Typed Logics With States

Jan van Eijck, *CWI, P.O. Box 94079, 1090 GB Amsterdam, NL.*
Email: jve@cwi.nl

Abstract

The paper presents a simple format for typed logics with states by adding a function for register update to standard typed lambda calculus. It is shown that universal validity of equality for this extended language is decidable (extending a well-known result of Friedman for typed lambda calculus). This system is next extended to a full fledged typed dynamic logic, and it is illustrated how the resulting format allows for very simple and intuitive representations of dynamic semantics for natural language and denotational semantics for imperative programming. The proposal is compared with some alternative approaches to formulating typed versions of dynamic logics.

Keywords: Type theory, compositionality, denotational semantics, dynamic semantics

1 Introduction

A slight extension to the format of typed lambda calculus is enough to model states (assignments of values to storage cells) in a very natural way. Let a set R of registers or storage cells be given. If we assume that the values to be stored all have type T, then the domain of the states is the set $R \to D_T$. Call this domain $D_\diamond$.

If we want to be able to talk about stores in the calculus itself, we have to do two things:

1. introduce a format for talking about register assignment, and
2. make sure that we cannot assign values to types that are themselves built out of stores.

Point (1) is easy: just add expressions $(r|E)$ to the language, where r is a register or store and E a possible value for that store. The type of $(r|E)$ will be $(\diamond, \diamond)$, for the act of putting the new value E in store cell r effects a mapping from states to states.

To see what (2) is about, note that we our definition of $D_\diamond$ becomes circular if we take $D_\diamond$ to be the set $R \to D_T$, for some type T with D_T defined in terms of $D_\diamond$. To avoid circularity we have to make sure that the types that are built using states are not among the possible values that can be stored in the memory cells.

We need not assume, of course, that all storage cells have the same type, but we must assume that the stored values are all of a type which does not depend on $D_\diamond$. What we need for this is a distinction between standard types (types defined without the help of $\diamond$) and extended types (types construed by means of $\diamond$, possibly among other things). Storage cells or registers are expressions of type $(\diamond, T)$, where T is a standard type. If r is a register of type $(\diamond, T)$, then $[\![r]\!]$, the interpretation of r, is in the domain $D_\diamond \to D_T$, i.e., $[\![r]\!]$ is a map from states to values of the stored type. Thus, under the assumption that storage cells can store items of all standard types, the domain $D_\diamond$ consists of all those functions f from $\cup_T R_{\diamond T}$ to $\cup_T D_T$ with the property that $f(r) \in D_T$ iff r has type $(\diamond, T)$.

We will present the basic format of typed logic with states (TLS) in Section 2. Section 3 presents an equational calculus for TLS, extending the familiar rules for β and η equality with axioms for register lookup (σ equalities), axioms for register update (ρ equalities), and an axiom of register extensionality (τ equality). Section 4 proves that these axioms are sound and complete for standard models. The completeness proof uses a detour via general models along the lines of Friedman [8]. Section 5 extends the familiar $\beta\eta$ reduction from typed logic to $\beta\eta\sigma\rho\tau$ reduction for TLS. Combining the fact that $\beta\eta\sigma\rho\tau$ reduction for TLS is strongly normalizing with the completeness theorem we get the rather surprising result that the relation $\models E = F$ is decidable. Section 6 adds equations to the language as expressions in their own right and defines the boolean operations and the universal and existential quantifiers in terms of those.

The next sections show that TLS can serve as a meeting ground for programming semantics and natural language semantics. Section 7 presents a translation of *while* programs into TLS, and Section 8 discusses various ways of formulating a dynamic semantics for natural language fragments into TLS. Section 9 compares the proposal with some alternative approaches. The final section of the paper puts some further work on the agenda.

2 How to Extend Typed Logics with States

To define *Typed Logic with States* (TLS), let B be a set of basic types. Then the set of standard types over B is given by:

$$T \quad ::= \quad B \mid (T_1, T_2)$$

Extend the set of standard types as follows (it is assumed that $\diamond \notin B$):

$$U \quad ::= \quad \diamond \mid T \mid (U_1, U_2)$$

Call the members of $U - T$ extended types. The extended types are the types in which $\diamond$ occurs. We will often abbreviate a type (U, U') as UU'.

For simplicity, we will not employ constants in the language (constants will be added in the extension of the language defined in Section 6). For every standard type T, registers of type $(\diamond, T)$ are allowed. As explained before, a register of type $(\diamond, T)$ is a store for items of type T. Let $R_{\diamond T}$ be the set of registers of type $(\diamond, T)$ Let R be the set $\bigcup_T R_{\diamond T}$. We assume that there is an enumeration $r_1, r_2, \ldots$ of the members of R.

We allow variables in all types U, indeed we assume a countably infinite supply V_U of them for every type U.

The set of expressions of our language of typed logics with states (TLS) is given by the rule (we use v for the variables and r for the registers):

$$E \quad ::= \quad v \mid r \mid (E_1 E_2) \mid (\lambda v.E) \mid (r|E).$$

For every register r there is a standard type T such that r has type $(\diamond, T)$. The formation of applications $(E_1 E_2)$ is constrained by the requirement that E_1, E_2 must have types (U_1, U_2) and U_1, respectively. If this requirement is met, $(E_1 E_2)$ is well-formed, and its type is U_2. If v has type U_1 and E type U_2, then the type of $(\lambda v.E)$ is (U_1, U_2). The type of $(r|E)$ is $(\diamond, \diamond)$. An expression $(r|E)$ is called a *state changer*, because it is to be interpreted as a function from states to states which changes the input state by assigning a new value E to register r. The formation of state changers $(r|E)$ is constrained by the requirement that if r has type $(\diamond, T)$ then E must have type T. Call this language $L_\diamond$.

If the types of v, E and F are known, the types of $(\lambda v.E)$ and (EF) are uniquely determined, and the type of $(r|E)$ is always $(\diamond, \diamond)$, so we will not always write all subscripts for types. We will also occasionally omit outer parentheses, and write $(\lambda v.E)$ as $\lambda v.E$, and (EF) as EF. An expression $\lambda v_1.(\lambda v_2.(\ldots(\lambda v_n.E)\ldots))$ will be written as $\lambda v_1 v_2 \ldots v_n.E$, and $(\ldots((EF_1)F_2)\ldots F_n)$ as $EF_1 \ldots F_n$, i.e., we assume that application, indicated by parentheses (and), associates to the left. Also, we will let type subscripts do double duty as indices, by writing, e.g., $((r|E_z)E_\diamond)$ instead of $((r|E_z)E'_\diamond)$. It is a feature of the language that expressions of different types are different, so this habit is harmless.

3 An Equational Calculus for TLS

The equational calculus $L_\diamond$ has as its formulas statements of the form $E = F$, with $E, F \in L_\diamond$. The axioms and rules employ the notion of substitution of F for free occurrences of v in E, with notation $E[v := F]$; the formal definition is completely standard. Also, $FV(E)$ is used for the set of variables with free occurrences in E; again, the definition is routine.

Reflexivity, symmetry and transitivity of equality:

$$E = E \qquad \frac{E = F}{F = E} \qquad \frac{E = F \quad F = G}{E = G}$$

Context rules:

$$\frac{E = F}{EG = FG} \qquad \frac{E = F}{GE = GF} \qquad \frac{E = F}{\lambda v.E = \lambda v.F} \qquad \frac{E = F}{(r|E) = (r|F)}$$

β and η axioms:

$$(\lambda v.E)F = E[v := F] \qquad \frac{}{\lambda v.Ev = E}\ v \notin FV(E)$$

σ axioms (for register lookup):

$$r_i((r_i|E)F) = E \qquad \frac{}{r_i((r_j|E)F) = r_i F}\ i \neq j$$

ρ axioms (for register update):

$$(r_i|E)((r_i|F)G) = (r_i|E)G \qquad \frac{}{(r_i|E)((r_j|F)G) = (r_j|F)((r_i|E)G)}\ i \neq j$$

τ axiom (for register extensionality):

$$(r|(rE))E = E.$$

Only the σ, ρ and τ axioms are new.

If $E = F$ can be derived with the rules from the axioms in a finite number of steps, we say that $E = F$ is a theorem of the calculus; we indicate this with $\vdash E = F$.

4 Soundness and Completeness for TLS

First we define full models for TLS. Let non-empty domains D_b be given for all $b \in B$. Then the full model over $\{D_b\}$ is constructed as follows:

$$\begin{aligned} D_{(T_1,T_2)} &:= D_{T_1} \to D_{T_2} \\ D_\diamond &:= \{s \in R \to \bigcup_T D_T \mid \forall r_{\diamond T} \in R\ s(r_{\diamond T}) \in D_T\} \\ D_{(U_1,U_2)} &:= D_{U_1} \to D_{U_2}. \end{aligned}$$

Note that in case all members of R are of the same type T, the domain $D_\diamond$ has the form $R \to D_T$. If $s \in D_\diamond$, $r \in R_{\diamond T}$ and $d \in D_T$, we use $s[r \mapsto d]$ for the function $f \in D_\diamond$ which is given by $f(r') = d$ if $r \approx r'$ and $f(r') = s(r')$ otherwise. Note that $s[r \mapsto d] \in D_\diamond$.

We refer to the full model over $\{D_b\}$ as $M = \{D_b\}$. An assignment in a full model $M = \{D_b\}$ is a function $g : V \to \bigcup_U D_U$ satisfying $g(v) \in D_U$ if $v \in V_U$.

Let g be an assignment for $M = \{D_b\}$. Then the interpretation function $[\![\cdot]\!]_g^M$ in M is defined as follows (note the use of $\boldsymbol{\lambda}$ for 'lambda abstraction in the metalanguage' in what follows):

$$\begin{aligned} [\![v_U]\!]_g^M &:= g(v_U) \\ [\![r_T]\!]_g^M &:= \text{the function given by } \boldsymbol{\lambda} s.s(r) \\ [\![(r_T|E_T)]\!]_g^M &:= \text{the function given by } \boldsymbol{\lambda} s.s[r_T \mapsto [\![E_T]\!]_g^M] \\ [\![(E_{(U_1,U_2)}E_{U_1})]\!]_g^M &:= [\![E_{(U_1,U_2)}]\!]_g^M([\![E_{U_1}]\!]_g^M) \\ [\![(\lambda v_{U_1}.E_{U_2})]\!]_g^M &:= \text{the function given by } \boldsymbol{\lambda} d.[\![E_{U_2}]\!]_{g[v_{U_1} \mapsto d]}^M \end{aligned}$$

Note in particular that the interpretation of a register $r_{\diamond T}$ is indeed a function in $D_\diamond \to D_T$, and that the interpretation of a state changer $(r|E)$ is indeed a function in $D_\diamond \to D_\diamond$.

If M is a full TLS model, we use $M \models E = F$ for: for every assignment g, $[\![E]\!]_g^M = [\![F]\!]_g^M$, and we use $\models E = F$ for: for all full models M it holds that $M \models E = F$.

Proposition 4.1 (Soundness)
$L_\diamond$ is sound for full TLS models: if $\vdash E = F$ then $\models E = F$.

To establish completeness, we make a detour via general TLS models. A general TLS structure is a set of non-empty domains D_U (for every type U), a function $A_\diamond : D_\diamond \times R \to \bigcup_T D_T$, and a set of application functions $A_{U_1 U_2} : D_{U_1 U_2} \times D_{U_1} \to D_{U_2}$, satisfying the following extensionality requirements:

1. if $A_\diamond(s, r) = A_\diamond(s', r)$ for all $r \in R$, then $s = s'$,
2. if $a, b \in D_{U_1 U_2}$ and for every $c \in D_{U_1}$ it holds that $A_{U_1 U_2}(a, c) = A_{U_1 U_2}(b, c)$, then $a = b$.

Note that a full TLS model is a general TLS structure where

$$D_\diamond = \{s \in R \to \bigcup_T D_T \mid \forall r_{\diamond T} \in R\ s(r_{\diamond T}) \in D_T\},$$

with $A_\diamond(s, r)$ given by $s(r)$, and where each $D_{U_1 U_2}$ is the full function space $D_{U_1} \to D_{U_2}$, with $A_{U_1 U_2}(a, b)$ given by $a(b)$.

LEMMA 4.2
Suppose $E \in L_\diamond$. Let

$$\begin{array}{rcl} [E] & := & \{F \mid \vdash E = F\}, \\ D_U & := & \{[E] \mid E \text{ has type } U\}, \\ A_\diamond([E], r) & := & [rE] \quad (E \text{ of type } \diamond,\ r \in R), \\ A_{U_1U_2}([E],[F]) & := & [EF] \quad (E \text{ of type } (U_1, U_2),\ F \text{ of type } U_1). \end{array}$$

Then $M_0 = (\{D_U\}, A_\diamond, \{A_{U_1U_2}\})$ is a general TLS structure.

PROOF. To see that M_0 is well-defined, note that if $\vdash E = F$ then E and F have the same type. Also, if $\vdash E = E'$ and $\vdash F = F'$, then $\vdash (EF) = (E'F')$.

To see that M_0 satisfies the requirements for a general structure, we must check the requirements on the A functions.

1. Assume $[E],[F] \in D_\diamond$, $[E] \neq [F]$. Then: $\vdash E = (r_{e_1}|G_{e_1}) \cdots (r_{e_n}|G_{e_n})E'$, where $n \geq 0$ and E' is such that $[E'] \neq [(r|H)E'']$, for any r, H, E''. To find such a form, just apply the β and η rules to simplify E, and when hitting on a term of the form $(r|H)G$, add $(r|H)$ to the state switcher list and go on with G. This process terminates by the fact that $\beta\eta$ reduction is strongly normalizing, plus the fact that E mentions only finitely many registers. Similarly, we find $F = (r_{f_1}|H_{f_1}) \cdots (r_{f_m}|H_{f_m})F'$, with the same constraints. By the τ axiom we may also assume that no G_{e_i} has the form $r_{e_i}E'$, and similarly, no H_{f_i} has the form $r_{f_i}F'$. By the ρ axioms we may assume that $(e_1, \ldots, e_n)$ and $(f_1, \ldots, f_m)$ are in increasing order and without repetitions.

If $[E'] \neq [F']$ we are done, for then we can take any

$$r \notin \{r_{e_1}, \ldots, r_{e_n}, r_{f_1}, \ldots, r_{f_m}\},$$

and by the second σ axiom we have $[rE] = [rE'] \neq [rF'] = [rF]$, because of the constraint on E', F'.

Now assume that $[E'] = [F']$ and that $(e_1 ... e_n) = (f_1 ... f_m)$. Then, by the fact that $[E] \neq [F]$ there must be a pair $(r_{e_i}|G_{e_i}), (r_{f_j}|H_{f_j})$ with $e_i = f_j$ and $[G_{f_i}] \neq [H_{f_j}]$. In this case we are done, for then by the first σ axiom, $[r_{e_i}E] = [G_{e_i}] \neq [H_{f_j}] = [r_{e_i}F]$.

Finally, assume that $[E'] = [F']$ and that $(e_1 ... e_n) \neq (f_1 ... f_m)$. Then either there is an $e_i \notin (f_1 ... f_m)$ or there is an $f_j \notin (e_1 ... e_n)$. Without loss of generality, assume the former. Then by the σ axioms $[r_{e_i}E] = [G_{e_i}] \neq [r_{e_i}E'] = [r_{e_i}F'] = [r_{e_i}F]$, and we are done.

2. Assume $[E], [E'] \in D_{U_1U_2}$. Suppose that for every $[F] \in D_{U_1}$, we have $[(EF)] = [(E'F)]$. Let $v \in V_{U_1}$ with $v \notin FV(E_1) \cup FV(E_2)$. Then $[(Ev)] = [(E'v)]$, so $\vdash (Ev) = (E'v)$. Therefore, $\vdash \lambda v.Ev = \lambda v.E'v$, and by means of the η axiom and two applications of the transitivity rule we derive from this that $\vdash E = E'$, and therefore $[E] = [E']$. ∎

An assignment in a general structure $(\{D_U\}, A_\diamond, \{A_{U_1U_2}\})$ is a function $g : V \to \bigcup_U D_U$ satisfying $g(v) \in D_U$ if $v \in V_U$.

A general model is a general structure $M = (\{D_U\}, A_\diamond, \{A_{U_1U_2}\})$ together with an interpretation function $[\![\cdot]\!]^M$ defined on the terms of the language, such that $[\![E_U]\!]^M$ is a function from assignments to D_U satisfying the following constraints (we will write $[\![E_U]\!]^M(g)$ as $[\![E_U]\!]^M_g$):

1. $[\![v_U]\!]_g^M = g(v_U)$,
2. for all $s \in D_\diamond$, all standard types T, all $r \in R_{\diamond T}$ it holds that
$$A_{\diamond T}([\![r_{\diamond T}]\!]_g^M, s) = A_\diamond(s, r),$$
3. for all $s \in D_\diamond$, all $r, r' \in R$ it holds that
$$A_\diamond(A_{\diamond\diamond}([\![(r|E)]\!]_g^M, s), r') = \begin{cases} [\![E]\!]_g^M & \text{if } r = r', \\ A_\diamond(s, r') & \text{otherwise,} \end{cases}$$
4. $[\![(E_{(U_1,U_2)} E_{U_1})]\!]_g^M = A_{U_1 U_2}([\![E_{(U_1,U_2)}]\!]_g^M, [\![E_{U_1}]\!]_g)$,
5. for all $d \in D_{U_1}$, $A_{U_1 U_2}([\![(\lambda v_{U_1}.E_{U_2})]\!]_g^M, d) = [\![E_{U_2}]\!]_{g[v_{U_1} \mapsto d]}^M$.

It follows from the requirements on general structures that there can be at most one interpretation function for any general structure M and any assignment g for M. This is so because the requirements on general structures force the values of $[\![r_T]\!]_g^M$, of $[\![(r|E)]\!]_g^M$ and of $[\![(\lambda v_{U_1}.E_{U_2})]\!]_g$ to be unique. If an interpretation for a general structure $M = (\{D_U\}, A_\diamond, \{A_{U_1 U_2}\})$ exists, then we call M a general model. We use $M \models E = F$ just in case every assignment g for general model M satisfies $[\![E]\!]_g^M = [\![F]\!]_g^M$.

A substitution θ is a mapping from variables to terms satisfying the constraint that $\theta(v_U)$ is of type U. This is extended to terms in the standard manner. We will use θE for the result of applying θ to E. Note that E and θE have the same type. Notation for the substitution θ' that differs only from θ in the fact that v is mapped to E is $\theta[v \mapsto E]$. If g is an assignment in the term general structure (i.e., the values $g(v)$ are term equivalence classes), then substitution θ represents g if it holds for all variables v that $\theta(v)$ is a representative of the equivalence class $g(v)$.

LEMMA 4.3
Let θ be a substitution that represents g in the term structure M. Then putting $[\![E]\!]_g^M = [\theta E]$ makes M a general model.

PROOF. To see that $[\![E]\!]_g^M = [\theta E]$ is well-defined, observe that θE_U has type U, so $[\theta E_U] \in D_U$. Next, we must check the requirements on the interpretation function.

1. $[\![v_U]\!]_g^M = [\theta(v_U)] =$ (by the fact that θ represents g) $= g(v_U)$.
2. $[\![r_T]\!]_g^M = [\theta(r_T)] = [r_T]$. By the fact that $A_\diamond([E], r) = [rE] = A_{\diamond T}([r], [E])$ this is indeed the required element of $D_{\diamond T}$.
3. The following reasoning shows that the requirement is met:
$$\begin{aligned} A_\diamond(A_{\diamond\diamond}([\![(r_T|E_T)]\!]_g^M, [E_\diamond]), r') &= A_\diamond(A_{\diamond\diamond}([\theta(r_T|E_T)], [E_\diamond]), r') \\ &= A_\diamond(A_{\diamond\diamond}([(r_T|\theta E_T)], [E_\diamond]), r') \\ &= A_\diamond([(r_T|\theta E_T)E_\diamond], r') \\ &= [r'((r_T|\theta E_T)E_\diamond)] \\ &= [\theta E_T] \text{ if } r = r', [r' E_\diamond] \text{ otherwise.} \end{aligned}$$
4. Here is the reasoning for this case:
$$\begin{aligned} [\![(E_{(U_1,U_2)} E_{U_1})]\!]_g^M &= [\theta(E_{(U_1,U_2)} E_{U_1})] \\ &= [\theta(E_{(U_1,U_2)})\theta E_{U_1}] \\ &= A_{U_1 U_2}([\![E_{(U_1,U_2)}]\!]_g^M, [\![E_{U_1}]\!]_g^M). \end{aligned}$$

5. Let $[E_{U_1}] \in D_{U_1}$. Then:

$$\begin{array}{rcl} A_{U_1U_2}([\![(\lambda v_{U_1}.E_{U_2})]\!]^M_g, [E_{U_1}]) & = & A_{U_1U_2}([\theta(\lambda v_{U_1}.E_{U_2})], [E_{U_1}]) \\ & = & A_{U_1U_2}([\lambda v_{U_1}.\theta(E_{U_2})], [E_{U_1}]) \\ & = & [\theta[v_{U_1} \mapsto E_{U_1}](E_{U_2})] \\ & = & [\![E_{U_2}]\!]^M_{g[v_{U_1} \mapsto [E_{U_1}]]}, \end{array}$$

where the last step is licensed because $\theta[v_{U_1} \mapsto E_{U_1}]$ represents $g[v_{U_1} \mapsto [E_{U_1}]]$.

This completes the check of the requirements and the proof. ■

THEOREM 4.4 (Generalized Completeness)

If for every general model M, $M \models E = F$, then $\vdash E = F$.

PROOF. By means of the construction of a canonical general model M_0. Let g be the identity assignment $v \mapsto [v]$. Then the identity substitution $\theta : v \mapsto v$ represents g, and we have:

$$[E] = [\theta E] = [\![E]\!]^{M_0}_g.$$

By construction we have: if $\not\vdash E = F$, then $M_0 \not\models [E] = [F]$. ■

Let $M = (\{D_U\}, A_\diamond, \{A_{U_1U_2}\})$ and $N = (\{E_U\}, B_\diamond, \{B_{U_1U_2}\})$ be general TLS models. A system $\{f_U\}$ is a *partial homomorphism* of M onto N iff the following hold:

1. Each f_U is a partial surjective map from D_U onto E_U.
2. If $f_\diamond$ is defined for $s \in D_\diamond$, then $f_\diamond(s)$ is the unique element of $E_\diamond$ with
$$f_T(A_\diamond(s, r)) = B_\diamond(f_\diamond(s), r),$$
for all T, all $r \in R_{\diamond T}$.
3. If $f_{U_1U_2}$ is defined for d then $f_{U_1U_2}(d)$ is the unique element of $E_{U_1U_2}$ such that $f_{U_2}(A_{U_1U_2}(d, x)) = B_{U_1U_2}(f_{U_1U_2}(d), f_{U_1}(x))$, for all $x \in \text{dom}\,(f_{U_1})$.
4. For all T, all $r \in R_{\diamond T}$, $f_{\diamond T}(\boldsymbol{\lambda} s.A_\diamond(s, r))$ is defined.
5. If $f_\diamond$ is defined for $s \in D_\diamond$ and f_T is defined for $d \in D_T$, and $r \in R$, then it holds that $f_\diamond$ is defined for $s[r \mapsto d] \in D_\diamond$, and $f_\diamond$ satisfies $f_\diamond(s[r \mapsto d]) = f_\diamond(s)[r \mapsto f_T(d)]$.

Note that a partial homomorphism $\{f_U\}$ is fully determined by $\{f_b \mid b \in B\}$.

PROPOSITION 4.5
If M, N are general models and $\{f_U\}$ a partial homomorphism of M onto N then $f_{\diamond T}(\boldsymbol{\lambda} s.A_\diamond(s, r)) = \boldsymbol{\lambda} s.B_\diamond(s, r)$.

PROOF. Because $\{f_U\}$ is a partial homomorphism, $f_{\diamond T}(\boldsymbol{\lambda} s.A_\diamond(s, r))$ is defined. By property (3) of partial homomorphisms, $f_{\diamond T}(\boldsymbol{\lambda} s.A_\diamond(s, r))$ is the unique element z of $E_{\diamond, T}$ such that

$$\begin{array}{rcl} f_T(A_{\diamond T}(\boldsymbol{\lambda} s.A_\diamond(s, r), s)) & = & f_T(A_\diamond(s, r)) \\ (\text{ property } (2)) & = & B_\diamond(f_\diamond(s), r) \\ & = & B_{\diamond T}(z, f_\diamond(s)), \end{array}$$

for all $s \in \text{dom}\,(f_\diamond)$. Because $f_\diamond$ is onto, it follows that $z = \boldsymbol{\lambda} s.B_\diamond(s, r)$. ■

PROPOSITION 4.6
If M, N are general models, g is an M assignment, h an N assignment, and $\{f_U\}$ a partial homomorphism of M onto N satisfying $f_U(g(v)) = h(v)$ for every U, every $v \in V_U$, then $f_U[\![E]\!]_g^M = [\![E]\!]_h^N$ for every term E of type U.

PROOF. Induction on the structure of E.

For E a variable the property holds by what is given about $\{f_U\}$.

For $r \in R$, we have by Proposition 4.5:

$$\begin{aligned} f_{\diamond T}([\![r]\!]_g^M) &= f_{\diamond T}(\boldsymbol{\lambda} s.A_\diamond(s,r)) \\ &= \boldsymbol{\lambda} s.B_\diamond(s,r) \\ &= [\![r]\!]_h^N. \end{aligned}$$

For expressions of the form (EF) we have:

$$\begin{aligned} f_U([\![EF]\!]_g^M) &= f_U(A_{U'U}([\![E]\!]_g^M, [\![F]\!]_g^M)) \\ &= B_{U'U}([\![E]\!]_h^N, [\![F]\!]_h^N) \\ &= [\![EF]\!]_h^N. \end{aligned}$$

To show $f_{U_1U_2}([\![\lambda v.E]\!]_g^M) = [\![\lambda v.E]\!]_h^N$, take $d \in \text{dom}\,(f_{U_1})$. We must establish that $f_{U_2}(A_{U_1U_2}([\![\lambda v.E]\!]_g^M, d)) = B_{U_1U_2}([\![\lambda v.E]\!]_h^N, f_{U_1}(d))$.

$$\begin{aligned} f_{U_2}(A_{U_1U_2}([\![\lambda v.E]\!]_g^M, d)) &= f_{U_2}([\![E]\!]_{g[v\mapsto d]}^M) \\ &= [\![E]\!]_{h[v\mapsto f_{U_1}(d)]}^N \\ &= B_{U_1U_2}([\![\lambda v.E]\!]_h^N, f_{U_1}(d)). \end{aligned}$$

Finally, to show $f_{\diamond\diamond}([\![r|E]\!]_g^M) = [\![r|E]\!]_h^N$, take $s \in D_\diamond$. We must show that

$$f_\diamond(A_{\diamond\diamond}([\![r|E]\!]_g^M, s)) = B_{\diamond\diamond}([\![r|E]\!]_h^N, f_\diamond(s)).$$

$$\begin{aligned} f_\diamond(A_{\diamond\diamond}([\![r|E]\!]_g^M, s)) &= f_\diamond(s[r \mapsto [\![E]\!]_g^M]) \\ &= f_\diamond(s)[r \mapsto f_\diamond([\![E]\!]_g^M) \\ &= f_\diamond(s)[r \mapsto [\![E]\!]_h^N] \\ &= B_{\diamond\diamond}([\![r|E]\!]_h^N, f_\diamond(s)). \end{aligned}$$

∎

PROPOSITION 4.7
If there is a partial homomorphism from M onto N, then $M \models E = F$ implies $N \models E = F$.

PROOF. Use Proposition 4.6. ∎

PROPOSITION 4.8
If $N = (\{E_U\}, A_\diamond, \{A_{U_1U_2}\})$ is a general model and $M = \{D_U\}$ is a full model, and moreover $|E_b| \leq |D_b|$ for $b \in B$, then there is a partial homomorphism from M onto N.

PROOF. Let $\{f_b\}$ be a set of arbitrary partial surjective maps from D_b to E_b. First extend $\{f_b\}$ to all standard types, as follows. Suppose f_{T_1} and f_{T_2} have been defined. Define $f_{T_1T_2}(d)$ to be the unique element of $E_{T_1T_2}$ (if it exists) such that $f_{T_2}(d(y)) = A_{T_1T_2}(f_{T_1T_2}(d), f_{T_1}(y))$, for all $y \in \text{dom}\,(f_{T_1})$.

To see that $f_{T_1T_2}$ is surjective, take $z \in E_{T_1T_2}$, and let $x \in D_{T_1T_2}$ be such that for all $y \in \text{dom}\,(f_{T_1})$, $x(y) \in f_{T_2}^{-1}(A_{T_1T_2}(z, f_{T_1}(y)))$. (This uses the surjectivity of f_{T_2}.) Then $f_{T_1T_2}(x) = z$.

Next define the map $f_\diamond$, by putting $f_\diamond(s) =$ the unique element of $E_\diamond$ (if it exists) with $f_T(s(r)) = A_\diamond(f_\diamond(s), r)$, for all standard types T, all $r \in R_{\diamond T}$. We can check that this map is surjective, as before.

Extend the map to all types U, in the same manner as before, and check surjectivity as before.

All f_U are surjective, and satisfy properties (2) and (3) by definition. Because M is full, property (4) boils down to: $f_{\diamond T}(\boldsymbol{\lambda} s.s(r))$ is defined. To check this property, we have to show that there is a unique $z \in E_{\diamond T}$ with $f_T(y(r)) = A_{\diamond T}(z, f_\diamond(y))$, for all $y \in \text{dom}\,(f_\diamond)$. Clearly, z is given by $\boldsymbol{\lambda} s.A_\diamond(s, r)$.

Finally, we check property (5). Assume $f_\diamond$ is defined for $s \in D_\diamond$, and f_T is defined for $d \in D_T$. Assume $r \in R_{\diamond T}$. We check whether $f_\diamond(s[r \mapsto d])$ is defined. By the construction of $f_\diamond$, this is the case iff there is a unique $z \in E_\diamond$ with $f_T(s[r \mapsto d](r') = A_\diamond(z, r')$ for all standard T, all $r' \in R_{\diamond T}$. Clearly, this z is given by $\boldsymbol{\lambda} s.s[r \mapsto f_T(d)]$. ■

THEOREM 4.9 (Full Completeness)
If $\models E = F$, then $\vdash E = F$.

PROOF. Suppose $\not\vdash E = F$. Then, by the generalized completeness theorem, $M_0 \not\models E = F$, where M_0 is the canonical general model. By proposition 4.8 there is a full model M of which M_0 is a partial homomorphic image. By proposition 4.7, $M \not\models E = F$. ■

Note that for all full models M with infinite base domains, the relation $M \models E = F$ coincides with the relation $M_0 \models E = F$, where M_0 is our canonical term model. This is because any full model with large enough base domains has the canonical model as a partial homomorphic image. Indeed, any equality that is true in a large enough full model will be true in the canonical term model. The situation is completely analogous to the case of the ordinary typed lambda calculus (see Friedman [8]).

5 Reducing TLS Expressions

If R is a relation on the set of expressions of $L_\diamond$ (a so-called notion of reduction), then R determines a relation $\xrightarrow{R}$ of one-step R reduction in the following standard manner:

$$\frac{(E, E') \in R}{E \xrightarrow{R} E'} \qquad \frac{E \xrightarrow{R} E'}{(FE) \xrightarrow{R} (FE')} \qquad \frac{E \xrightarrow{R} E'}{(EF) \xrightarrow{R} (E'F)}$$

$$\frac{E \xrightarrow{R} E'}{(\lambda v.E) \xrightarrow{R} (\lambda v.E')} \qquad \frac{E \xrightarrow{R} E'}{(r|E) \xrightarrow{R} (r|E')}$$

R reduction (notation $\overset{R}{\twoheadrightarrow}$) is the reflexive transitive closure of $\xrightarrow{R}$:

$$\frac{E \xrightarrow{R} E'}{E \overset{R}{\twoheadrightarrow} E'} \qquad E \overset{R}{\twoheadrightarrow} E \qquad \frac{E \overset{R}{\twoheadrightarrow} E' \quad E' \overset{R}{\twoheadrightarrow} E''}{E \overset{R}{\twoheadrightarrow} E''}$$

Recall that the notion of beta reduction is the relation between an expression of the form $((\lambda v.E)E')$ and the expression $E[v := E']$. β reduction is the relation $\overset{\beta}{\twoheadrightarrow}$. Similarly, the notion of η reduction is the relation between $\lambda v.Ev$ and E (provided $v \notin FV(E)$). We add three new notions of reduction to this list:

- The notion of σ reduction is the union of the relations

  $$(r_i((r_i|E)F), E)$$

  and

  $$(r_i((r_j|E)F), r_iF), \text{ provided } i \neq j.$$

- The notion of ρ reduction is the union of the relations

  $$((r_i|E)((r_i|F)G), (r_i|E)G)$$

  and

  $$((r_i|E)((r_j|F)G), (r_j|F)((r_i|E)G)), \text{provided } j < i.$$

- The notion of τ reduction is the relation $((r|(rE))E, E)$.

From the soundness of the $L_\diamond$ calculus it follows that β reduction, η reduction, σ reduction, ρ reduction and τ reduction are all sound (after all, these reduction notions are nothing but directed versions of the corresponding equality axioms).

The standard reduction notion for typed logic is $\beta\eta$ reduction i.e., reduction for the notion $\beta \cup \eta$), and the standard results of typed logic that $\beta\eta$ reduction is confluent and strongly normalizing extend readily to the setting of TLS:

PROPOSITION 5.1
$\beta\eta$ reduction is confluent, i.e., $E \overset{\beta\eta}{\twoheadrightarrow} F$ and $E \overset{\beta\eta}{\twoheadrightarrow} F'$ together imply that there is a G with $F \overset{\beta\eta}{\twoheadrightarrow} G$ and $F' \overset{\beta\eta}{\twoheadrightarrow} G$.

PROPOSITION 5.2
$\beta\eta$ reduction is strongly normalizing for TLS, i.e., every reduction sequence $E \xrightarrow{\beta\eta} F \xrightarrow{\beta\eta} \cdots$ terminates.

Let $\overset{\sigma\rho\tau}{\twoheadrightarrow}$ be the relation of R reduction for $R = \sigma \cup \rho \cup \tau$.

PROPOSITION 5.3
$\overset{\sigma\rho\tau}{\twoheadrightarrow}$ is weakly confluent, i.e., $E \xrightarrow{\sigma\rho\tau} F$ and $E \xrightarrow{\sigma\rho\tau} F'$ together imply that there is a G with $F \overset{\sigma\rho\tau}{\twoheadrightarrow} G$ and $F' \overset{\sigma\rho\tau}{\twoheadrightarrow} G$.

PROOF. The claim is proved by a case analysis. We just give one case in full. The reasoning for the other cases is similar.

Assume $E[r((r|E_T)E_\diamond)] \xrightarrow{\sigma\rho\tau} E[E_T]$, and $E[r(r|E_T)E_\diamond] \xrightarrow{\sigma\rho\tau} F$, with $F \not\approx E[E_T]$. Then there are four sub-cases to consider.

1. The reduction $E[r((r|E_T)E_\diamond)] \xrightarrow{\sigma\rho\tau} F$ leaves subexpression $(r(r|E_T)E_\diamond)$ unaffected, i.e., we can write F as $E'[r(r|E_T)E_\diamond]$. In this case both F and $E[E_T]$ reduce in one σ step to $E'[E_T]$.

2. The reduction $E[r((r|E_T)E_\diamond)] \xrightarrow{\sigma\rho\tau} F$ removes subexpression $(r(r|E_T)E_\diamond)$. In this case $E[E_T]$ reduces in one σ step to F.
3. The reduction $E[r((r|E_T)E_\diamond)] \xrightarrow{\sigma\rho\tau} F$ affects E_T. In this case, F can be written as $E[r(r|E'_T)E_\diamond]$, and both F and $E[E_T]$ reduce in one σ step to $E[E'_T]$.
4. The reduction $E[r((r|e_T)E_\diamond)] \xrightarrow{\sigma\rho\tau} F$ affects $E_\diamond$. In this case F reduces in one σ step to $E[E_T]$.

∎

PROPOSITION 5.4
$\sigma\rho\tau$ reduction is strongly normalizing, i.e., every reduction sequence $E \xrightarrow{\sigma\rho\tau} F \xrightarrow{\sigma\rho\tau} \cdots$ terminates.

PROOF. Every $\sigma\rho\tau$ reduction step either reduces the number of symbols in an expression, or (ρ steps of the second kind) is a step towards putting a sequence of registers in increasing order. ∎

PROPOSITION 5.5
$\overset{\sigma\rho\tau}{\twoheadrightarrow}$ is confluent, i.e., $E \overset{\sigma\rho\tau}{\twoheadrightarrow} F$ and $E \overset{\sigma\rho\tau}{\twoheadrightarrow} F'$ together imply that there is a G with $F \overset{\sigma\rho\tau}{\twoheadrightarrow} G$ and $F' \overset{\sigma\rho\tau}{\twoheadrightarrow} G$.

PROOF. By an application of Newman's lemma (see Klop [17]) from Propositions 5.3 and 5.4. ∎

Let $\beta\eta\sigma\rho\tau$ reduction be the relation $\overset{R}{\twoheadrightarrow}$ for $R = \beta \cup \eta \cup \sigma \cup \rho \cup \tau$. It follows immediately from the soundness of $L_\diamond$ that $\beta\eta\sigma\rho\tau$ reduction is sound.

Here is an example $\beta\sigma\tau$ reduction:

$$\begin{array}{rcl}
(\lambda p.(p\ (r|x)i))(\lambda i.((r'|(r'x))(ri))) & \xrightarrow{\beta} & ((\lambda i.((r'|(r'x))(ri)))\ (r|x)i) \\
& \xrightarrow{\beta} & ((r'|(r'x))(r\ (r|x)i)) \\
& \xrightarrow{\sigma} & ((r'|(r'x))x) \\
& \xrightarrow{\tau} & x.
\end{array}$$

To prove that $\beta\eta\sigma\rho\tau$ reduction is confluent we again need to make a detour via weak confluence. This is because the combination of $\beta\eta$ reduction and $\sigma\rho\tau$ reduction is not orthogonal (see Klop [17]). Intuitively, orthogonality of two reduction relations R and S means that R reduction steps never 'spoil' opportunities for S reduction and vice versa. If two confluent reduction relations are orthogonal, then their union is again confluent. In our case, orthogonality fails due to the presence of τ reductions (an opportunity to apply $(r|(rE))E \to E$ may in principle be spoiled by a reduction $E \to E'$).

PROPOSITION 5.6
$\beta\eta\sigma\rho\tau$ reduction is weakly confluent, i.e., $E \xrightarrow{\beta\eta\sigma\rho\tau} F$ and $E \xrightarrow{\beta\eta\sigma\rho\tau} F'$ together imply that there is a G with $F \overset{\beta\eta\sigma\rho\tau}{\twoheadrightarrow} G$ and $F' \overset{\beta\eta\sigma\rho\tau}{\twoheadrightarrow} G$.

PROOF. A case analysis similar to the analysis in the proof of Proposition 5.3. ∎

PROPOSITION 5.7
$\xrightarrow{\beta\eta\sigma\rho\tau}$ is strongly normalizing, i.e., every reduction sequence $E \xrightarrow{\beta\eta\sigma\rho\tau} F \xrightarrow{\beta\eta\sigma\rho\tau} \cdots$ terminates.

PROOF. The proof is a straightforward adaptation of the proof of strong normalization for typed lambda calculus. See Barendregt [1], Appendix A, or Hindley and Seldin [14]. ∎

PROPOSITION 5.8
$\stackrel{\beta\eta\sigma\rho\tau}{\twoheadrightarrow}$ is confluent, i.e., $E \stackrel{\beta\eta\sigma\rho\tau}{\twoheadrightarrow} F$ and $E \stackrel{\beta\eta\sigma\rho\tau}{\twoheadrightarrow} F'$ together imply that there is a G with $F \stackrel{\beta\eta\sigma\rho\tau}{\twoheadrightarrow} G$ and $F' \stackrel{\beta\eta\sigma\rho\tau}{\twoheadrightarrow} G$.

PROOF. Again by an application of Newman's lemma from Propositions 5.6 and 5.7. ∎

THEOREM 5.9
The relation $\vdash E = F$ is recursive.

PROOF. Immediate from the fact that $\beta\eta\sigma\rho\tau$ reduction is strongly normalizing. Just reduce E and F to find out if their normal forms are the same (modulo changes in bound variables). ∎

THEOREM 5.10
The relation $\models E = F$ is recursive.

PROOF. Immediate from Theorem 5.9 and the Completeness Theorem 4.9. ∎

It should be noted that deciding equality in TLS is not cheap, for Statman's result that the typed lambda calculus is not elementary recursive [26] applies to TLS as well.

6 The Logical Theory of TLS

Equations in typed logic have the form $E = F$, where E, F are assumed to be of the same type. If we assume a basic type t, with $D_t = \{1, 0\}$, we can consider equations as terms of type t, by extending the language with a rule:

$$E_t \quad ::= \quad E_U = E'_U$$

Note that this language extension boils down to adding constants $Q_{(U,(U,t))}$ for every type U, and writing the term $((Q_{(U,(U,t))}E_U)F_U)$ as $E_U = F_U$. The interpretation of $E_U = F_U$ in (standard or general) model M, for assignment g, is given by (note the use of **equal**s for 'identity at the meta-level'):

$$[\![E_U = F_U]\!]^M_g \quad := \quad \begin{cases} 1 & \text{if } [\![E_U]\!]^M_g \textbf{ equals } [\![F_U]\!]^M_g, \\ 0 & \text{if } [\![E_U]\!]^M_g \text{ does not } \textbf{equal } [\![F_U]\!]^M_g. \end{cases}$$

We agree to write $E_t = F_t$ as $E \leftrightarrow F$. Call an expression of type t a formula. Some useful abbreviations for formulas and formula-forming operators can now be given:

$$\begin{array}{rcl} \top & := & (\lambda x_t.x_t) = (\lambda x_t.x_t) \\ \bot & := & (\lambda x_t.x_t) = (\lambda x_t.\top) \\ \neg & := & (\lambda x_t.(x_t \leftrightarrow \bot)) \\ \wedge & := & (\lambda x_t y_t.(\lambda z_{(t,t)}.((zx) \leftrightarrow y)) = (\lambda z_{(t,t)}.(z\top))) \\ \forall x_U E & := & (\lambda x_U.E) = (\lambda x_U.\top) \end{array}$$

Write $(E \wedge F)$ for $((\wedge E)F)$, and abbreviate $\neg(\neg E \wedge \neg F)$ as $(E \vee F)$, $\neg(E \wedge \neg F)$ as $(E \rightarrow F)$, and $\neg\forall x_U \neg E$ as $\exists x_U E$. $\forall x_1(\forall x_2 \ldots (\forall x_n E \ldots))$ will be written as $\forall x_1 x_2 \ldots x_n E$, and similarly for $\exists$. We will also omit brackets in n-ary conjunctions and disjunctions, so we write $E_1 \wedge \ldots \wedge E_n$ and $E_1 \vee \ldots \vee E_n$.

A formula E_t is valid if $[\![E_t]\!]_g^M = 1$ for every choice of M, g (M a standard model).

A logical calculus for this extended TLS language can be defined by adding the following axioms and rules to the equational calculus (see e.g. Gallin [9]).

Axiom scheme for the Booleans (giving an explicit definition of D_t as $\{0, 1\}$):

$$(E_{tt}\top \wedge E_{tt}\bot) \leftrightarrow \forall x_t E_{tt} x.$$

Extensionality expressed by means of universal quantification:

$$\forall x(Ex = Fx) \leftrightarrow (E = F).$$

Axiom schemes for fixing the meanings of the constants $Q_{(U,(U,t))}$:

$$v_U = w_U \rightarrow E_{Ut} v_U \leftrightarrow E_{Ut} w_U.$$

Inference rule (B_t' is the result of replacing an occurrence of E_U in B_t by E_U'):

$$\frac{E_U = E_U' \quad B_t}{B_t'}$$

This logic is not complete for full models, for the extended language is expressive enough to define the standard model of arithmetic, and the existence of a complete logic for TLS would contradict Tarski's theorem of the non-axiomatizability of arithmetical truth.

Still, we know that the incomplete logic of types and states contains the TLS equational calculus. Also, we have completeness for general models (as can be shown by a standard extension of the completeness proof given above; see also Henkin [13]). This means that universal validities that continue to hold in all general models are provable in the logic.

As an example, we mention that for every register $r \in R_{\diamond T}$ of the language the following formula is valid and provable:

$$\forall i_\diamond \forall x_T \exists j_\diamond (j = (r|x)i).$$

This is certainly valid, by virtue of the definition of the domain $D_\diamond$. The principle is also provable, for it continues to hold in general models.

The truth of the formula illustrates that the construction of the typed domains itself guarantees that there are 'enough states', and that no update axiom in the style of [15] is needed, or rather, that such an update axiom is 'provided algebraically' by the $(r|E)$ functions and the σ, ρ, τ principles governing their behaviour.

It is instructive to check how the notion of equality works out for the types $(\diamond, T)$. Note that the following formula is valid (and indeed an axiom of the TLS logic):

$$(E_{\diamond T} = F_{\diamond T}) \leftrightarrow \forall x_\diamond (Ex = Fx).$$

This axiom says that two functions g, h in $D_\diamond \rightarrow D_T$ are equal iff for every $s \in D_\diamond$, $g(s) = h(s)$.

PROPOSITION 6.1
Let $r, r' \in R_{\diamond T}$. On the assumption that $|D_T| \geq 2$, the formula $r = r'$ evaluates to true in general model M iff r and r' are the same register.

PROOF. Let M be a general model. According to the rule for interpreting equalities, $[\![r = r']\!]_g^M = 1$ iff $[\![r]\!]_g^M(s) = [\![r']\!]_g^M(s)$ for all $s \in D_\diamond$. By the evaluation clause for store registers, this requirement boils down to: $A_{\diamond T}(r, s) = A_{\diamond T}(r', s)$ for all $s \in D_\diamond$. By requirement (2) on interpretation in general models, this is the case iff $A_\diamond(s, r) = A_\diamond(s, r')$ for all $s \in D_\diamond$. If $r \approx r'$ (r and r' are the same register), then this requirement is always met, of course. If $r \not\approx r'$, then we have two cases.

- If we assume that D_T contains at most one element, then $A_\diamond$ cannot assign different objects to r, r', for any $s \in D_\diamond$, in the first place. Therefore, in this case the equality relation holds between $[\![r]\!]_g^M$ and $[\![r']\!]_g^M$ for any pair r, r'.
- Suppose D_T contains at least two objects (call them d_1 and d_2). Assume $s \in D_\diamond$ with $A_\diamond(s, r) = d_1$. Suppose $g(v) = d_2$, and let $s' := A_{\diamond\diamond}([\![r'|v]\!]_g^M, s)$. Then it follows from requirement (3) on interpretations in general models that $A_\diamond(s', r) = d_1$ and $A_\diamond(s', r') = d_2$. Therefore, in this case $[\![r]\!]_g^M$ and $[\![r']\!]_g^M$ are not equal.

∎

The proposition shows that equalities for registers behave in a reasonable fashion.

7 Denotational Semantics in TLS

The assignment statement $r := t$ is the basic building block of imperative programming. Therefore, an analysis of imperative programs in a language that has register assignment in its algebraic basis is in a sense more natural than analysing the assignment statement $r := t$ in a more roundabout way. In this section we demonstrate the semantic analysis of *while* programs in TLS.

Let z be the type of integers, and assume that r ranges over $R_{\diamond z}$. Then the following defines the language of *while* programs with program variables taken from the set $R_{\diamond z}$.

$$\begin{array}{rcl}
N & ::= & \mathbf{0} \mid \cdots \mid \mathbf{9} \mid N\mathbf{0} \mid \cdots \mid N\mathbf{9} \\
A & ::= & r \mid N \mid (A_1 + A_2) \mid (A_1 - A_2) \mid (A_1 \diamond A_2) \\
B & ::= & A_1 = A_2 \mid A_1 < A_2 \mid A_1 \leq A_2 \mid \neg B \mid (B_1 \wedge B_2) \\
S & ::= & r := A \mid \text{skip} \mid (S_1; S_2) \\
 & & \mid \text{if } B \text{ then } S_1 \text{ else } S_2 \mid \text{while } B \text{ do } S
\end{array}$$

The meanings of the statements of the *while* language are partial functions over $R_{\diamond z} \to \mathbb{Z}$, i.e., members of $(R_{\diamond z} \to \mathbb{Z}) \hookrightarrow (R_{\diamond z} \to \mathbb{Z})$, where $\mathbb{Z}$ is used for the set of integers. Recall that if we build a TLS logic over basic type set $\{z, t\}$, and assume that $R_{\diamond z}$ is the set of all registers, then the domain $D_\diamond$ has the form $R_{\diamond z} \to \mathbb{Z}$. Thus, the members of $(R_{\diamond z} \to \mathbb{Z}) \hookrightarrow (R_{\diamond z} \to \mathbb{Z})$ are in fact members of $D_\diamond \hookrightarrow D_\diamond$. We can represent a member of $D_\diamond \hookrightarrow D_\diamond$ (a partial function from states to states) by means of its graph, which is a member of $D_{(\diamond,(\diamond,t))}$. Meanings of *while* statements can be represented as members of $D_{(\diamond,(\diamond,t))}$. We will now show how one can refer to them by means of TLS expressions in a straightforward way.

First define a relation $\sqsubseteq$ on $D_{(\diamond,(\diamond,t))}$ by means of:

$$F \sqsubseteq G \iff \forall ij(Fij \to Gij),$$

where F, G are expressions of type $(\diamond,(\diamond,t))$, and i, j variables of type $\diamond$.

Next, define a function FIX in

$$(D_{(\diamond,(\diamond,t))} \to D_{(\diamond,(\diamond,t))}) \to (D_{(\diamond,(\diamond,t))} \to D_t)$$

by means of:

$$\text{FIX} := \lambda Fg.\forall ij(Fgij = gij),$$

where F is a variable of type $((\diamond,(\diamond,t)),(\diamond,(\diamond,t)))$, g a variable of type $(\diamond,(\diamond,t))$, and i, j are variables of type $\diamond$.

If H has type $D_{(\diamond,(\diamond,t))} \to D_{(\diamond,(\diamond,t))}$ and h type $D_{(\diamond,(\diamond,t))}$, then FIX Hh has type D_t. The expression FIX Hh reduces to:

$$\forall su(Hhsu = hsu),$$

which is true iff (the interpretation of) h is the graph of a fixed point of (the interpretation of) H.

Finally, define a function μ in

$$(D_{(\diamond,(\diamond,t))} \to D_{(\diamond,(\diamond,t))}) \to D_{(\diamond,(\diamond,t))}$$

by means of:

$$\mu := \lambda Fij.\exists g(\text{FIX } Fg \wedge \forall d(\text{FIX } Fd \to g \sqsubseteq d) \wedge gij),$$

where F a variable of type $((\diamond,(\diamond,t)),(\diamond,(\diamond,t)))$, g, d are variables of type $(\diamond,(\diamond,t))$, and i, j variables of type $\diamond$.

If H has type $D_{(\diamond,(\diamond,t))} \to D_{(\diamond,(\diamond,t))}$, then μH has type $D_{(\diamond,(\diamond,t))}$, and μH reduces to

$$\lambda su.\exists g(\text{FIX } Hg \wedge \forall d(\text{FIX } Hd \to g \sqsubseteq d) \wedge gsu).$$

This expression denotes the graph of the least fixed point of (the interpretation of) H, if it exists, and the empty relation on $D_\diamond$ otherwise.

The translation of *while* to TLS now proceeds in four stages: first we translate numerals into type z, next arithmetical expressions into type $(\diamond, z)$, then boolean expressions into type $(\diamond, t)$, and finally statements into type $(\diamond,(\diamond,t))$.

Translation of numerals into type z (we assume that we have a 0 for zero, that s names the successor function, that $+, -, \times$ are names of the functions for addition, subtraction and multiplication in $\mathbb{Z}$, and we abbreviate $s0$ as 1, ..., $sssssssss0$ as 9 and $ssssssssss0$ as 10.

$$\begin{array}{rcl}
\mathbf{0}^\circ & := & 0 \\
 & \vdots & \\
\mathbf{9}^\circ & := & 9 \\
(N\mathbf{0})^\circ & := & N^\circ \times 10 \\
 & \vdots & \\
(N\mathbf{9})^\circ & := & (N^\circ \times 10) + 9
\end{array}$$

Translation of arithmetical expressions into type $(\diamond, z)$, using the functions $+, -, \times$ on the domain $\mathbb{Z}$:

$$\begin{array}{rcl} N^\bullet & := & \lambda i.N^\circ \\ r^\bullet & := & r \\ (A_1 + A_2)^\bullet & := & \lambda i.(A_1^\bullet i + A_2^\bullet i) \\ (A_1 - A_2)^\bullet & := & \lambda i.(A_1^\bullet i - A_2^\bullet i) \\ (A_1 * A_2)^\bullet & := & \lambda i.(A_1^\bullet i \times A_2^\bullet i) \end{array}$$

Note that the translation of r is r itself. This is correct, for r is a variable in the programming language, but a register in the TLS language, and it has type $(\diamond, z)$.

Translation of boolean expressions into type $(\diamond, t)$. We already have $=$ in type $(z, (z, t))$. Here we assume that we also have a relation $<$ in $(z, (z, t))$ available. $\leq$ is then defined as $\lambda PQ.(P < Q \vee P = Q)$.

$$\begin{array}{rcl} (A_1 = A_2)^\bullet & := & \lambda i.(A_1^\bullet i = A_2^\bullet i) \\ (A_1 < A_2)^\bullet & := & \lambda i.(A_1^\bullet i < A_2^\bullet i) \\ (A_1 \leq A_2)^\bullet & := & \lambda i.(A_1^\bullet i \leq A_2^\bullet i) \\ (\neg B)^\bullet & := & \lambda i.\neg(B^\bullet i) \\ (B_1 \wedge B_2)^\bullet & := & \lambda i.(B_1^\bullet i \wedge B_2^\bullet i) \end{array}$$

Translation of statements into type $(\diamond, (\diamond, t))$:

$$\begin{array}{rcl} (r := A)^\bullet & := & \lambda ij.((r|(A^\bullet i))i = j) \\ \text{skip}^\bullet & := & \lambda ij.(i = j) \\ (S_1; S_2)^\bullet & := & \lambda ij.\exists k(S_1^\bullet ik \wedge S_2^\bullet kj) \\ (\text{if } B \text{ then } S_1 \text{ else } S_2)^\bullet & := & \lambda ij.((B^\bullet i \wedge S_1^\bullet ij) \vee (\neg B^\bullet i \wedge S_2^\bullet ij)) \\ (\text{while } B \text{ do } S)^\bullet & := & (\mu\lambda gij.((B^\bullet i \wedge \exists k(S^\bullet ik \wedge gkj)) \\ & & \qquad \vee(\neg B^\bullet i \wedge i = j)))) \end{array}$$

PROPOSITION 7.1
The translation $S^\bullet$ is correct in the sense that the interpretation of $S^\bullet$ corresponds to the semantic function specified for S by the standard semantics for the *while* language.

PROOF. First specify the semantics of *while* by some other means, e.g., in operational style, by means of transition rules, and then engage in a lengthy induction exercise (see e.g. Plotkin [24], or the textbook presentation in Nielson and Nielson [23]). ∎

The proposition shows that TLS provides a cheap way of representing the meanings of *while* programs (by means of simpleton domains, so to speak).

The purpose of this section was to demonstrate that reflexive domains (domains D that are isomorphic to the function space $[D \to D]$ of all continuous functions on D, proposed as a denotational semantics for programming in Scott and Strachey [25]) are by no means essential for providing a denotational semantics of recursion in imperative programming. The same point is made at greater length in Muskens [22], but with the help of a higher order logic that introduces registers by means of logical axioms, while we have register assignment in the algebraic basis of our set-up.

8 Representing Dynamic NL Semantics

Let D_e be a domain of basic entities, and assume that all registers are of type $R_{\diamond e}$, and therefore all state changers are of the form $(r_{\diamond e}|E_e)$. This assumption entails that the domain $D_\diamond$ has the form $R_{\diamond e} \to D_e$. We consider the language L of dynamic predicate logic [11] given by:

$$\begin{array}{rcl} t & ::= & r \mid c \\ \phi & ::= & r \mid P^n t_1 \cdots t_n \mid (\phi_1 ; \phi_2) \mid \neg \phi \end{array}$$

Dynamic implication $\phi_1 \Rightarrow \phi_2$ is defined as $\neg(\phi_1 ; \neg\phi_2)$, and the hackneyed example 'If a farmer owns a donkey he beats it' gets the following rendering:

$$(r_1; Fr_1; r_2; Dr_2; Or_1 r_2) \Rightarrow Br_1 r_2.$$

There are various ways of specifying a dynamic semantics for such a language, and these different specifications suggest different translations into TLS.

Perhaps the simplest presentation of the semantics of L is as a relation $[\![\cdot]\!]$ on the set $D_\diamond$. Assume a domain of discourse D_e and an interpretation I for the predicates P and the constants c_e. Let s, s', s'' range over $D_\diamond$. If $s \in D_\diamond$, then I_s is the interpretation function on terms given by $I_s(c) := I(c)$, $I_s(r) := s(r)$. The interpretation relation $[\![\cdot]\!]$ for the formulas of dynamic predicate logic is given by:

$$\begin{array}{rcl} s[\![r]\!]s' & \text{iff} & s' = s[r \mapsto d] \text{ for some } d \text{ in } D_e \\ s[\![P^n t_1 \cdots t_n]\!]s' & \text{iff} & s = s' \text{ and } (I_s t_1, \cdots, I_s t_n) \in I(P) \\ s[\![\phi_1 ; \phi_2]\!]s' & \text{iff} & \text{there is an } s'' \text{ with } s[\![\phi_1]\!]s'' \text{ and } s''[\![\phi_2]\!]s' \\ s[\![\neg\phi]\!]s' & \text{iff} & s = s' \text{ and there is no } s'' \text{ with } s[\![\phi]\!]s'' \end{array}$$

This semantic specification suggests a straightforward translation into TLS, where the translations for terms get type $(\diamond, e)$ and those for dynamic formulas get type $(\diamond, (\diamond, t))$.

In the translation $^\circ$ for terms we map registers to themselves, ordinary constants to constant functions in $(\diamond, e)$:

$$\begin{array}{rcl} r^\circ & := & r \\ c^\circ & := & \lambda i.c \end{array}$$

The translation $^\bullet$ for formulas is a completely straightforward rendering of the semantic specifications in TLS:

$$\begin{array}{rcl} r^\bullet & := & \lambda ij.\exists x_e((r|x_e)i = j) \\ (P^n t_1 \cdots t_n)^\bullet & := & \lambda ij.(i = j \wedge P(t_1^\circ i) \cdots (t_n^\circ i)) \\ (\phi_1 ; \phi_2)^\bullet & := & \lambda ij.\exists k(\phi_1^\bullet ik \wedge \phi_2^\bullet kj) \\ (\neg\phi)^\bullet & := & \lambda ij.(i = j \wedge \neg\exists k \phi^\bullet ik) \end{array}$$

This is essentially the translation given in [21].

The dynamic semantics above can also, equivalently, be given in functional style, as a mapping from sets of states to sets of states, as follows:

$$[\![r]\!](A) \quad := \quad \{s[r \mapsto d] \mid s \in A, d \in D_e\}$$

$$
\begin{aligned}
[\![P^n t_1 \cdots t_n]\!](A) &:= \{s \in A \mid (I_s t_1, \cdots, I_s t_n) \in I(P)\} \\
[\![\phi_1 ; \phi_2]\!](A) &:= [\![\phi_2]\!]([\![\phi_1]\!](A)) \\
[\![\neg\phi]\!](A) &:= \{s \in A \mid [\![\phi]\!](\{s\}) = \emptyset\}
\end{aligned}
$$

This suggests a translation in type $((\diamond, t)(\diamond, t))$, as follows (assume p is a variable of type $(\diamond, t)$):

$$
\begin{aligned}
r^\diamond &:= \lambda p \lambda i. \exists j \exists x (pj \wedge (r|x)j = i) \\
(P^n t_1 \cdots t_n)^\diamond &:= \lambda p \lambda i. (pi \wedge P(t_1^\circ i) \cdots (t_n^\circ i)) \\
(\phi_1 ; \phi_2)^\diamond &:= \lambda p. \phi_2^\diamond (\phi_1^\diamond p) \\
(\neg\phi)^\diamond &:= \lambda p \lambda i. (pi \wedge \neg \exists k \phi^\diamond (\lambda j. j = i) k)
\end{aligned}
$$

Less obviously, it is also possible to specify the semantics for L as a function mapping states to sets of sets of states:

$$
||\phi||(s) \quad := \quad \{A \subseteq D_\diamond \mid s' \in A \text{ for some } s' \text{ with } s[\![\phi]\!]s'\}.
$$

This is the basis for the system of Dynamic Montague Grammar proposed in [10]. Essentially, the DMG translation of L boils down to:

$$
\phi^\dagger \quad := \quad \lambda p. \exists j (\phi^\bullet ij \wedge pj).
$$

Note that this translation has i free. In fact, the DMG translation uses Montague's intensional typed logic with $^\cup$ and $^\cap$ (IL), but every IL formula has a corresponding typed logic formula in a language with one extra basic type (in our case: $\diamond$), with possibly one extra variable i free in this basic type (see Gallin [9]).

In our framework, it would be more natural to abstract over the state variable. This yields the following translation into type $(\diamond, ((\diamond, t), t))$:

$$
\phi^\ddagger \quad := \quad \lambda i \lambda p. \exists j (\phi^\bullet ij \wedge pj).
$$

To see that this higher type does make sense, let us bother to spell out the semantics for L as a function in $D_\diamond \to \mathcal{P}\mathcal{P}D_\diamond$.

$$
\begin{aligned}
||r||(s) &:= \{A \subseteq D_\diamond \mid s[r \mapsto d] \in A \text{ for some } d \in D_e\} \\
||P^n t_1 \cdots t_n||(s) &:= \{A \subseteq D_\diamond \mid s \in A \text{ and } (I_s t_1, \cdots, I_s t_n) \in I(P)\} \\
||\phi_1 ; \phi_2||(s) &:= \bigcup \{||\phi_2||(s') \mid \{s'\} \in ||\phi_1||(s)\} \\
||\neg\phi||(s) &:= \{A \subseteq D_\diamond \mid s \in A \text{ and } ||\phi||(s) = \{\emptyset\}\}
\end{aligned}
$$

Because of the equation

$$
||\phi||(s) \quad = \quad \{A \subseteq D_\diamond \mid s' \in A \text{ for some } s' \text{ with } s[\![\phi]\!]s'\}
$$

we have that sets in $||\phi||(s)$ are always up-sets, in the sense that they are all of the form $\uparrow \mathbf{B}$, where $\uparrow \mathbf{B} := \{B' \supseteq B \mid B \in \mathbf{B}\}$. Indeed, we can write the function $||\cdot||$ as follows:

$$
||\phi||(s) \quad = \quad \uparrow \{\{s'\} \mid s[\![\phi]\!]s'\}.
$$

In [10] an alternative negation $\sim$ is proposed, to be used in addition to $\neg$, with semantics

$$||\sim\phi||(s) \quad := \quad \mathcal{P}\mathcal{P}D_s - ||\phi||(s).$$

For an extensive discussion of the pros and cons of this dynamic negation see the second chapter of Dekker [4]. In brief, the meaning of an expression containing $\sim$ negations will not be an up-set, for if $||\phi||(s)$ is an up-set, $||\sim\phi||(s)$ is a down-set, and vice versa. This is slightly problematic, for in general one wants that the meaning of an expression acts as an information update, and information updates correspond to up-sets. On the positive side, we have that $||\phi||(s)$ and $||\sim\sim\phi||(s)$ are the same, so we recover the law of double negation at the dynamic level.

Note that $\sim\phi$ is also readily translated into TLS, by means of the following extension of ‡:

$$(\sim\phi)^{\ddagger} \quad := \quad \lambda i\lambda p.(\neg((\phi^{\ddagger}p)i)).$$

The further extensions proposed in Dekker [4] also have obvious translations in TLS.

9 Comparisons

In this section we will compare TLS with some of the typed logics that have been proposed for integrating discourse representation theory [16, 7] into a Montague-style compositional framework [3, 10, 20, 21, 18]. The proposals that come closest to the present approach are [3] and [20].

The two main flavours of dynamic typed logic are a dynamic version of Montague's [19] IL, and a dynamic version of Gallin's [9] Ty_2. Both of these take a typed logic in which possible worlds either figure as a domain for forming intensions (IL) or as a basic type (Ty_2), and impose extra meaning postulates intended to force these possible worlds to behave like stores for a given set of store constants.

One good reason for allowing quantifying over states, i.e., for following the Ty_2 tradition rather than the IL route, is that an extension with 'real' possible worlds is trivial. Just build your TLS models over a basic type set B which includes the type s, and interpret s as the domain D_s of possible worlds. Translations of dynamic modal predicate logic [5] and similar logics into TLS are now straightforward. Also, TLS comes with a notion of reduction which is both precise and elegant, while reduction of expressions in dynamic versions of IL is hopelessly cumbersome.

The general procedure for distilling a typed logic with states out of ordinary typed logic is as follows (we assume for simplicity that the values to be stored all have the same type T): in the footsteps of Janssen [15], one starts out with an arbitrary domain D_s, one introduces a set C of store constants of type (s, T) (where T does not depend on s), and one tries to force D_s to behave as the function space $C \to D_T$. The two principles needed for this procedure are:

1. states can only differ in the values of the store constants,
2. there are enough states, in the sense that an update of an arbitrary store constant with an arbitrary value will always yield a new state.

Unfortunately, it is impossible to formulate a postulate for (1) as a statement of the typed language. A postulate for (1) must refer to a constant REG which is assumed to be true of a term E iff E is a register of values of type T. Assume for simplicity that all registers store values of the same type T, and consider the following postulate.

$$\text{DIST} \qquad \forall i_s j_s (\forall x_{(s,T)} (\text{REG } x \to (xi = xj)) \to i = j).$$

Obviously, given a compositional semantics, there can be no interpretation function with $[\![\text{REG } x]\!] = 1$ iff x itself is a register for values of type T, simply because compositionality means that the objects in a model (in this case: the object $[\![x]\!]$) do not reveal the names that have been used to refer to them (in this case: the register name x).

Note, however, that what DIST attempts to express is true for all TLS models. The following principle (which cannot be expressed as a formula of the language for reasons explained above) holds for every general model $M = (\{D_U\}, A_\diamond, \{A_{UU'}\})$ and every assignment g for M:

$$\begin{aligned} [\![i = j]\!]_g^M = 1 \quad & \text{iff} \quad g(i) = g(j) \\ & \text{iff} \quad A_\diamond(g(i), r) = A_\diamond(g(j), r) \text{ for all } r \in R. \end{aligned}$$

This illustrates that distributivity is built into the TLS general models (and therefore also into the full models) from the start.

Since (1) cannot be enforced by a formula of standard typed logic, one way around it is to make a virtue out of a necessity by dropping the requirement that states are only made up out of the named registers. This is what Muskens proposes in [20]. Here a constant ST is introduced to be interpreted as the property of being a store function. Appropriate axioms are employed to ensure that store functions are the stuff that states are made of. Assume that ST is a constant for the property of being a store function. Then the formula

$$\forall x_{(s,T)} ((\text{ST}x \wedge x \neq c) \to (xi = xj))$$

can abbreviate the relation of differing at most in the value of store register c. Notation: $i[c]j$. Next, it is ensured by means of axioms that there are enough states and that *at least all store constants* are interpreted as store functions, as follows:

$$\begin{array}{ll} \text{AX 1} & \forall i \forall x \forall e (\text{ST}x \to \exists j (i[x]j \wedge xj = e)) \\ \text{AX 2} & \text{ST}c \text{ for each store constant } c \\ \text{AX 3} & c \neq c' \text{ for each pair of different store constants } c, c'. \end{array}$$

To see that these axioms are consistent, consider the following model (personal communication from Reinhard Muskens): states are the functions $s \in \mathbb{N} \to \mathbb{N}$ with the property that $\{n \in \mathbb{N} \mid s(n) \neq 0\}$ is finite. This set of states is denumerable, so we can arrange states and registers in a square as follows:

	u_1	u_2	u_3	u_4	u_5	$\cdots$
s_1						
s_2						
s_3						
s_4						
s_5						
$\vdots$						

Now the columns can be taken to be the interpretations of the register constants, so u_i is interpreted as the i-th column. It is not difficult to see that the store functions are precisely those $F \in S \to \mathbb{N}$ with the property that there is some $n \in \mathbb{N}$ such that for all $s \in S$, $F(s) = s(n)$. (n is the index of the column.) Therefore we put STf true iff f corresponds to a column.

One thing to check now is that the store functions are indeed independent. We want to ensure that, for example, $\lambda i.u_1 i + u_2 i$, does not correspond to a register. But this is easy. Take an arbitrary register (column) u_i, and an arbitrary state (row) s_j. Suppose we put a new value m at the position of $s_j(i)$ (the place where the column and the row intersect). Then the resulting row must be present somewhere in the table, for it is again a function s with the property that $\{n \in \mathbb{N} \mid s(n) \neq 0\}$ is finite. But this means that u_i must be independent of the other stores. Thus, $\lambda i.u_1 i + u_2 i$ does not correspond to a store.

Note that all the axioms are true in this model. This shows that the axioms are consistent. Note that the axioms do not ensure that the states are built of precisely the registers mentioned in the axioms. For an example of a model with anonymous registers, consider the following slight extension of the previous model: states and interpretation of ST as before, interpretation of registers as follows ($A_1, A_2, A_3, \ldots$ are anonymous registers):

	u_1	A_1	u_2	A_2	u_3	$\cdots$
s_1						
s_2						
s_3						
s_4						
s_5						
$\vdots$						

This is also a model of the axioms AX1–AX3. It seems that the choice between the approach of Muskens and the present approach depends on whether one wants DIST or not.

The postulate that should guarantee requirement (2) (enough states) can be formulated in TLS as follows:

$$\text{UPDATE} \qquad \forall i_\diamond \forall x_T \exists j_\diamond (j = \ (c|x)i)$$

We have already seen above that the UPDATE postulate is true in all TLS models (note that in Muskens' approach UPDATE is guaranteed by AX 1).

10 Further Work

The two main areas of further work are:

- exploring further use of TLS as a tool,
- further logical investigation of TLS.

Proposals for dynamic semantics are more readily compared when they are formulated within the same framework. In the area of NL semantics, suitable candidates for translation into TLS are the sequence semantics for dynamic predicate logic from

[27], the extension of dynamic predicate logic with procedures from [6], the modalized versions of dynamic predicate logic of [5, 12], and so on.

In the area of programming language semantics, static program analysis for *while* programs is easily performed within TLS. Note, for instance, that partial correctness assertions $\{P\}S\{Q\}$ are readily translated into TLS formulas by means of:

$$\forall ij((Pi \wedge S^{\bullet}ij) \rightarrow Qj),$$

while total correctness assertions take the form:

$$\forall i(Pi \rightarrow \exists j(S^{\bullet}ij \wedge Qj).$$

In a slightly different direction, TLS is a suitable tool for variable dependency analysis of *while* programs, including reasoning about safety of the analysis with respect to a given semantic specification (see [23] for examples of such reasoning).

There are further logical questions to be asked about TLS. Modifications of the σ, ρ and τ axioms may provide an interesting connection with Van Benthem's weak predicate logics [2], which are also the result of varying the restrictions on the set of available variable assignments. As an example, consider the ρ axiom permitting the swap between $(r_i|E)((r_j|F)G)$ and $(r_j|F)((r_i|E)G)$. This expresses independence of the registers; if we drop this axiom we allow models with states where register r_i may depend in some way on r_j or vice versa. In a similar way, one can consider dropping the other ρ axiom, $(r_i|E)((r_i|F)G) = (r_i|E)G$, which expresses the destructiveness of register update. If one considers $(r_i|E)((r_i|F)G)$ and $(r_i|E)G$ as different states, this means that registers effectively become stacks. Together with an appropriate mechanism for register lookup (lookup at arbitrary depths in the register stacks, or an operator for throwing the top of the stack away) this would give us the typing superstructure for a dynamic logic with stack updates in the spirit of the already mentioned [27].

Acknowledgements

Thanks to Erik Barendsen, Johan van Benthem, Paul Dekker, Theo Janssen, Jan-Willem Klop, Reinhard Muskens, Femke van Raamsdonk and Anne Troelstra for helpful discussions, and to four anonymous referees of this Journal for spurring me on to considerable improvements with their feedback on previous versions of this paper. All remaining errors are my own.

References

[1] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics (2nd ed.).* North-Holland, Amsterdam, 1984.

[2] J. van Benthem. *Exploring Logical Dynamics.* CSLI & Folli, 1996.

[3] G. Chierchia. Anaphora and dynamic binding. *Linguistics and Philosophy*, 15(2):111–183, 1992.

[4] P. Dekker. *Transsentential Meditations.* PhD thesis, Department of Philosophy, University of Amsterdam, 1993.

[5] J. van Eijck and G. Cepparello. Dynamic modal predicate logic. In M. Kanazawa and C.J. Pinón, editors, *Dynamics, Polarity and Quantification*, pages 251–276. CSLI, Stanford, 1994.

[6] J. van Eijck and N. Francez. Verb-phrase ellipsis in dynamic semantics. In M. Masuch and L. Polos, editors, *Applied Logic: How, What and Why?*, pages 29–60. Kluwer, Dordrecht, 1995.

[7] J. van Eijck and H. Kamp. Representing discourse in context. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic in Linguistics*, pages 179–237. Elsevier, Amsterdam, 1996.

[8] H. Friedman. Equality between functionals. In R. Parikh, editor, *Logic Colloquium, Boston 1972–1973*, pages 22–37. 1975.

[9] D. Gallin. *Intensional and Higher-Order Modal Logic; with Applications to Montague Semantics.* North Holland, Amsterdam, 1975.

[10] J. Groenendijk and M. Stokhof. Dynamic Montague Grammar. In L. Kalman and L. Polos, editors, *Papers from the Second Symposium on Logic and Language*, pages 3–48. Akademiai Kiadoo, Budapest, 1990.

[11] J. Groenendijk and M. Stokhof. Dynamic predicate logic. *Linguistics and Philosophy*, 14:39–100, 1991.

[12] J. Groenendijk, M. Stokhof, and F. Veltman. Coreference and modality. In S. Lappin, editor, *The Handbook of Contemporary Semantic Theory*, pages 179–213. Blackwell, Oxford, 1996.

[13] L. Henkin. Completeness in the theory of types. *Journal of Symbolic Logic*, 15:81–91, 1950.

[14] J.R. Hindley and J.P. Seldin. *Introduction to Combinators and λ-Calculus.* Cambridge University Press, Cambridge, 1986.

[15] T.M.V. Janssen. *Foundations and Applications of Montague Grammar.* CWI Tract 19. CWI, Amsterdam, 1986.

[16] H. Kamp. A theory of truth and semantic representation. In J. Groenendijk et al., editors, *Formal Methods in the Study of Language.* Mathematisch Centrum, Amsterdam, 1981.

[17] J.W. Klop. Term rewriting systems. In S. Abramski, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, pages 1–116. Oxford University Press, 1992.

[18] M. Kohlhase, S. Kuschert, and M. Pinkal. A type-theoretic semantics for λ-DRT. In P. Dekker and M. Stokhof, editors, *Proceedings of the Tenth Amsterdam Colloquium*, Amsterdam, 1996. ILLC.

[19] R. Montague. The proper treatment of quantification in ordinary English. In J. Hintikka e.a., editor, *Approaches to Natural Language*, pages 221–242. Reidel, 1973.

[20] R. Muskens. Tense and the logic of change. In U. Egli et al., editor, *Lexical Knowledge in the Organization of Language*, pages 147–183. W. Benjamins, 1995.

[21] R. Muskens. Combining Montague Semantics and Discourse Representation. *Linguistics and Philosophy*, 19:143–186, 1996.

[22] R. Muskens. Program semantics and classical logic. Unpublished Manuscript, 1997.

[23] H.R. Nielson and F. Nielson. *Semantics with Applications.* John Wiley and Sons, 1992.

[24] G. Plotkin. *Structural Operational Semantics.* Aarhus University, Denmark, 1981. Lecture notes.

[25] D.S. Scott and C. Strachey. Towards a mathematical semantics for computer languages. In J. Fox, editor, *Computers and Automata*, pages 19–46. Polytechnic Institute of Brooklyn Press, 1971.

[26] R. Statman. The typed lambda-calculus is not elementary recursive. *Theoretical Computer Science*, 9:73–81, 1979.

[27] C.F.M. Vermeulen. Sequence semantics for dynamic predicate logic. *Journal of Logic, Language, and Information*, 2:217–254, 1993.

Received 23 October, 1996