to operational and axiomatic semantics for ALGOL-like languages: an overview, in: E. Clarke and D. Kozen, eds., Proc. Logics of Programs, (Pittsburgh, 1983), Lecture Notes in Computer Science, Vol. 164 (Springer, Berlin, 1983) 474–500.

[38] S. Weeks and M. Felleisen, On the orthogonality of assignments and procedures in Algol, in: *Proc. 20th Annual ACM Symp. on Principles of Programming Languages* (Charleston, 1993) 57–70.

[39] G. Winskel, *Formal Semantics of Programming languages.* Foundations of Computing Series (MIT Press, Cambridge, MA, 1993).