Dynamic Logic

Prof. P.H. Schmitt

Institut für Theoretische Informatik
Fakultät für Informatik
Universität Karlsruhe (TH)

Logic Summer School, Canberra, February, 2009

Overview

Lecture 1	Dynamic Logic for Regular Programs	Monday
Lecture 2	Propositional Dynamic Logic	Tuesday
Lecture 3	Completeness	Wednesday
Lecture 4	Typed First-Order Logic	Wednesday
Lecture 5	Updates	Thursday
Lecture 6	Dynamic Logic for Javacard	Friday

Dynamic Logic for Regular Programs

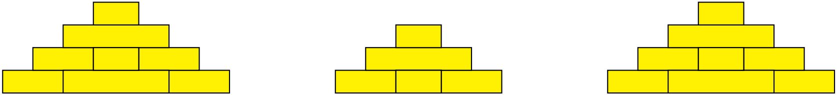
Prof. P.H. Schmitt

Institut für Theoretische Informatik
Fakultät für Informatik
Universität Karlsruhe (TH)

Logic Summer School, Canberra, February, 2009

Motivating Example

Introductory Example



The Towers of Hanoi

The Instructions

1. Move alternatingly the smallest disc and another one.
2. If moving the smallest disc put it on the stack it did not come from in the previous move.
3. If not moving the smallest disc do the only legal move,

More Formaly:

$$moveS; moveO; moveS; moveO; \ldots$$

more concise:

$$(moveS; moveO)^*$$

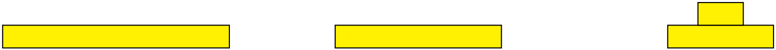
improved:

$$moveS; testForStop; (moveO; moveS; testForStop)^*$$

States of the Environment

$$stack(n,m) = \begin{cases} k > 0 & \text{on stack } n \text{ at position } m \\ & \text{there is a disk of size } k \\ 0 & \text{on stack } n \text{ at position } m \\ & \text{there is no disk} \end{cases}$$

with $1 \leq n \leq 3$ and $1 \leq m \leq d$ with d the number of disks.



stack	*first*	*second*	*third*
position 4	0	0	0
position 3	0	0	0
position 2	0	0	1
position 1	4	3	2

Properties of the Environment

testForStop

$$\forall m(1 \leq m \leq d \rightarrow stack(3, m) \neq 0)$$

that is to say: stack 3 is full

Invariant: OrderedStacks

$$\bigwedge_{1 \leq n \leq 3} \forall m_1, m_2 \quad ((1 \leq m_1 < m_2 \leq d \wedge stack(n, m_1) \neq 0) \\ \rightarrow stack(n, m_1) > stack(n, m_2))$$

that is to say: size of disks decreases on each stack

Invariants

A formula ϕ is an invariant for an action A if:

whenever ϕ is true before A
it is also true after the execution of action A.

more formal

$$\text{OrderedStacks} \rightarrow \langle \mathit{moveS} \rangle \ \text{OrderedStacks}$$

Dynamic Logic

Dynamic Logic

- Allows to reason about properties of composite actions.
- Actions are explicitly part of the language.
- Extends modal logic and first-order logic.

Syntax
Vocabulary

For every vocabulary Σ we will define the following categories of syntactic objects

terms, $Term_\Sigma$
formulas, Fml_Σ
programs, Π

As usual a vocabulary Σ consists of

- a set of function symbols $f, g, f_i, \ldots$ with fixed number of arguments,
- 0-place functions symbols will also be called constant symbols,
- a set of predicate symbols $p, q, p_i, \ldots$ with fixed number of arguments.

By Var we denote an infinite set of variable symbols.

Syntax
Terms

1. $x \in Term_\Sigma$ for $x \in \mathrm{Var}$
   Every variable symbols is a term.
2. $f(t_1, \ldots, t_n \in Term_\Sigma$
   for every n-place functions symbol $f \in \Sigma$ and $t_1, \ldots, t_n \in Term_\Sigma$

Syntax

Formulas and Programs

1. atomic formulas
   $r(t_1, \ldots, t_n) \in Fml_\Sigma$ for every n-place relation symbol $r \in \Sigma$ and terms $t_i \in Term_\Sigma$.
2. equations
   $t_1 = t_2 \in Fml_\Sigma$ for $t_1, t_2 \in Term_\Sigma$
3. closure under predicate logic operators
   If $F_1, F_2 \in Fml_\Sigma$ then also
   $F_1 \vee F_2$ $F_1 \wedge F_2$, $F_1 \rightarrow F_2$, $\neg F_1$, $\forall x F_1$ and $\exists x F_1$.
4. modal operators
   $[\pi]F, \langle\pi\rangle F \in Fml_\Sigma$ for $F \in Fml_\Sigma$ and $\pi \in \Pi$.

Syntax

Formulas and Programs (continued)

5. atomic programs
   $(x := t) \in \Pi$ for $t \in Term_\Sigma$ and $x \in \mathrm{Var}$.
6. composite programs
   If $\pi_1, \pi_2 \in \Pi$ then
   - 6.1 $\pi_1 ; \pi_2 \in \Pi$ — sequential composition
   - 6.2 $\pi_1 \cup \pi_2 \in \Pi$ — nondeterministic choice
   - 6.3 $\pi^* \in \Pi$ — iteration
7. tests
   $con? \in \Pi$ for every quantifierfree formula $con \in Fml_\Sigma$.

Π as defined above is called the set of regular programs.

Semantics

Kripke Structures

For every first-order structure $\mathcal{M} = (M, val_{\mathcal{M}})$ we will define a Kripke structure

$$\mathcal{K}_{\mathcal{M}} = (S, \rho, \models)$$

with

S	the set of states
$\rho : \Pi \rightarrow S \times S$	the accessibility relations
$\models \;\; \subseteq S \times Fml_{\Sigma}$	the evaluation relation

$\mathcal{M}$ is called the domain of computation of $\mathcal{K}$.

Semantics

The Set of States

The set of states for Kripke structure $\mathcal{K}$ is the set of all assignments u of elements in the universe M to variables in Var:

$$S = \mathrm{Var} \rightarrow M$$

For every $t \in Term_{\Sigma}$ we denote by

$$val_{\mathcal{M},u}(t)$$

the usual first-order evaluation of t in $\mathcal{M}$ with variables in t are interpreted via u.

Notation: for $s \in S$, $x \in \mathrm{Var}$, $a \in M$

$$s[x/a](y) = \begin{cases} a & \text{if } y = x \\ s(y) & \text{otherwise} \end{cases}$$

Semantics

Formulas and Programs

$$
\begin{array}{lll}
s \models r(t_1, \ldots, t_n) & \text{iff} & (val_{\mathcal{M},u}(t_1), \ldots, val_{\mathcal{M},u}(t_n)) \in val_{\mathcal{M}}(r) \\
s \models t_1 = t_2 & \text{iff} & val_{\mathcal{M},u}(t_1) = val_{\mathcal{M},u}(t_2) \\
s \models F & & F \text{ matching one of } F_1 \vee F_2, F_1 \wedge F_2, \\
 & & F_1 \rightarrow F_2, \neg F_1, \forall x F_1 \text{ or } \exists x F_1 \\
 & & \text{as usual.} \\
s \models [\pi]F & \text{iff} & \text{for all } s' \text{ with } (s, s') \in \rho(\pi) \\
 & & s' \models F \\
s \models \langle\pi\rangle F & \text{iff} & \text{there exists } s' \text{ with } (s, s') \in \rho(\pi) \\
 & & \text{and } s' \models F
\end{array}
$$

Semantics

Formulas and Programs (continued)

$(u, u') \in \rho(x := t)$	iff	$u' = u[x/val_{\mathcal{M},u}(t)]$
$(u, u') \in \rho(\pi_1; \pi_2)$	iff	there exists $w \in S$ with $(u, w) \in \rho(\pi_1)$ and $(w, u') \in \rho(\pi_2)$
$(u, u') \in \rho(\pi_1 \cup \pi_2)$	iff	$(u, u') \in \rho(\pi_1)$ or $(u, u') \in \rho(\pi_2)$
$(u, u') \in \rho(\pi^*)$	iff	there exists n and $u_1, \dots u_n \in S$ such that $u_1 = u$ and $u_n = u'$ and $(u_i, u_{i+1}) \in \rho(\pi)$ for $1 \leq i < n$
$(u, u') \in \rho(con?)$	iff	$u = u'$ and $u \models con$

Example

$$
\begin{array}{lll}
(u, u') \in \rho(con?; \pi) & \text{iff} & \text{exists } w \text{ with} \\
& & (u, w) \in \rho(con?) \text{ and } (w, u') \in \rho(\pi) \\
& \text{iff} & \text{exists } w \text{ with} \\
& & u \models con,\ w = u \text{ and } (w, u') \in \rho(\pi) \\
& \text{iff} & u \models con \text{ and } (u, u') \in \rho(\pi)
\end{array}
$$

Defined Operations 1

$$
\begin{array}{ll}
\multicolumn{2}{l}{(u,u') \in \rho((con?;\pi_1) \cup (\neg con?;\pi_2))} \\
\text{iff} & (u,u') \in \rho((con?;\pi_1)) \text{ or} \\
 & (u,u') \in \rho((\neg con?;\pi_2)) \\
\text{iff} & u \models con \text{ and } (u,u') \in \rho(\pi_1) \text{ or} \\
 & u \models \neg con \text{ and } (u,u') \in \rho(\pi_2) \\
\text{iff} & (u,u') \in \rho(\mathbf{if}\ con\ \mathbf{then}\ \pi_1\ \mathbf{else}\ \pi_2)
\end{array}
$$

Thus:

$$(con?;\pi_1) \cup (\neg con?;\pi_2) \equiv (\mathbf{if}\ con\ \mathbf{then}\ \pi_1\ \mathbf{else}\ \pi_2)$$

Defined Operations 2

$(u, w) \in \rho((A?; \pi)^*; \neg A?)$

iff there exist $n \in \mathbb{N}$ and $u_1, \ldots, u_n \in S$ with $u_1 = u$
$(u_i, u_{i+1}) \in \rho(A?; \pi)$ for all i, $1 \leq i < n$ and
$(u_n, w) \in \rho(\neg A?)$

iff there exist $n \in \mathbb{N}$ and $u_1, \ldots, u_n \in S$ with $u_1 = u$, $u_n = w$
$(u_i, u_{i+1}) \in \rho(\pi)$ and $u_i \models A$ for all i, $1 \leq i < n$ and
$w \models \neg A$

iff $(u, w) \in \rho(\textbf{while } A \textbf{ do } \pi)$

Thus

$$\textbf{while } A \textbf{ do } \pi \quad \equiv \quad (A?; \pi)^*; \neg A?$$

Defined Operations 3

Exercise

Show that

$$\textbf{repeat}\ \alpha\ \textbf{until}\ A \quad \equiv \quad \alpha; (\neg A?; \alpha)^*$$

Examples
of DL formulas

- $pre \rightarrow [\pi]\ post$ — partial correctness
  equivalent to Hoare triple $\{pre\}\ \pi\ \{post\}$.
- $pre \rightarrow \langle\pi\rangle\ post$ — total correctness
  equivalent to Hoare triple $pre\ \{\pi\}\ post$.
- $\langle\pi\rangle\mathbf{true}$ — program π terminates.
- $\langle\pi_1\rangle F \rightarrow \langle\pi_2\rangle F$ — property of program transformation
- $[\mathbf{while\ true\ do}\ y := y + 1]\ \mathbf{false}$ — is always true
- $s \models \langle(r(x,z)?; x := z)^*\rangle\ x = y$ — transitive closure
  the pair $(s(x), s(y))$ is in the transitive closure of the relation $val_{\mathcal{M}}(r)$ in the computational domain.

Validity

Uninterpreted Case

Σ	a vocabulary
$\mathcal{M}$	a Σ structure
$\mathcal{K}_{\mathcal{M}}$	the Kripke structure with computation domain $\mathcal{M}$
$s \in S$	a state in the state space of $\mathcal{K}_{\mathcal{M}}$
F, G	formulas in Fml_{Σ} possibly with free variables

$s \models F$	F is true in state s	$(\mathcal{K}_{\mathcal{M}}, s) \models F$ if necessary
$\mathcal{K}_{\mathcal{M}} \models F$	F is true in $\mathcal{K}_{\mathcal{M}}$	$s \models F$ for all $s \in S$.
$\vdash F$	F is valid	$\mathcal{K}_{\mathcal{M}} \models F$ for all $\mathcal{M}$.
$G \vdash F$	G (locally) entails F	for all $\mathcal{M}$ and all $s \in S$ if $s \models G$ then also $s \models F$
$G \vdash^{g} F$	G globally entails F	for all $\mathcal{M}$ if $s \models G$ for all $s \in S$ then $s \models F$ for all $s \in S$

Examples

of valid formulas

We assume variable x does not occur in program π.

1. $(\exists x \langle\pi\rangle F) \leftrightarrow (\langle\pi\rangle\exists x\ F)$
2. $(\forall x\ [\pi]F) \leftrightarrow ([\pi]\forall x\ F)$
3. $(\exists x\ [\pi]F) \rightarrow ([\pi]\exists x\ F)$
4. $([\pi]\exists x\ F) \rightarrow (\exists x\ [\pi]F)$ if π is deterministic
5. $(\langle\pi\rangle\forall x\ F) \rightarrow (\forall x\ \langle\pi\rangle F)$
6. $(\forall x\ \langle\pi\rangle F) \rightarrow (\langle\pi\rangle\forall x\ F)$ if π is deterministic
7. $(\langle\pi\rangle(F \wedge G)) \rightarrow (((\langle\pi\rangle F) \wedge \langle\pi\rangle G)$
8. $(\langle\pi\rangle(F \wedge G)) \leftrightarrow (((\langle\pi\rangle F) \wedge \langle\pi\rangle G)$ if π is deterministic

Another Valid Formula

$$x = y \land \forall x(f(g(x)) = x) \rightarrow$$
$$[\textbf{while } p(y) \textbf{ do } y := g(y)]\langle\textbf{while } y \neq x \textbf{ do } y := f(y)\rangle\textbf{true}$$

Validity

Interpreted Case

Let $\mathcal{M}$ be a fixed Σ structure.

$\vdash_{\mathcal{M}} F$	F is $\mathcal{M}$-valid	$\mathcal{K}_{\mathcal{M}} \models F$ for all $\mathcal{M}$.
$G \vdash_{\mathcal{M}} F$	G (locally) $\mathcal{M}$-entails F	for all $s \in S$ if $s \models G$ then also $s \models F$
$G \vdash^{g}_{\mathcal{M}} F$	G globally $\mathcal{M}$-entails F	if $s \models G$ for all $s \in S$ then $s \models F$ for all $s \in S$

Examples

with computational domain $\mathcal{M} = (\mathbb{N}, 0, +, -, >)$

- $(p(0) \wedge \forall x(p(x) \rightarrow p(x+1))) \rightarrow \forall x p(x)$
- $\neg \exists x(0 < x \wedge x < 1)$
- $$\begin{array}{rcl} TC^0_R(x,y,z) & \leftrightarrow & (z = 0 \wedge x = y) \vee \\ & & (z > 0 \wedge \exists u(TC^0_R(x,u,z-1) \wedge R(u,y)) \end{array}$$
- $TC_R(x,y) \leftrightarrow \exists z(TC^0_R(x,y,z))$
  $TC_R(x,y)$ defines the reflexive, transitive closure of R.

Dynamic Logic
Lecture 2: Propositional Dynamic Logic

Prof. P.H. Schmitt

Institut für Theoretische Informatik
Fakultät für Informatik
Universität Karlsruhe (TH)

Logic Summer School, Canberra, February, 2009

Syntax of PDL

Formulas and Programs

1. atomic formulas
   $p \in PFml$ for any propositional variable $p \in PVar$.
2. equations do no longer exist
3. closure under propositional logic operators
   If $F_1, F_2 \in PFml$ then also $F_1 \vee F_2$ $F_1 \wedge F_2$, $F_1 \rightarrow F_2$, $\neg F_1$
4. modal operators
   $[\pi]F, \langle\pi\rangle F \in PFml$ for $F \in PFml$ and $\pi \in \Pi$.

Syntax of PDL

Formulas and Programs (continued)

5. atomic programs

   $a \in \Pi$ for every atomic program $a \in AP$

6. composite programs

   If π_1,$\pi_2 \in \Pi$ then

   6.1 $\pi_1;\pi_2 \in \Pi$ sequential composition
   6.2 $\pi_1 \cup \pi_2 \in \Pi$ nondeterministic choice
   6.3 $\pi^* \in \Pi$ iteration

7. tests

   $con? \in \Pi$ for every formula $con \in PFml$.

   rich tests

Π as defined above is called the set of regular programs.

Semantics of PDL

Propositional Kripke Structures

A propositional Kripke structure

$$\mathcal{K} = (S, \models, \rho)$$

is determined by:

S	the set of states
$\models \ \subseteq (S \times PVar)$	evaluation of propositional atoms in states
$\rho : AP \to S \times S$	the accessibility relations for atomic programs

The semantics definition will extend

- $\models$ to a relation $\models \ \subseteq (S \times PFml)$ and
- ρ to a function $\Pi \to S \times S$.

We will use the infix notation $s \models F$ instead of $(s, F) \in \ \models$.

Semantics of PDL

Formulas and Programs

$s \models p,\ p \in PVar$	iff	$s(p) = \mathbf{true}$
$s \models F$	iff	F matching one of $F_1 \vee F_2, F_1 \wedge F_2,$ $F_1 \rightarrow F_2, \neg F_1$ as usual.
$s \models [\pi]F$	iff	for all s' with $(s, s') \in \rho(\pi)$ $s' \models F$
$s \models \langle\pi\rangle F$	iff	there exists s' with $(s, s') \in \rho(\pi)$ and $s' \models F$

Semantics of PDL

Formulas and Programs (continued)

$(u, u') \in \rho(a),\ a \in AP$	iff	$(u, u') \in \rho(a)$
$(u, u') \in \rho(\pi_1; \pi_2)$	iff	there exists $w \in S$ with $(u, w) \in \rho(\pi_1)$ and $(w, u') \in \rho(\pi_2)$
$(u, u') \in \rho(\pi_1 \cup \pi_2)$	iff	$(u, u') \in \rho(\pi_1)$ or $(u, u') \in \rho(\pi_2)$
$(u, u') \in \rho(\pi^*)$	iff	there exists n and $u_1, \ldots u_n \in S$ such that $u_1 = u$ and $u_n = u'$ and $(u_i, u_{i+1}) \in \rho(\pi)$ for $1 \leq i < n$
$(u, u') \in \rho(con?)$	iff	$u = u'$ and $u \models\ con$

Example

of propositional tautologies

1. $[\pi_1; \pi_2]F \leftrightarrow [\pi_1][\pi_2]F$
2. $[\pi_1 \cup \pi_2]F \leftrightarrow ([\pi_1]F \land [\pi_2]F)$
3. $[(\pi)^*]F \leftrightarrow (F \land [\pi][(\pi)^*]F)$
4. $\langle\pi\rangle F \leftrightarrow \neg[\pi]\neg F$
5. $\langle\pi_1; \pi_2\rangle F \leftrightarrow \langle\pi_1\rangle\langle\pi_2\rangle F$
6. $\langle\pi_1 \cup \pi_2\rangle F \leftrightarrow (\langle\pi_1\rangle F \lor \langle\pi_2\rangle F)$
7. $\langle(\pi)^*\rangle F \leftrightarrow (F \lor \langle\pi\rangle\langle(\pi)^*\rangle F)$
8. $[\pi](F \rightarrow G) \rightarrow ([\pi]F \rightarrow [\pi]G)$
9. $[(\pi)^*](F \rightarrow [\pi]F) \rightarrow (F \rightarrow [(\pi)^*]F)$

A Calculus for Propositional Dynamic Logic

Axioms

All propositional tautologies			(A1)
$\langle\pi\rangle(F \vee G)$	$\leftrightarrow$	$\langle\pi\rangle F \vee \langle\pi\rangle G$	(A2)
$\langle\pi_1;\pi_2\rangle F$	$\leftrightarrow$	$\langle\pi_1\rangle\langle\pi_2\rangle F$	(A3)
$\langle\pi_1 \cup \pi_2\rangle F$	$\leftrightarrow$	$\langle\pi_1\rangle F \vee \langle\pi_2\rangle F$	(A4)
$\langle\pi^*\rangle F$	$\leftrightarrow$	$F \vee \langle\pi\rangle\langle\pi^*\rangle F$	(A5)
$\langle A?\rangle F$	$\leftrightarrow$	$A \wedge F$	(A6)
$[\pi^*](F \to [\pi]F)$	$\to$	$(F \to [\pi^*]F)$	(A7)
$[\pi](F \to G)$	$\to$	$([\pi]F \to [\pi]G)$	(A8)

Rules

$$\frac{F,\ F \to G}{G} \qquad \text{(MP)}$$

$$\frac{F}{[\pi]F} \qquad \text{(G)}$$

Theorem

The presented calculus is sound and complete.

Proof
See e.g.,pp. 559-560
in David Harel's article *Dynamic Logic*
in the *Handbook of Philosophical Logic*, *Volume II*,
published by D.Reidel in 1984.
or
D. Harel, D. Kozen and J. Tiuryn
Dynamic Logic
in *Handbook of Philosophical Logic,* 2^{nd} *edition* , *volume 4*
by Kluwer Academic Publisher, 2001.

Is

Propositional Dynamic Logic

decidable?

Fischer-Ladner Closure

Let S_0 be a set of formulas in $PFml$.
The Fischer-Ladner closure of S_0 is the smallest subset $S \subseteq PFml$ satisfying:

1	$S_0 \subseteq S$		
2	$\neg G \in S$	$\Rightarrow$	$G \in S$
3	$(G_1 \vee G_2) \in S$	$\Rightarrow$	$G_1 \in S$ and $G_2 \in S$
4	$\langle \pi \rangle G \in S$	$\Rightarrow$	$G \in S$
5	$\langle \pi_1 ; \pi_2 \rangle G \in S$	$\Rightarrow$	$\langle \pi_1 \rangle \langle \pi_2 \rangle G \in S$
6	$\langle \pi_1 \cup \pi_2 \rangle G \in S$	$\Rightarrow$	$\langle \pi_1 \rangle G \in S$ and $\langle \pi_2 \rangle G \in S$
7	$\langle \pi_1^* \rangle G \in S$	$\Rightarrow$	$\langle \pi_1 \rangle \langle \pi_1^* \rangle \in S$
8	$\langle G_1 ? \rangle G_2 \in S$	$\Rightarrow$	$G_1 \in S$ and $G_2 \in S$

For $F \in PFml$ we denote by $FL(F)$ the Fischer-Ladner closure of $\{F\}$.
We assume that F does not contain $[\ ]$, $\wedge$, $\rightarrow$.

Fischer-Ladner Closure

A Tableau Procedure

$F \in PFml$

$cl^{\Diamond}(F)$ is smallest set C with $F \in C$ and if $\langle\pi\rangle G \in C$ then $G \in C$.

Notation: $cl^{\Diamond}(F) = \{F_1, \ldots F_k\}$, $cl^{\Diamond}(G) = \{G_1, \ldots G_m\}$.

$$\frac{\neg F}{F_1 \ \ldots F_k} \qquad \frac{F \vee G}{F_1 \ldots F_k, G_1 \ldots G_m}$$

$$\frac{\langle\pi_1 \cup \pi_2\rangle F}{\langle\pi_1\rangle F \quad \langle\pi_2\rangle F} \qquad \frac{\langle\pi_1 ; \pi_2\rangle F}{\langle\pi_1\rangle\langle\pi_2\rangle F \quad \langle\pi_2\rangle F}$$

$$\frac{\langle\pi^*\rangle F}{\langle\pi_1\rangle\langle\pi^*\rangle F} \qquad \frac{\langle F?\rangle G}{F_1 \ \ldots F_k}$$

Fischer-Ladner Closure

First Step in Tableau Procedure

When constructing the tableau for a formula F with

$$cl^{\diamondsuit}(F) = \{F_1, \dots F_k\}$$

the first step is

$$start$$

$$F_1 \qquad F_i \qquad F_k$$

After every rule appliction during tableau construction it is true:

if there is a node labeled $\langle\pi\rangle G$, then there is also a node labeled G.

Fischer-Ladner Closure

Example

Computation of $FL(p \rightarrow \langle (q?;a)^*;\neg q?\rangle r)$.

$$p \rightarrow \langle (q?;a)^*;\neg q?\rangle r$$

$$\neg p \qquad r \qquad \langle (q?;a)^*;\neg q?\rangle r$$

$$p \qquad \langle (q?;a)^*\rangle\langle \neg q?\rangle r \qquad \langle \neg q?\rangle r$$

$$\langle q?;a\rangle\langle (q?;a)^*\rangle\langle \neg q?\rangle r \qquad \neg q$$

$$\langle q?\rangle\langle a\rangle\langle (q?;a)^*\rangle\langle \neg q?\rangle r \qquad \langle a\rangle\langle (q?;a)^*\rangle\langle \neg q?\rangle r \qquad q$$

$$q$$

Leaves of the tableau tree are shown in red.

Fischer-Ladner Closure

Properties of the Tableau Procedure

1. The procedure terminates
2. The set of all formulas generated by the procedure starting with the formula(s) $cl^{\Diamond}(F)$ is the Fischer-Ladner closure of F.
3. In particular, we now know that a finite Fischer-Ladner closure exists for every F.

Comment

It can be shown that the cardinality of $FL(F)$ is not greater than the size of F (i.e., the number of symbols in F).
But, this is not strictly needed for the decidability result.

Filtration

Equivalent States

Let $\mathcal{K} = (S, \models, \rho)$ be a propositional Kripke structure, $\Gamma \subseteq PFml$. The relation $\sim_\Gamma$ on S is defined by:

$$s_1 \sim_\Gamma s_2 \text{ iff } s_1 \models F \Leftrightarrow s_2 \models F \text{ for all } F \in \Gamma$$

It is not hard to see that $\sim_\Gamma$ is an equivalence relation.

Filtration

Quotient Structure

The quotient structure $\mathcal{K}_\Gamma = (S_\Gamma, \models_\Gamma, \rho_\Gamma)$ for $\mathcal{K} = (S, \models, \rho)$ with respect to the equivalence relation $\sim_\Gamma$ is defined by:

$[s]$	$=$	$\{s' \mid s \sim_\Gamma s'\}$	equiv. class of s
S_Γ	$=$	$\{[s] \mid s \in S\}$	
$[s] \models_\Gamma p$	$\Leftrightarrow$	$s \models p$	for $p \in \Gamma$
$[s] \models_\Gamma p$		arbitrary	otherwise
$([s_1], [s_2]) \in \rho_\Gamma(a)$	iff	for all $\langle\pi\rangle F \in \Gamma$	$a \in AP$
		if $s_1 \models \neg\langle\pi\rangle F$ then $s_2 \models \neg F$	

To guarantee that this definition is independent of the choice of representatives for equivalence classes we assume that $\langle\pi\rangle F \in \Gamma$ implies $F \in \Gamma$. The given definition of ρ_Γ is equivalent to

$([s_1], [s_2]) \in \rho_\Gamma(a)$	iff	for all $[\pi]F \in \Gamma$
		if $s_1 \models [\pi]F$ then $s_2 \models F$

Filtration

Properties

Let

F be PFml formula,

$\Gamma = FL(F)$ the Fischer-Ladner closure of F

$\mathcal{K} = (S, \models, \rho)$ a propositional Kripke structure

$\mathcal{K}_\Gamma = (S_\Gamma, \models_\Gamma, \rho_\Gamma)$ its quotient modulo $\sim_\Gamma$,

then the following is true for all $G \in \Gamma$, $\pi \in \Pi$ and $s_1, s_2 \in S$

1. Since Γ is finite, the relation $\sim_\Gamma$ has only finitely many equivalence classes, i.e., S_Γ is finite.
2. $([s_1], [s_2]) \in \rho_\Gamma(\pi)$ implies for all $\langle\pi\rangle B \in \Gamma$
   $s_1 \models \neg\langle\pi\rangle B \Rightarrow s_2 \models \neg B$
3. $(s_1, s_2) \in \rho(\pi)$ entails $([s_1], [s_2]) \in \rho_\Gamma(\pi)$
4. $s \models G$ iff $[s] \models G$

A Taste of the Proof

Item 4 $s \models G$ iff $[s] \models G$

Proof by induction on the complexity of G.
We consider the step from B to $G = \langle\pi\rangle B$.

Implication from left to right

If $s_1 \models \langle\pi\rangle B$, the there is s_2 with $(s_1, s_2) \in \rho(\pi)$ and $s_2 \models B$.
By induction hypothesis also $[s_2] \models B$
and by part 3 also $([s_1], [s_2]) \in \rho_\Gamma(\pi)$
thus $[s_1] \models < \pi > B$.

Implication from right to left

From $[s_1] \models \langle\pi\rangle B$ we get $[s_2]$, $([s_1], [s_2]) \in \rho_\Gamma(\pi)$ and $[s_2] \models B$
By induction hypothesis also $s_2 \models B$.
Assume $s_1 \models \neg\langle\pi\rangle B$. Part 2 yields $s_2 \models \neg B$
A contradiction.
Thus $s_1 \models \langle\pi\rangle B$.

A Taste of the Proof

$([s_1],[s_2]) \in \rho_\Gamma(\pi) \wedge s_1 \models \neg\langle\pi\rangle B \Rightarrow s_2 \models \neg B$ for all $\langle\pi\rangle B \in \Gamma$

Proof by induction on the complexity of π.
We consider the step from π to π^*.
$([s_1],[s_2]) \in \rho_\Gamma(\pi^*)$ yields by definition states $u_0, \ldots, u_k$ such that $[s_1] = [u_0]$, $[s_2] = [u_k]$ and for all $0 \leq i < k$ $([u_i],[u_{i+1}]) \in \rho_\Gamma(\pi)$.
By induction hypothesis
$u_i \models \neg\langle\pi\rangle C \Rightarrow u_{i+1} \models \neg C$ for all $\langle\pi\rangle C \in \Gamma$, all $0 \leq i < k$
We need to show
$s_1 \models \neg\langle\pi^*\rangle B \Rightarrow s_2 \models \neg B$ for all $\langle\pi^*\rangle B \in S$.
Observe that $\neg\langle\pi^*\rangle B \leftrightarrow \neg B \wedge \neg\langle\pi\rangle\langle\pi^*\rangle B$ is a tautology.
From $s_1 \models \neg\langle\pi^*\rangle B$ we thus get $s_1 \models \neg\langle\pi\rangle\langle\pi^*\rangle B$.
From $\langle\pi\rangle\langle\pi^*\rangle B \in \Gamma = FL(F)$ and $s_1 \sim_\Gamma u_0$ we know $u_0 \models \neg\langle\pi\rangle\langle\pi^*\rangle B$
Induction hypothesis with $C = \langle\pi^*\rangle B$ yields $u_1 \models \neg\langle\pi^*\rangle B$
Repeat this argument to obtain $u_k \models \neg\langle\pi^*\rangle B$
$u_k \models \neg B$ by the tautology. $s_2 \models \neg B$ via $u_k \sim_\Gamma s_2$

Theorem

The problem to decide satisfiability for an arbitrary PFml formula F is decidable.

Proof

Try simultaneously to derive $\neg F$ using Harel's calculus and to find a finite model for F by exhaustive search.
If F is satisfiable we will find a finite model for it. If F is not satisfiable we will find a finite derivation for $\neg F$.

If you do not wish to use the completeness result of Harel's calculus, you can use the finite bound n_F on the size of the Fischer-Ladner closure and exhaustively search through all Kripke structures upto size n_F.

Related Results

The problem to decide for $F, G \in PFml$ wether $G \vdash F$ holds is decidable.

Proof Use the deduction theorem $G \vdash F$ iff $\vdash G \rightarrow F$.

The problem to decide for $F, G \in PFml$ wether $G \vdash^g F$ holds is undecidable.

Meyer, Strett, and Mirowska 1981.

Theorem For $F, G \in PFml$

$$G \vdash^g F \text{ iff } \vdash [(a_1 \cup \ldots \cup a_n)^*]G \rightarrow F$$

where $a_1 \ldots a_n$ are all atomic programs occuring in F or G.

Nonstandard Propositional Kripke Structures

$(u,u') \in \rho(a),\ a \in AP$	iff	$(u,u') \in \rho(a)$
$(u,u') \in \rho(\pi_1;\pi_2)$	iff	there exists $w \in S$ with $(u,w) \in \rho(\pi_1)$ and $(w,u') \in \rho(\pi_2)$
$(u,u') \in \rho(\pi_1 \cup \pi_2)$	iff	$(u,u') \in \rho(\pi_1)$ or $(u,u') \in \rho(\pi_2)$
$(u,u') \in \rho(\pi^*)$	iff	there exists n and $u_1, \dots u_n \in S$ such that $u_1 = u$ and $u_n = u'$ and $(u_i, u_{i+1}) \in \rho(\pi)$ for $1 \leq i < n$
$(u,u') \in \rho(con?)$	iff	$u = u'$ and $u \models\ con$

replace by

$\rho(\pi^*)$ is reflexive and transitive and $\rho(\pi) \subseteq \rho(\pi^*)$ and satisfies

$s \models [a^*]B \Leftrightarrow s \models B \wedge [a;a^*]B$

$s \models [a^*]B \Leftrightarrow s \models B \wedge [a^*](B \to [a]B)$

Theorem

Nonstandard and standard Kripke structures have the same tautologies.

Propositional Kripke Structures

Alternatives

A propositional Kripke structure $\mathcal{K} = (S, \models, \rho)$ is determined by:

S	the set of states
$\models \subseteq (S \times PVar)$	evaluation of propositional atoms in states
$\rho : AP \to S \times S$	the accessibility relations for atomic programs

$S = 2^{PVar}$ the set of states

- *Equivalent* to old setting with restriction:
  for all $a \in AP$, all $s_1, s_2 \in S$:
  if $(s_1 \models p \Leftrightarrow s_2 \models p)$ for all $p \in PVar$
  then $(s_1, s) \in \rho(a)$ iff $(s_2, s) \in \rho(a)$.
- Strictly larger set of tautologies.
- Obviously decidable.

Dynamic Logic
Lecture 3: Completeness

Prof. P.H. Schmitt

Institut für Theoretische Informatik
Fakultät für Informatik
Universität Karlsruhe (TH)

Logic Summer School, Canberra, February, 2009

Failure of the Compactness Theorem

The (infinite) set of DL formulas

$$\{\langle \textbf{while } p(x) \textbf{ do } x := f(x)\rangle \mathbf{1}\} \cup \{p(f^n(x)) \mid n \geq 0\}$$

is not satisfiable, but every finite subset is.

Consequence
Full first-order Dynamic Logic ist nor axiomatisable.

An Infinitary Calculus

Axioms

Axioms for first-order Logic

Axioms for PDL

$\langle x := t \rangle F \quad \leftrightarrow \quad F[x/t]$ for all first-order F

Rules

$$\frac{F,\ F \rightarrow G}{G}$$ (modus ponens)

$$\frac{F}{[\pi]F} \qquad \frac{F}{\forall x F}$$ (generalisations)

$$\frac{G \rightarrow [\pi^n]F \text{ for all } n}{G \rightarrow [\pi^*]F}$$ for any first-order formula F (infinitary convergence)

Theorem For any formula F

F is a tautology iff $\vdash_{\mathsf{INF}} F$

(Harel 1984).

Arithmetic Completeness

Axioms

All first-order formulas valid in $\mathbb{N}$

Axioms for PDL

$\langle x := t\rangle F \quad \leftrightarrow \quad F[x/t]$ for all first-order F

Rules

$$\frac{F,\ F \rightarrow G}{G}$$ (modus ponens)

$$\frac{F}{[\pi]F} \qquad \frac{F}{\forall x F}$$ (generalisations)

$$\frac{F(n+1) \rightarrow \langle\pi\rangle F(n) \text{ for all } n}{F(n) \rightarrow \langle\pi^*\rangle F(0)}$$ for any first-order formula F (convergence)

Theorem For any formula F

F is $\mathbb{N}$-valid iff $\vdash_{\mathbb{N}} F$

Arithmetic Completeness

Main Idea of the Proof

Coding Lemma

For every DL formula F there is a first-order formula F_L such that

$$(\mathcal{K}_{\mathbb{N}}, u) \models F \text{ iff } (\mathbb{N}, u) \models F_L$$

Digression

Coding of Pairs and Finite Sequences

There are formulas $first$ and snd in the vocabulary of $\mathbb{N}$ such that:

$$\mathbb{N} \models \forall a \forall b \exists k (\forall x (first(k,x) \leftrightarrow x = a) \land \forall x (snd(k,x) \leftrightarrow x = b))$$

Let

$$\begin{array}{lcl} k & = & \frac{1}{2}((a+b)(a+b+1)) + a \\ first(u,x) & \equiv & \exists z (u = \frac{1}{2}((x+z)(x+z+1)) + x \\ snd(u,x) & \equiv & \exists z (u = \frac{1}{2}((z+x)(z+x+1)) + z \end{array}$$

There is a formula seq in the vocabulary of $\mathbb{N}$ such that for every $n \in \mathbb{N}$ and any sequence $k_0, \ldots, k_{n-1}$ there is $k \in \mathbb{N}$ satisfying for each i, $0 \leq i < n$

$$\mathbb{N} \models \forall x (seq(k,i,x) \leftrightarrow x = k_i)$$

Example to Coding Lemma

$F(x, y) \equiv \langle (x > 0)?; x := x - 1)^* \rangle \; x = 0$ Compute F_L

$$
\begin{aligned}
F_L \quad \equiv \quad & \exists n \exists k (x = seq(k, 0) \; \wedge 0 = seq(k, n) \; \wedge \\
& \forall i (0 \leq i < n \to seq(k, i) > 0 \; \wedge \\
& seq(k, i + 1) = seq(k, i)) - 1)) \\
\equiv \quad & \exists n \exists k (\forall z (seq(k, 0, z) \to x = z) \; \wedge \\
& \forall z (seq(k, n, z) \to 0 = z) \; \wedge \\
& \forall i \forall u \forall w (0 \leq i < n \wedge seq(k, i, u) \wedge seq(k, i + 1, w) \\
& \; \to u > 0 \; \wedge w = u - 1)
\end{aligned}
$$

Notation $F_L \equiv \exists n F_0(n)$

An Example Derivation

$\vdash_{\mathbb{N}} \langle\alpha^*\rangle\ x = 0$

1	$\vdash_{\mathbb{N}} \exists n F_0(n)$	since $\mathbb{N} \models \exists n F_0(n)$
2	$\vdash_{\mathbb{N}} F_0(0) \rightarrow x = 0$	since $\mathbb{N} \models F_0(0) \rightarrow x = 0$
3		$\mathbb{N} \models F_0(n+1) \rightarrow \langle\alpha\rangle F_0(n)$
4	$F_0(n+1) \rightarrow \langle\alpha\rangle F_0(n)$	from 3 with IndHyp
5	$F_0(n) \rightarrow \langle\alpha^*\rangle F_0(0)$	by convergence rule from 4
6	$\forall n(F_0(n) \rightarrow \langle\alpha^*\rangle F_0(0))$	by generalisation rule from 5
7	$\exists n(F_0(n)) \rightarrow \langle\alpha^*\rangle F_0(0)$	first-order tautology
8	$\langle\alpha^*\rangle F_0(0)$	$1, 7$ and modus ponens
9	$\langle\alpha^*\rangle x = 0$	$2, 8$ and modus ponens

with $\alpha \equiv x > 0?; x := x - 1$.

Soundness

The assignment axiom

$$\langle x := t \rangle F \leftrightarrow F[x/t] \quad \text{for first-order } F$$

is universally valid, since for every structure $\mathcal{M}$ and state u

$$(\mathcal{M}, u') \models F \quad \text{iff} \quad (\mathcal{M}, u) \models F[x/t]$$

with:
$$u'(y) = \begin{cases} u(y) & \text{if } y \neq x \\ val_{\mathcal{M},u}(t) & \text{if } y = x \end{cases}$$

This is known as the Substitution Lemma.
It only works if the substitution t for x in F is collision free, i.e.;
t does not contain a variable z such that there is an occurence of x in F within the scope of a quantifier $\forall z$ or $\exists z$.
This can always be achieved by renaming bound variables.

Dynamic Logic
Lecture 4: Typed First-Order Logic

Prof. P.H. Schmitt

Institut für Theoretische Informatik
Fakultät für Informatik
Universität Karlsruhe (TH)

Logic Summer School, Canberra, February, 2009

First-Order Logic For Realistic Program Verification

What is Missing?

- Types (sorts)
- A method to deal with partial functions
- Programming language specific constructs
- Front-end for specification languages in use
- Open architecture

An Example Program

Adapted from

Object-Oriented Software Development by Xiaoping Jia

```
public class Point{
 int x,y;
 public void move(int dx, int dy){
     x = x + dx;
     y = y + dy;
   }
 public boolean equals(Object other){
   if (other != null && other instanceof Point)
      {Point p = (Point) other;
       return (x == p.x && y == p.y);
       }
   else {return false;}
 }}
```

Added Expressiveness

$$\forall\ Point\ p; \forall\ Object\ ob; ($$
$$p \neq\ null \land ob \neq\ null \land ob \sqsubseteq Point \rightarrow$$
$$\{pp := p\}\{pob := ob\}\langle b = pp.equals(pob)\rangle$$
$$b = \mathbf{1} \leftrightarrow (p.x \doteq (Point)ob.x \land p.y \doteq (Point)ob.y))$$

Sorts, e.g., $Point$, $Object$, int, $boolean$
Subsort relations, e.g., $Point \sqsubseteq Object$.

Added Expressiveness

$$\forall\ Point\ p; \forall\ Object\ ob; ($$
$$p \neq\ null \wedge ob \neq\ null \wedge ob \sqsubseteq Point \rightarrow$$
$$\{pp := p\}\{pob := ob\}\langle b = pp.equals(pob)\rangle$$
$$b = \mathbf{1} \leftrightarrow (p.x \doteq (Point)ob.x \wedge p.y \doteq (Point)ob.y))$$

Sorted logical variables.
Sorted program variables.

Added Expressiveness

$$\forall\ Point\ p; \forall\ Object\ ob; ($$
$$p \neq\ null \land ob \neq\ null \land ob \sqsubseteq Point \rightarrow$$
$$\{pp := p\}\{pob := ob\}\langle b = pp.equals(pob)\rangle$$
$$b = \mathbf{1} \leftrightarrow (p.x \doteq (Point)ob.x \land p.y \doteq (Point)ob.y))$$

Sorted symbols: $x, y : Point \rightarrow int$,
$equals : Object \times\ Object \rightarrow boolean$

Alternative syntax: $p.x$ instead of $x(p)$
$pp.equals(pob)$ instead of $equals(pp, pob)$

Added Expressiveness

$$\forall\ Point\ p; \forall\ Object\ ob; ($$
$$p \neq\ null \land ob \neq\ null \land ob \sqsubseteq Point \rightarrow$$
$$\{pp := p\}\{pob := ob\}\langle b = pp.equals(pob)\rangle$$
$$b = \mathbf{1} \leftrightarrow (p.x \doteq (Point)ob.x \land p.y \doteq (Point)ob.y))$$

Type related operations: unary is-of-type relations,
unary *cast* operations

Difference between dynamic and static type becomes an issue.

Added Expressiveness

$$\forall\ Point\ p; \forall\ Object\ ob; ($$
$$p \neq\ null \land ob \neq\ null \land ob \sqsubseteq Point \rightarrow$$
$$\{pp := p\}\{pob := ob\}\langle b = pp.equals(pob) \rangle$$
$$b = \mathbf{1} \leftrightarrow (p.x \doteq (Point)ob.x \land p.y \doteq (Point)ob.y))$$

Programming language specific constructs.

Type Hierarchy

Before declaring the function and predicate symbols of a first-order language a type hierchy $(\mathcal{T}, \sqsubseteq)$ has to be fixed.

It consists of the set $\mathcal{T}$ of available types and a subtype relation $\sqsubseteq$ on $\mathcal{T}$.

We assume that every type hierachy

$\bot \in \mathcal{T}$	empty type
$\top \in \mathcal{T}$	universal type

$\mathcal{T}_a \subseteq \mathcal{T}$ set of abstract types.

Intention: Every element of an abstract type is also an element of one of its strict subtypes.

Syntax of Typed Predicate Logic

Given: A type hierarchy $(\mathcal{T}, \sqsubseteq)$, a set of types with a subsort relation.

Given: a sorted signature Σ

We define sets $\{Term_\Sigma^A\}_{A \in \mathcal{T}}$ of *terms of (static) type* A:

- $x \in Term_\Sigma^A$ for any variable $x : A \in Var$,
- $f(t_1, \ldots, t_n) \in Term_\Sigma^A$
  for any function symbol $f : A_1, \ldots, A_n \to A$,
  and terms $t_i \in Term_\Sigma^{A'_i}$ with $A'_i \sqsubseteq A_i$ for $i = 1, \ldots, n$,
- $p(t_1, \ldots, t_n)$ is an atomic formulas
  for any predicate symbol $p : A_1, \ldots, A_n$ and terms $t_i \in Term_\Sigma^{A'_i}$ with $A'_i \sqsubseteq A_i$ for $i = 1, \ldots, n$
- Rest unchanged.

Models of Typed Predicate Logic

Given a type hierarchy and a signature, a model is determined by

- a *domain* $\mathcal{D}$,
- a *dynamic type function* $\delta : \mathcal{D} \to \mathcal{T}_d$, and
- an *interpretation* $\mathcal{I}$,

such that for $\mathcal{D}^A := \{d \in \mathcal{D} \mid \delta(d) \sqsubseteq A\}$, it holds that

- $\mathcal{D}^A$ is non-empty for all $A \in \mathcal{T}_d$,
- for any function symbol $f : A_1, \ldots, A_n \to A$,

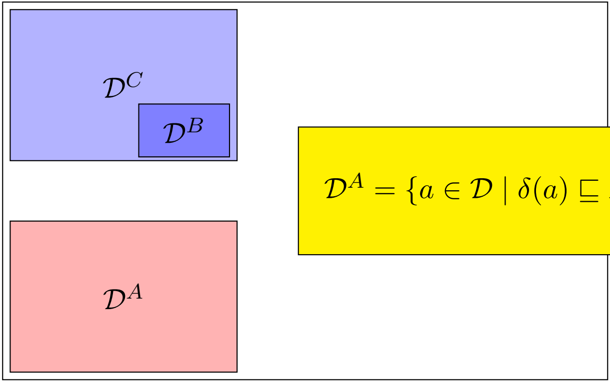
$$\mathcal{I}(f) : \mathcal{D}^{A_1} \times \ldots \times \mathcal{D}^{A_n} \to \mathcal{D}^A \quad ,$$

- for any predicate symbol $p : A_1, \ldots, A_n$,

$$\mathcal{I}(p) \subseteq \mathcal{D}^{A_1} \times \ldots \times \mathcal{D}^{A_n} \quad .$$

- for type predicates, $\mathcal{I}(\sqsubseteq A) = \mathcal{D}^A$,
- for type casts, $\mathcal{I}((A))(x) = x$ if $\delta(x) \sqsubseteq A$, otherwise $\mathcal{I}((A))(x)$ may be an arbitrary but fixed element of $\mathcal{D}^A$.

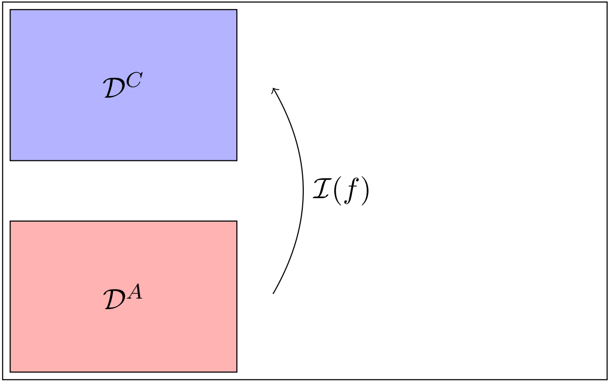
A View of the Domain



$$\mathcal{D}^A = \{a \in \mathcal{D} \mid \delta(a) \sqsubseteq A\}$$

A in $\mathcal{T}_d$ $\quad \mathcal{T}_d \subseteq \mathcal{T}$ set of non-abstract types.
C in $\mathcal{T}_d$ with $A \sqcap C = \bot$
B in $\mathcal{T}_d$ with $B \sqsubseteq C$

A View of the Domain



A in $\mathcal{T}_d$, C in $\mathcal{T}_d$ with $A \sqcap C = \bot$
$f : A \to C$
$\mathcal{I}(f)$ not defined outside $\mathcal{D}^A$.

Semantics of Typed Predicate Logic

Let $\mathcal{M} = (\mathcal{D}, \delta, \mathcal{I})$ be a model, and β a variable assignment.

The evaluation of terms $t \quad \mathrm{val}_{\mathcal{M}}(\beta, t)$

and the interpretation of formulas $F \quad (\mathcal{M}, \beta) \models F$

are inductively defined as usual.

Examples

Subtypes

Let A be an abstract type and $A_1, \ldots A_k$ all its immediate subtypes. Furthermore let x be a variable of type A. Then

$$\forall x.(x \sqsubseteq A <\!\!-\!\!> x \sqsubseteq A_1 \mid \ldots \mid x \sqsubseteq Ak)$$

is logically valid.

If A is a non-abstract type then

$$\forall x.(x \sqsubseteq A <\!\!-\!\!> x \sqsubseteq A_1 \mid \ldots \mid x \sqsubseteq A_k)$$

is satisfiable, but not logically valid.

Logical validity depends on the type hierarchy.
But if this stays fixed it does not depend on the signature.

Examples

The type predicates

Let A be a non-empty type, $x : A$ and $y : \top$, then

$$\forall y.(y \sqsubseteq A <\!\!-\!\!> \exists x.(x \doteq y))$$
$$\forall y.(y \sqsubseteq A <\!\!-\!\!> (A)y \doteq y)$$

are both logically valid.

This shows that the predicates $y \sqsubseteq A$ could be eliminated without reducing the expressive power of our language.

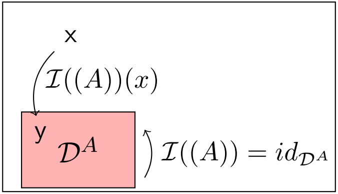
Undefined Values

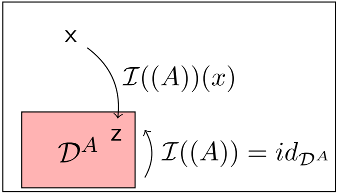
handled by underspecification

Consider the cast function $(A) : \top \to A$.

Interpretation $\mathcal{M}_1$

Interpretation $\mathcal{M}_2$





Examples

Undefined Values

Let x, y be variables of type $\top$, A a non-empty type.
The following formulas are logically valid

$$\forall y.\exists x.((A)y \doteq x)$$
$$\forall y.((A)y \doteq (A)y)$$

while

$$\forall y.\forall x.(\neg x \sqsubseteq A \wedge \neg y \sqsubseteq A \rightarrow (A)x \doteq (A)y)$$
$$\forall x.(\neg x \sqsubseteq A \rightarrow (A)x \doteq c)$$

are satisfiable but not logically valid.

Sequents

Definition

A *sequent* is a pair of sets of closed formulae written as

$$\underbrace{\phi_1, \ldots, \phi_m}_{\text{antecedent}} \Longrightarrow \underbrace{\psi_1, \ldots, \psi_n}_{\text{succedent}}$$

Shorthand:

$$\Gamma, \phi \Longrightarrow \psi, \Delta$$

Sequents

Semantics

A sequent

$$\phi_1, \ldots, \phi_m \Longrightarrow \psi_1, \ldots, \psi_n \quad .$$

is *valid* iff the formula

$$\phi_1 \ \& \ \ldots \ \& \ \phi_m \ \text{->} \ \psi_1 \mid \ldots \mid \psi_n$$

is valid.

The empty conjunction is set to true while the empty disjunction is set to false.

Sequents

Examples

$\Longrightarrow \psi$ is valid iff $\mathrm{true} \mathrel{->} \psi$ is valid
iff ψ is valid

$\phi \Longrightarrow$ is valid iff $\phi \mathrel{->} \mathrm{false}$ is valid
iff $\neg\phi$ is valid

$\Longrightarrow$ is valid iff $\mathrm{true} \mathrel{->} \mathrm{false}$ is valid
Thus: $\Longrightarrow$ is not valid

Rules

Soundness

$$\frac{\Gamma' ==> \Delta'}{\Gamma ==> \Delta} \quad \text{or} \quad \frac{\Gamma_1 ==> \Delta_1 \quad \Gamma_2 ==> \Delta_2}{\Gamma ==> \Delta}$$

A rule is sound if

validity of $\Gamma' ==> \Delta'$ implies validity of $\Gamma ==> \Delta$

or

validity of $\Gamma_1 ==> \Delta_1$ and validity of $\Gamma_2 ==> \Delta_2$ implies validity of $\Gamma ==> \Delta$

Propositional Rules

$$\text{and}-\text{left}\ \frac{\Gamma, \phi, \psi \Longrightarrow \Delta}{\Gamma, \phi \mathbin{\&} \psi \Longrightarrow \Delta} \qquad \text{and}-\text{right}\ \frac{\Gamma \Longrightarrow \phi, \Delta \qquad \Gamma \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \phi \mathbin{\&} \psi, \Delta}$$

$$\text{or}-\text{right}\ \frac{\Gamma \Longrightarrow \phi, \psi, \Delta}{\Gamma \Longrightarrow \phi \mid \psi, \Delta} \qquad \text{or}-\text{left}\ \frac{\Gamma, \phi \Longrightarrow \Delta \qquad \Gamma, \psi \Longrightarrow \Delta}{\Gamma, \phi \mid \psi \Longrightarrow \Delta}$$

$$\text{imp}-\text{right}\ \frac{\Gamma, \phi \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \phi \mathbin{-\!>} \psi, \Delta} \qquad \text{imp}-\text{left}\ \frac{\Gamma \Longrightarrow \phi, \Delta \qquad \Gamma, \psi \Longrightarrow \Delta}{\Gamma, \phi \mathbin{-\!>} \psi \Longrightarrow \Delta}$$

$$\text{not}-\text{left}\ \frac{\Gamma \Longrightarrow \phi, \Delta}{\Gamma, !\,\phi \Longrightarrow \Delta} \qquad \text{not}-\text{right}\ \frac{\Gamma, \phi \Longrightarrow \Delta}{\Gamma \Longrightarrow !\,\phi, \Delta}$$

Classical Quantifier Rules

$$\text{all} - \text{right} \; \frac{\Gamma \Longrightarrow [x/c](\phi), \Delta}{\Gamma \Longrightarrow \forall x.\phi, \Delta}$$

with $c : \to A$ a new constant, if $x : A$.

$$\text{all} - \text{left} \; \frac{\Gamma, \forall x.\phi, [x/t](\phi) \Longrightarrow \Delta}{\Gamma, \forall x.\phi \Longrightarrow \Delta}$$

with $t \in Term^{A'}$ ground, $A' \sqsubseteq A$, if $x : A$.

$$\text{ex} - \text{left} \; \frac{\Gamma, [x/c](\phi) \Longrightarrow \Delta}{\Gamma, \exists x.\phi \Longrightarrow \Delta}$$

with $c : \to A$ a new constant, if $x : A$.

$$\text{ex} - \text{right} \; \frac{\Gamma \Longrightarrow \exists x.\phi, [x/t](\phi), \Delta}{\Gamma \Longrightarrow \exists x.\phi, \Delta}$$

with $t \in Term^{A'}$ ground, $A' \sqsubseteq A$, if $x : A$.

Closure Rules

$$\text{close} \; \frac{}{\Gamma, \phi \Longrightarrow \phi, \Delta}$$

$$\text{close} - \text{false} \; \frac{}{\Gamma, \mathrm{false} \Longrightarrow \Delta} \quad \text{close} - \text{true} \; \frac{}{\Gamma \Longrightarrow \mathrm{true}, \Delta}$$

An Example Derivation

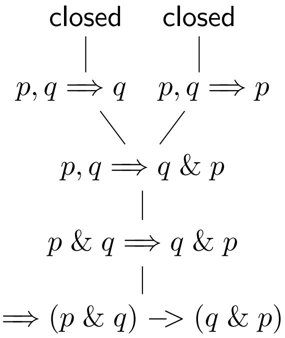
Let us try to prove that $(p \,\&\, q) \rightarrow (q \,\&\, p)$ is valid.

$$\text{imp} - \text{right} \quad \frac{\Gamma, \phi \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \phi \rightarrow \psi, \Delta}$$

$$\text{and} - \text{left} \quad \frac{\Gamma, \phi, \psi \Longrightarrow \Delta}{\Gamma, \phi \,\&\, \psi \Longrightarrow \Delta}$$

$$\text{and} - \text{right} \quad \frac{\Gamma \Longrightarrow \phi, \Delta \qquad \Gamma \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \phi \,\&\, \psi, \Delta}$$

$$\text{close} \quad \frac{}{\Gamma, \phi \Longrightarrow \phi, \Delta}$$



Equality Rules

$$\text{eq}-\text{left}\quad \frac{\Gamma, t_1 \doteq t_2, [z/t_1](\phi), [z/t_2](\phi) \Longrightarrow \Delta}{\Gamma, t_1 \doteq t_2, [z/t_1](\phi) \Longrightarrow \Delta}$$

if $\sigma(t_2) \sqsubseteq \sigma(t_1)$.

$$\text{eq}-\text{right}\quad \frac{\Gamma, t_1 \doteq t_2 \Longrightarrow [z/t_2](\phi), [z/t_1](\phi), \Delta}{\Gamma, t_1 \doteq t_2 \Longrightarrow [z/t_1](\phi), \Delta}$$

if $\sigma(t_2) \sqsubseteq \sigma(t_1)$.

$$\text{eq}-\text{symm}-\text{left}\quad \frac{\Gamma, t_2 \doteq t_1 \Longrightarrow \Delta}{\Gamma, t_1 \doteq t_2 \Longrightarrow \Delta}$$

$$\text{eq}-\text{close}\quad \frac{}{\Gamma \Longrightarrow t \doteq t, \Delta}$$

Pitfalls with Equality Rules

$$\mathsf{eq-left-wrong}\ \frac{\Gamma, t_1 \doteq t_2, [z/t_1](\phi), [z/t_2](\phi) \Longrightarrow \Delta}{\Gamma, t_1 \doteq t_2, [z/t_1](\phi) \Longrightarrow \Delta}$$

Consider

1. types $B \sqsubseteq A$, but $B \neq A$,
2. constants $a : \to A$ and $b : \to B$,
3. a predicate $p : B$,

Applying the eq-left-wrong rule on the sequent

$$b \doteq a, p(b) \Longrightarrow$$

yields

$$b \doteq a, p(b), p(a) \Longrightarrow$$

But $p(a)$ is not a correctly typed formula!

Equality Rules (continued)

eq − left′

$$\frac{\Gamma, t_1 \doteq t_2, [z/t_1](\phi), [z/(A)t_2](\phi) \Longrightarrow \Delta}{\Gamma, t_1 \doteq t_2, [z/t_1](\phi) \Longrightarrow \Delta}$$

with $A := \sigma(t_1)$.

eq − right′

$$\frac{\Gamma, t_1 \doteq t_2 \Longrightarrow [z/(A)t_2](\phi), [z/t_1](\phi), \Delta}{\Gamma, t_1 \doteq t_2 \Longrightarrow [z/t_1](\phi), \Delta}$$

with $A := \sigma(t_1)$.

Typing Rules

$$\text{type}-\text{eq}\quad \frac{\Gamma, t_1 \doteq t_2, t_2 \sqsubseteq \sigma(t_1), t_1 \sqsubseteq \sigma(t_2) \Longrightarrow \Delta}{\Gamma, t_1 \doteq t_2 \Longrightarrow \Delta}$$

$$\text{type}-\text{glb}\quad \frac{\Gamma, t \sqsubseteq A, t \sqsubseteq B, t \sqsubseteq A \sqcap B \Longrightarrow \Delta}{\Gamma, t \sqsubseteq A, t \sqsubseteq B \Longrightarrow \Delta}$$

$$\text{type}-\text{static}\quad \frac{\Gamma, t \sqsubseteq \sigma(t) \Longrightarrow \Delta}{\Gamma \Longrightarrow \Delta}$$

$$\text{type}-\text{abstract}\quad \frac{\Gamma, t \sqsubseteq A, t \sqsubseteq B_1 \mid \cdots \mid t \sqsubseteq B_k \Longrightarrow \Delta}{\Gamma, t \sqsubseteq A \Longrightarrow \Delta}$$

with $A \in \mathcal{T} \setminus \mathcal{T}_\lceil$ and $B_1, \ldots, B_k$ the direct subtypes of A.

Casting Rules

cast − add − left

$$\frac{\Gamma, [z/t](\phi), t \sqsubseteq A, [z/(A)t](\phi) \Longrightarrow \Delta}{\Gamma, [z/t](\phi), t \sqsubseteq A \Longrightarrow \Delta}$$

where $A \sqsubseteq \sigma(t)$.

cast − add − right

$$\frac{\Gamma, t \sqsubseteq A \Longrightarrow [z/(A)t](\phi), [z/t](\phi), \Delta}{\Gamma, t \sqsubseteq A \Longrightarrow [z/t](\phi), \Delta}$$

where $A \sqsubseteq \sigma(t)$.

cast − del − left

$$\frac{\Gamma, [z/t](\phi), [z/(A)t](\phi) \Longrightarrow \Delta}{\Gamma, [z/(A)t](\phi) \Longrightarrow \Delta}$$

where $\sigma(t) \sqsubseteq A$.

cast − del − right

$$\frac{\Gamma \Longrightarrow [z/t](\phi), [z/(A)t](\phi), \Delta}{\Gamma \Longrightarrow [z/(A)t](\phi), \Delta}$$

where $\sigma(t) \sqsubseteq A$.

Casting Rules (continued)

$$\mathsf{cast-type-left}\ \frac{\Gamma, (A)t \sqsubseteq B, t \sqsubseteq A, t \sqsubseteq B \Longrightarrow \Delta}{\Gamma, (A)t \sqsubseteq B, t \sqsubseteq A \Longrightarrow \Delta}$$

$$\mathsf{cast-type-right}\ \frac{\Gamma, t \sqsubseteq A \Longrightarrow t \sqsubseteq B, (A)t \sqsubseteq B, \Delta}{\Gamma, t \sqsubseteq A \Longrightarrow (A)t \sqsubseteq B, \Delta}$$

$$\mathsf{close-subtype}\ \frac{}{\Gamma, t \sqsubseteq A \Longrightarrow t \sqsubseteq B, \Delta}$$

with $A \sqsubseteq B$.

$$\mathsf{close-empty}\ \frac{}{\Gamma, t \sqsubseteq \bot \Longrightarrow \Delta}$$

Pitfalls with Typing Rules

$$\text{wrong} - \text{cast} - \text{del} - \text{left} \; \frac{\Gamma, [z/(A)t](\phi), t \sqsubseteq A, [z/t](\phi) \Longrightarrow \Delta}{\Gamma, [z/(A)t](\phi), t \sqsubseteq A \Longrightarrow \Delta}$$

Correctness Theorem

Assume

1. a fixed type hierarchy,
2. an admissible signature,
3. a sequent $\Gamma \Longrightarrow \Delta$ and
4. a partial structure $\mathcal{M}_0$

If there is a closed sequent proof for $\Gamma \Longrightarrow \Delta$ then $\Gamma \Longrightarrow \Delta$ is $\mathcal{M}_0$-valid.

Proof: By induction on the length of a closed proof provided that all used rules are $\mathcal{M}_0$-sound.

For the uninterpreted case also completeness can be proved, see M. Giese, *A Calculus for Type Predicates and Type Coercion*, Proceeding Tableaux 2005, pp 123–137, Springer LNAI Vol 3702.

Dynamic Logic
Lecture 5: Updates

Prof. P.H. Schmitt

Institut für Theoretische Informatik
Fakultät für Informatik
Universität Karlsruhe (TH)

Logic Summer School, Canberra, February4, 2009

Contents

Setting the Stage

- So far, the only constructs of Dynamic Logic referring to state changes were the modal operators $\langle\pi\rangle$ and $[\pi]$.
- We now introduce the new syntactic category *Updates* of updates. For any DL formula φ and any update $u \in$ *Updates*

$$\{u\}\varphi$$

  will also be a DL formula, and for any DL term t

$$\{u\}t$$

  will also be a DL term.
- For every state $S = (\mathcal{A}, \beta)$ we will define an updated state

$$\{u\}S.$$

- $S \models \{u\}\varphi$ iff $\{u\}S \models \varphi$
- $val_S(\{u\}t) = val_{\{u\}S}(t)$

Elementary Updates

If

f is an n-ary non-rigid function symbol with result type A,
$t_1, \ldots, t_n$ are terms with types matching the signature of f,
and
t a DL Term of type A', $A' \sqsubseteq A$,

then

$$f(t_1, \ldots, t_n) := t$$

is an elementary update.

Definition

elementary updates as seen,

sequential updates $u_1 \,; u_2$

parallel updates $u_1 \,||\, u_2$

update application $\{u_1\}\, u_2$

quantified updates `for` x; φ; u_1

where u_1 and u_2 are updates, x is a logical variable, and φ is a DL formula.

Quantified Updates

General form:

$$\texttt{for } x;\ \varphi;\ u$$

Typical example:

$$\texttt{for } n;\ 0 \leq n \wedge n \leq max;\ h(n) := 0$$

Intended meaning:

For all n between 0 and max set the value of $h(n)$ to 0.

Semantics of Updates

First Attempt

Elementary updates:

Let $S = (\mathcal{A}, \beta)$ be a state,
Then

$$\{f(t_1, \ldots, t_n) := t\}S = (\mathcal{B}, \beta)$$

where $\mathcal{B}$ coincides with $\mathcal{A}$ except that

$$f^{\mathcal{B}}(val_{\mathcal{A}}(\beta, t_1), \ldots, val_{\mathcal{A}}(\beta, t_n)) = val_{\mathcal{A}}(\beta, t)$$

Sequential updates:

$$\{u_1; u_2\}S = \{u_2\}(\{u_1\}S)$$

Semantics of Updates

First Attempt (cont.)

Problems:

Parallel updates may contain clashes!

e.g.

$$f(a) := 0 \,||\, f(b) := 1$$

when $S \models a \doteq b$.

Also quantified updates may lead to clashes.

e.g.

$$\mathtt{for}\ x;\ 0 \leq x \wedge x \leq 1;\ g(f(x)) := x$$

when $S \models f(0) \doteq f(1)$.

Two Step Definition

Of The Semantics of Updates

syntactic update		set of semantic updates		application of semantic updates
u	$\longrightarrow$	$U = val_{\mathcal{A}}(\beta, u)$	$\longrightarrow$	$U(S) = (\mathcal{B}, \beta)$
$S = (\mathcal{A}, \beta)$				$\{u\}S$
start state				updated state

Semantic Updates
Definition

A semantic update (for a state S) is a triple

$$(\underbrace{f, (d_1, \ldots, d_n)}_{location}, \underbrace{d}_{value})$$

such that

- $f : A_1, \ldots, A_n \to A \in \mathrm{FSym}_{nr}$,
- $d_i \in \mathcal{D}^{A_i}$ $(1 \leq i \leq n)$, and
- $d \in \mathcal{D}^A$

Consistent Semantic Updates

Definition

A set U of semantic updates is called consistent
if for any two

$$(f, (d_1, \ldots, d_n), d), (f', (d'_1, \ldots, d'_m), d') \in U$$

with

$$f = f', n = m, \text{ and } d_i = d'_i \ (1 \leq i \leq n)$$

we get

$$d = d'.$$

i.e. Equal locations are assigned equal values

Application of Semantic Updates

Definition

Let U be a consistent set of semantic updates and $S = (\mathcal{A}, \beta)$ a state.

The updated state is

$$U(S) = (\mathcal{B}, \beta)$$

where $\mathcal{B}$ coincides with $\mathcal{A}$ except that

$$f^{\mathcal{B}}(d_1, \ldots, d_n) = d$$

for all $(f, (d_1, \ldots, d_n), d) \in U$.

Note:
Updates do not affect the assignment β to free logical variables.
Logical variables are considered rigid.
Program variables are non-rigid constants.

Evaluating Syntactic Updates

Definition

- $val_{\mathcal{A}}(\beta, f(t_1, \ldots, t_n) := t) = \{\, (f, (d_1, \ldots, d_n), d) \,\}$
  with $d_i = val_{\mathcal{A}}(\beta, t_i)$ $(1 \leq i \leq n)$ and $d = val_{\mathcal{A}}(\beta, t)$,
- $val_{\mathcal{A}}(\beta, u_1\,;u_2) = (U_1 \cup U_2) \setminus C$ where
  $$\begin{aligned} U_1 &= val_{\mathcal{A}}(\beta, u_1) \\ U_2 &= val_{\mathcal{B}}(\beta, u_2) \text{ with } (\mathcal{B}, \beta) = U_1(\mathcal{A}, \beta) \\ C &= \{(f, (d_1, \ldots, d_n), d) \mid \; (f, (d_1, \ldots, d_n), d) \in U_1 \text{ and } \\ &\qquad (f, (d_1, \ldots, d_n), d') \in U_2 \\ &\qquad \text{for some } d' \neq d\}, \end{aligned}$$
- $val_{\mathcal{A}}(\beta, u_1 \,||\, u_2) = (U_1 \cup U_2) \setminus C$ where
  $$\begin{aligned} U_1 &= val_{\mathcal{A}}(\beta, u_1) \\ U_2 &= val_{\mathcal{A}}(\beta, u_2) \\ C &= \{(f, (d_1, \ldots, d_n), d) \mid \; (f, (d_1, \ldots, d_n), d) \in U_1 \text{ and } \\ &\qquad (f, (d_1, \ldots, d_n), d') \in U_2 \\ &\qquad \text{for some } d' \neq d\}, \end{aligned}$$

last win semantics

Evaluating Syntactic Updates

Definition for Quantified Updates

- Let A be the type of x.

  $val_{\mathcal{A}}(\beta, \texttt{for } x;\ \varphi;\ u) =$
  $$\bigcup\{val_{\mathcal{A}}(\beta_x^a, u) \mid a \in \mathcal{D}^A \text{ with } (\mathcal{A}, \beta_x^a) \models \varphi\} \setminus C$$

where:

$$C = \{(f,(d_1,\ldots,d_n),d) \mid \begin{array}{l} \text{there are } a, b \text{ with} \\ (\mathcal{A},\beta_x^a) \models \varphi, (\mathcal{A},\beta_x^b) \models \varphi \\ (f,(d_1,\ldots,d_n),d) \in val_{\mathcal{A}}(\beta_x^a, u) \\ (f,(d_1,\ldots,d_n),d') \in val_{\mathcal{A}}(\beta_x^b, u) \\ \text{and } b \prec a \text{ and } d \neq d'\} \end{array}$$

Here $\prec$ is some well-ordering on $\mathcal{D}^A$ (fixed in advance).

least witness wins semantics

Evaluating Syntactic Updates

For Updates Applied on Updates

- $val_{\mathcal{A}}(\beta, \{u_1\}\ u_2) = \mathsf{val}_{\mathcal{B}}(\beta, u_2)$ with $(\mathcal{B}, \beta) = val_{\mathcal{A}}(\beta, u_1)(\mathcal{A}, \beta)$.

Example

Swapping

Consider the two (syntactic) updates:

$$u_1 = a := b$$
$$u_2 = b := a$$

$(\mathcal{B}, \beta)$	$val_{\mathcal{B}}(\beta, a)$	$val_{\mathcal{B}}(\beta, b)$
$\{u_1\}(\mathcal{A}, \beta)$	$val_{\mathcal{A}}(\beta, b)$	$val_{\mathcal{A}}(\beta, b)$
$\{u_2\}(\mathcal{A}, \beta)$	$val_{\mathcal{A}}(\beta, a)$	$val_{\mathcal{A}}(\beta, a)$
$\{u_1; u_2\}(\mathcal{A}, \beta)$	$val_{\mathcal{A}}(\beta, b)$	$val_{\mathcal{A}}(\beta, b)$
$\{u_1 \mid\mid u_2\}(\mathcal{A}, \beta)$	$val_{\mathcal{A}}(\beta, b)$	$val_{\mathcal{A}}(\beta, a)$
$\{\{u_1\}u_2\}(\mathcal{A}, \beta)$	$val_{\mathcal{A}}(\beta, a)$	$val_{\mathcal{A}}(\beta, b)$

Example

Arity > 0

Consider the two (syntactic) updates:

$$u_1 = a := b$$
$$u_2 = f(a) := b$$

$(\mathcal{B}, \beta)$	$val_{\mathcal{B}}(\beta, a)$	$val_{\mathcal{B}}(\beta, f(a))$	$val_{\mathcal{B}}(\beta, f(b))$
$\{u_1\}(\mathcal{A}, \beta)$	$val_{\mathcal{A}}(\beta, b)$	$val_{\mathcal{A}}(\beta, f(b))$	$val_{\mathcal{A}}(\beta, f(b))$
$\{u_2\}(\mathcal{A}, \beta)$	$val_{\mathcal{A}}(\beta, a)$	$val_{\mathcal{A}}(\beta, b)$	
$\{u_1; u_2\}(\mathcal{A}, \beta)$	$val_{\mathcal{A}}(\beta, b)$	$val_{\mathcal{A}}(\beta, b)$	$val_{\mathcal{A}}(\beta, b)$
$\{u_1 \,\|\, u_2\}(\mathcal{A}, \beta)$	$val_{\mathcal{A}}(\beta, b)$		
$\{\{u_1\}u_2\}(\mathcal{A}, \beta)$	$val_{\mathcal{A}}(\beta, a)$		$val_{\mathcal{A}}(\beta, b)$

$f(a)$ can be a different location in $\mathcal{B}$ than in $\mathcal{A}$! (if a has been changed)
$f(a)$ and $f(b)$ can be the same location ("aliasing")! (if $\mathcal{A}, \beta \models a \doteq b$)

Example
Clashes

Consider the (syntactic) update:

$$u \quad = \quad f(a) := 0 \mid\mid f(b) := 1$$

$(\mathcal{B}, \beta)$	$Clash$	$val_{\mathcal{B}}(\beta, f(a))$	$val_{\mathcal{B}}(\beta, f(b))$
$\{u\}(\mathcal{A}, \beta) \quad \mathcal{A}, \beta \not\models a \doteq b$	no	0	1
$\{u\}(\mathcal{A}, \beta) \quad \mathcal{A}, \beta \models a \doteq b$	yes	1	1

Remember: Last win semantics

Example

Clashes in Quantified Updates

Consider the (syntactic) update:

$$u \quad = \quad \texttt{for}\ x;\ 0 \leq x \wedge x \leq 1;\ g(f(x)) := x$$

$(\mathcal{B}, \beta)$	$Clash$	$val_{\mathcal{B}}(\beta, g(f(0)))$	$val_{\mathcal{B}}(\beta, g(f(1)))$
$\{u\}(\mathcal{A}, \beta)$ $\mathcal{A}, \beta \not\models f(0) \doteq f(1)$	no	0	1
$\{u\}(\mathcal{A}, \beta)$ $\mathcal{A}, \beta \models f(0) \doteq f(1)$	yes	0	0

Remember: Least witness wins semantics (we assume $0 \prec 1$)

Rewrite Rules
for
Evaluating Updates

New Auxiliary Syntax

$$\begin{array}{rcl} \texttt{for}\ x\ \{u\} & = & \texttt{for}\ x;\ true;\ u \\ \texttt{for}\ x;\ \phi;\ u & = & \texttt{for}\ x\ \{\texttt{if}\ \ \phi\ \{u\}\} \end{array}$$

$$val_{\mathcal{B}}(\beta, \text{REJECT}(u_1, u_2)) = \{(f, (d_1, \ldots, d_n), d) \in val_{\mathcal{B}}(\beta, u_1) \mid \text{there is no } d' \text{ with } (f, (d_1, \ldots, d_n), d') \in val_{\mathcal{B}}(\beta, u_2)\}$$

$$val_{\mathcal{B}}(\beta, \text{NON-REC}(u, f, \bar{t})) = \begin{cases} d & \text{if } (f, (val_{\mathcal{B}}(\beta, \bar{t}), d) \in val_{\mathcal{B}}(\beta, u)) \\ f^{\mathcal{B}}(val_{\mathcal{B}}(\beta, \bar{t})) & \text{otherwise} \end{cases}$$

$$(\mathcal{B}, \beta) \models \text{IN-DOM}(f, \bar{t}, u) \Leftrightarrow (f, (val_{\mathcal{B}}(\beta, \bar{t})), d) \in val_{\mathcal{B}}(\beta, u)$$

$$val_{\mathcal{B}}(\beta, min\ x.\ \phi) = \begin{cases} min_{\prec}(A) & \text{if } A \neq \emptyset \\ min_{\prec}(U) & \text{if } A = \emptyset \end{cases}$$

$A = \{d \in U \mid (\mathcal{B}, \beta_x^d) \models \phi\}, U =$ domain for the type of x

Direct Rewrite Rules for Updates

t a term, $\bar{t}$ a tupel of terms, ϕ a formula

$$
\begin{array}{rcll}
\{u\}\, x & \rightarrow & x & x \in Var \quad (R1) \\
\{u\}\, f(\bar{t}) & \rightarrow & \textsc{non-rec}(u, f, \{u\}\, \bar{t}) & \quad (R2) \\
\{u\}\, \textbf{if } \phi \textbf{ then } t_1 \textbf{ else } t_2 & \rightarrow & & \\
\multicolumn{3}{r}{\textbf{if } \{u\}\, \phi \textbf{ then } \{u\}\, t_1 \textbf{ else } \{u\}\, t_2} & \quad (R3) \\
\{u\}\, min\ x.\ \phi & \rightarrow & min\ x.\ \{u\}\, \phi & x \notin fv(u) \quad (R4) \\
\{u\}\, lit & \rightarrow & lit & lit \in \{\mathbf{1}, \mathbf{0}\} \quad (R5) \\
\{u\}\, \phi_1 * \phi_2 & \rightarrow & \{u\}\, \phi_1 * \{u\}\, \phi_2 & * \in \{\wedge, \vee\} \quad (R6) \\
\{u\}\, \neg\phi & \rightarrow & \neg\{u\}\, \phi & \quad (R7) \\
\{u\}\, Qx\phi & \rightarrow & Qx\{u\}\, \phi & Q \in \{\forall, \exists\} \\
 & & & x \notin fv(u) \quad (R8) \\
\{u\}\, t_1 * t_2 & \rightarrow & \{u\}\, t_1 * \{u\}\, t_2 & * \in \{\doteq, <\} \quad (R9)
\end{array}
$$

Strategy: to evalutate $\{u\}\, t$ or $\{u\}\, \phi$ apply the direct rewrite rules above to reduce t and ϕ to the simplest cases, u remains unchanged, then use the rewrite rules for $\textsc{non-rec}(u, f, \bar{s})$.

Rewrite Rules For $\textsc{non-rec}(u, f, \bar{s})$

$$
\begin{array}{rcll}
\textsc{non-rec}(\mathbf{skip}, f, \bar{t}) & \rightarrow & f(\bar{t}) & (R10)\\
\textsc{non-rec}(f(\bar{s}) := r, f, \bar{t}) & \rightarrow & \mathbf{if}\ \bar{t} \doteq \bar{s}\ \mathbf{then}\ r\ \mathbf{else}\ f(\bar{t}) & (R11)\\
\textsc{non-rec}(g(\bar{s}) := r, f, \bar{t}) & \rightarrow & f(\bar{t}) \qquad f \neq g & (R12)\\
\textsc{non-rec}(u_1 \,||\, u_2, f, \bar{t}) & \rightarrow & \begin{array}{l}\mathbf{if}\ \ \textsc{in-dom}(f, \bar{t}, u_2)\\ \mathbf{then}\ \textsc{non-rec}(u_2, f, \bar{t})\\ \mathbf{else}\ \ \textsc{non-rec}(u_1, f, \bar{t})\end{array} & (R13)\\
\textsc{non-rec}(\texttt{if}\ \ \phi\ \{u\}, f, \bar{t}) & \rightarrow & \begin{array}{l}\mathbf{if}\ \ \phi\\ \mathbf{then}\ \textsc{non-rec}(u, f, \bar{t})\\ \mathbf{else}\ \ f(\bar{t})\end{array} & (R14)
\end{array}
$$

$x \notin fv(\bar{t})$ and $r = min\ x.\ \textsc{in-dom}(f, \bar{t}, u)$

$$\textsc{non-rec}(\texttt{for}\ x\ \{u\}, f, \bar{t}) \rightarrow \textsc{non-rec}(\{x/r\}u, f, \bar{t}) \qquad (R15)$$

The rewrite rules for $\textsc{non-rec}(u, f, \bar{s})$ needed to make use of the $\textsc{in-dom}(u, f, \bar{s})$ predicate.
Thus, we need rewrite rules for $\textsc{in-dom}(u, f, \bar{s})$.

Rewrite Rules For $\text{IN-DOM}(u, f, \bar{s})$

$$
\begin{array}{rcll}
\text{IN-DOM}(f, \bar{t}, \mathbf{skip}) & \rightarrow & \mathbf{0} & & (R16) \\
\text{IN-DOM}(f, \bar{t}, f(\bar{s}) := r) & \rightarrow & \bar{t} \doteq \bar{s} & & (R17) \\
\text{IN-DOM}(f, \bar{t}, g(\bar{s}) := r) & \rightarrow & \mathbf{0} & f \neq g & (R18) \\
\text{IN-DOM}(f, \bar{t}, u_1 \,||\, u_2) & \rightarrow & \begin{array}{l} \text{IN-DOM}(f, \bar{t}, u_1) \\ \vee\ \text{IN-DOM}(f, \bar{t}, u_2) \end{array} & & (R19) \\
\text{IN-DOM}(f, \bar{t}, \texttt{if}\ \ \phi\ \{u\}) & \rightarrow & \phi \wedge \text{IN-DOM}(f, \bar{t}, u) & & (R20) \\
\text{IN-DOM}(f, \bar{t}, \texttt{for}\ x\ \{u\}) & \rightarrow & \exists x\ \text{IN-DOM}(f, \bar{t}, u) & x \notin fv(\bar{t}) & (R21)
\end{array}
$$

Example

$$u = f(a) := 0 \,||\, f(b) := 1$$

Example

$u = f(a) := 0 \,||\, f(b) := 1$

$$1 \quad \{u\}\, f(a)$$

Example

$u = f(a) := 0 \,||\, f(b) := 1$

$$\begin{array}{ll} 1 & \{u\}\, f(a) \\ 2 & \equiv \textsc{non-rec}(u, f, \{u\}\, a) \quad (R2) \end{array}$$

Example

$u = f(a) := 0 \,||\, f(b) := 1$

$$
\begin{array}{lll}
1 & \{u\}\, f(a) & \\
2 & \equiv \textsc{non-rec}(u, f, \{u\}\, a) & (R2) \\
4 & \equiv \textsc{non-rec}(u, f, \textsc{non-rec}(u, a, ())) & (R2)
\end{array}
$$

Example

$u = f(a) := 0 \,||\, f(b) := 1$

$$\begin{array}{lll}
1 & \{u\}\ f(a) & \\
2 & \equiv \textsc{non-rec}(u, f, \{u\}\ a) & (R2) \\
4 & \equiv \textsc{non-rec}(u, f, \textsc{non-rec}(u, a, ())) & (R2) \\
5 & \quad \textsc{non-rec}(u, a, ()) &
\end{array}$$

Example

$u = f(a) := 0 \,||\, f(b) := 1$

$$
\begin{array}{lll}
1 & \{u\}\, f(a) & \\
2 & \equiv \textsc{non-rec}(u, f, \{u\}\, a) & (R2) \\
4 & \equiv \textsc{non-rec}(u, f, \textsc{non-rec}(u, a, ())) & (R2) \\
5 & \quad \textsc{non-rec}(u, a, ()) & \\
6 & \quad \equiv \textbf{if } \textsc{in-dom}(a, (), f(b) := 1) & \\
  & \qquad \textbf{then } \textsc{non-rec}(f(b) := 1, a, ()) & \\
  & \qquad \textbf{else } \textsc{non-rec}(f(a) := 0, a, ()) & (R13)
\end{array}
$$

Example

$u = f(a) := 0 \,||\, f(b) := 1$

$$
\begin{array}{lll}
1 & \{u\}\; f(a) & \\
2 & \equiv \textsc{non-rec}(u, f, \{u\}\; a) & (R2) \\
4 & \equiv \textsc{non-rec}(u, f, \textsc{non-rec}(u, a, ())) & (R2) \\
5 & \quad \textsc{non-rec}(u, a, ()) & \\
6 & \quad \equiv \textbf{if } \mathbf{0} & (R18) \\
 & \qquad \textbf{then } \textsc{non-rec}(f(b) := 1, a, ()) & \\
 & \qquad \textbf{else } \textsc{non-rec}(f(a) := 0, a, ()) & (R13)
\end{array}
$$

Example

$u = f(a) := 0 \,||\, f(b) := 1$

$$
\begin{array}{lll}
1 & \{u\}\, f(a) & \\
2 & \;\equiv \textsc{non-rec}(u, f, \{u\}\, a) & (R2) \\
4 & \;\equiv \textsc{non-rec}(u, f, \textsc{non-rec}(u, a, ())) & (R2) \\
5 & \quad \textsc{non-rec}(u, a, ()) & \\
6 & \quad\;\equiv \textbf{if } \mathbf{0} & (R18) \\
  & \qquad\quad \textbf{then } \textsc{non-rec}(f(b) := 1, a, ()) & \\
  & \qquad\quad \textbf{else } \textsc{non-rec}(f(a) := 0, a, ()) & (R13) \\
7 & \quad\;\equiv \textsc{non-rec}(f(a) := 0, a, ()) &
\end{array}
$$

Example

$u = f(a) := 0 \,||\, f(b) := 1$

$$
\begin{array}{lll}
1 & \{u\}\, f(a) & \\
2 & \equiv \text{NON-REC}(u, f, \{u\}\, a) & (R2) \\
4 & \equiv \text{NON-REC}(u, f, \text{NON-REC}(u, a, ())) & (R2) \\
5 & \quad \text{NON-REC}(u, a, ()) & \\
6 & \quad \equiv \textbf{if } \mathbf{0} & (R18) \\
 & \qquad \textbf{then } \text{NON-REC}(f(b) := 1, a, ()) & \\
 & \qquad \textbf{else } \text{NON-REC}(f(a) := 0, a, ()) & (R13) \\
7 & \quad \equiv a & (R12)
\end{array}
$$

Example

$u = f(a) := 0 \,||\, f(b) := 1$

$$
\begin{array}{lll}
1 & \{u\}\ f(a) & \\
2 & \equiv \textsc{non-rec}(u, f, \{u\}\ a) & (R2) \\
4 & \equiv \textsc{non-rec}(u, f, \textsc{non-rec}(u, a, ())) & (R2) \\
5 & \quad \textsc{non-rec}(u, a, ()) & \\
6 & \quad \equiv \textbf{if } \mathbf{0} & (R18) \\
  & \qquad \textbf{then } \textsc{non-rec}(f(b) := 1, a, ()) & \\
  & \qquad \textbf{else } \textsc{non-rec}(f(a) := 0, a, ()) & (R13) \\
7 & \quad \equiv a & (R12) \\
8 & \equiv \textsc{non-rec}(u, f, a) & (4, 7)
\end{array}
$$

Example

$u = f(a) := 0 \,||\, f(b) := 1$

$$
\begin{array}{lll}
1 & \{u\}\ f(a) & \\
2 & \equiv \textsc{non-rec}(u, f, \{u\}\ a) & (R2) \\
4 & \equiv \textsc{non-rec}(u, f, \textsc{non-rec}(u, a, ())) & (R2) \\
5 & \quad \textsc{non-rec}(u, a, ()) & \\
6 & \quad \equiv \textbf{if } \mathbf{0} & (R18) \\
 & \qquad \textbf{then } \textsc{non-rec}(f(b) := 1, a, ()) & \\
 & \qquad \textbf{else } \textsc{non-rec}(f(a) := 0, a, ()) & (R13) \\
7 & \quad \equiv a & (R12) \\
8 & \equiv \textsc{non-rec}(u, f, a) & (4, 7) \\
9 & \equiv \textbf{if } \textsc{in-dom}(f, a, f(b) := 1) & \\
 & \quad \textbf{then } \textsc{non-rec}(f(b) := 1, f, a) & \\
 & \quad \textbf{else } \textsc{non-rec}(f(a) := 0, f, a) & (R13)
\end{array}
$$

Example

$u = f(a) := 0 \,||\, f(b) := 1$

$$
\begin{array}{lll}
1 & \{u\}\; f(a) & \\
2 & \equiv \textsc{non-rec}(u, f, \{u\}\; a) & (R2) \\
4 & \equiv \textsc{non-rec}(u, f, \textsc{non-rec}(u, a, ())) & (R2) \\
5 & \quad \textsc{non-rec}(u, a, ()) & \\
6 & \quad \equiv \textbf{if } \mathbf{0} & (R18) \\
 & \qquad \textbf{then } \textsc{non-rec}(f(b) := 1, a, ()) & \\
 & \qquad \textbf{else } \textsc{non-rec}(f(a) := 0, a, ()) & (R13) \\
7 & \quad \equiv a & (R12) \\
8 & \equiv \textsc{non-rec}(u, f, a) & (4, 7) \\
9 & \equiv \textbf{if } a \doteq b & (R17) \\
 & \quad \textbf{then } \textsc{non-rec}(f(b) := 1, f, a) & \\
 & \quad \textbf{else } \textsc{non-rec}(f(a) := 0, f, a) & (R13)
\end{array}
$$

Example

$u = f(a) := 0 \,||\, f(b) := 1$

1	$\{u\}\, f(a)$	
2	$\equiv \textsc{non-rec}(u, f, \{u\}\, a)$	$(R2)$
4	$\equiv \textsc{non-rec}(u, f, \textsc{non-rec}(u, a, ()))$	$(R2)$
5	$\quad \textsc{non-rec}(u, a, ())$	
6	$\quad \equiv \textbf{if } \mathbf{0}$	$(R18)$
	$\qquad \textbf{then } \textsc{non-rec}(f(b) := 1, a, ())$	
	$\qquad \textbf{else } \textsc{non-rec}(f(a) := 0, a, ())$	$(R13)$
7	$\quad \equiv a$	$(R12)$
8	$\equiv \textsc{non-rec}(u, f, a)$	$(4, 7)$
9	$\equiv \textbf{if } a \doteq b$	$(R17)$
	$\quad \textbf{then if } a \doteq b \textbf{ then } 1 \textbf{ else } f(a)$	$(R13)$
	$\quad \textbf{else } \textsc{non-rec}(f(a) := 0, f, a)$	$(R13)$

Example

$u = f(a) := 0 \,||\, f(b) := 1$

$$
\begin{array}{lll}
1 & \{u\}\ f(a) & \\
2 & \equiv \textsc{non-rec}(u, f, \{u\}\ a) & (R2) \\
4 & \equiv \textsc{non-rec}(u, f, \textsc{non-rec}(u, a, ())) & (R2) \\
5 & \quad \textsc{non-rec}(u, a, ()) & \\
6 & \quad \equiv \textbf{if } \mathbf{0} & (R18) \\
 & \qquad \textbf{then } \textsc{non-rec}(f(b) := 1, a, ()) & \\
 & \qquad \textbf{else } \textsc{non-rec}(f(a) := 0, a, ()) & (R13) \\
7 & \quad \equiv a & (R12) \\
8 & \equiv \textsc{non-rec}(u, f, a) & (4,7) \\
9 & \equiv \textbf{if } a \doteq b & (R17) \\
 & \quad \textbf{then if } a \doteq b \textbf{ then } 1 \textbf{ else } f(a) & (R13) \\
 & \quad \textbf{else } 0 & (R13)
\end{array}
$$

Example

$u = f(a) := 0 \,||\, f(b) := 1$

$$
\begin{array}{rll}
1 & \{u\}\, f(a) & \\
2 & \equiv \textsc{non-rec}(u, f, \{u\}\, a) & (R2) \\
4 & \equiv \textsc{non-rec}(u, f, \textsc{non-rec}(u, a, ())) & (R2) \\
5 & \quad \textsc{non-rec}(u, a, ()) & \\
6 & \quad \equiv \textbf{if } \mathbf{0} & (R18) \\
  & \qquad \textbf{then } \textsc{non-rec}(f(b) := 1, a, ()) & \\
  & \qquad \textbf{else } \textsc{non-rec}(f(a) := 0, a, ()) & (R13) \\
7 & \quad \equiv a & (R12) \\
8 & \equiv \textsc{non-rec}(u, f, a) & (4, 7) \\
9 & \equiv \textbf{if } a \doteq b & (R17) \\
  & \quad \textbf{then if } a \doteq b \textbf{ then } 1 \textbf{ else } f(a) & (R13) \\
  & \quad \textbf{else } 0 & (R13) \\
10 & \textbf{if } a \doteq b \textbf{ then } 1 \textbf{ else } 0 &
\end{array}
$$

Another Example

$u = \texttt{for}\ x\ \{u_0\},\ u_0 = \texttt{if}\ \ 0 \leq x \wedge x \leq 1\ \{g(f(x)) := x\}$

Another Example

$u = \texttt{for}\ x\ \{u_0\},\ u_0 = \texttt{if}\ \ 0 \leq x \wedge x \leq 1\ \{g(f(x)) := x\}$

$$1 \quad \{u\}\, g(0)$$

Another Example

$$u = \texttt{for}\ x\ \{u_0\},\ u_0 = \texttt{if}\ \ 0 \leq x \wedge x \leq 1\ \{g(f(x)) := x\}$$

$$\begin{array}{ll} 1 & \{u\}\ g(0) \\ 2 & \equiv \textsc{non-rec}(u, g, \{u\}\ 0) \quad (R2) \end{array}$$

Another Example

$u = \texttt{for}\ x\ \{u_0\},\ u_0 = \texttt{if}\ \ 0 \leq x \wedge x \leq 1\ \{g(f(x)) := x\}$

$$
\begin{array}{lll}
1 & \{u\}\ g(0) & \\
2 & \equiv \textsc{non-rec}(u, g, \{u\}\ 0) & (R2) \\
3 & \equiv \textsc{non-rec}(u, g, 0) & \mathit{short}
\end{array}
$$

Another Example

$u = \texttt{for}\ x\ \{u_0\},\ u_0 = \texttt{if}\ \ 0 \leq x \wedge x \leq 1\ \{g(f(x)) := x\}$
$r = min\ x.\ \textsc{in-dom}(g, 0, u_0)$

1	$\{u\}\ g(0)$	
2	$\equiv \textsc{non-rec}(u, g, \{u\}\ 0)$	$(R2)$
3	$\equiv \textsc{non-rec}(u, g, 0)$	*short*
4	$\equiv \textsc{non-rec}(\{x/r\}u_0, g, 0)$	$(R15)$

Another Example

$u = \texttt{for}\ x\ \{u_0\},\ u_0 = \texttt{if}\ \ 0 \leq x \wedge x \leq 1\ \{g(f(x)) := x\}$
$r = min\ x.\ 0 \leq x \wedge x \leq 1 \wedge f(x) \doteq 0 \quad (R20)(R17)$

1	$\{u\}\ g(0)$	
2	$\equiv \textsc{non-rec}(u, g, \{u\}\ 0)$	$(R2)$
3	$\equiv \textsc{non-rec}(u, g, 0)$	*short*
4	$\equiv \textsc{non-rec}(\{x/r\}u_0, g, 0)$	$(R15)$
5	$\equiv$ **if** $0 \leq r \wedge r \leq 1$	
	then $\textsc{non-rec}(g(f(r)) := r, g, 0)$	
	else $g(0)$	$(R14)$

Another Example

$u = \texttt{for}\ x\ \{u_0\},\ u_0 = \texttt{if}\ \ 0 \leq x \wedge x \leq 1\ \{g(f(x)) := x\}$
$r = min\ x.\ 0 \leq x \wedge x \leq 1 \wedge f(x) \doteq 0 \quad (R20)(R17)$

1	$\{u\}\ g(0)$	
2	$\equiv \textsc{non-rec}(u, g, \{u\}\ 0)$	$(R2)$
3	$\equiv \textsc{non-rec}(u, g, 0)$	*short*
4	$\equiv \textsc{non-rec}(\{x/r\}u_0, g, 0)$	$(R15)$
5	$\equiv$ **if 1**	Def.of r
	then $\textsc{non-rec}(g(f(r)) := r, g, 0)$	
	else $g(0)$	$(R14)$

Another Example

$u = \texttt{for}\ x\ \{u_0\},\ u_0 = \texttt{if}\ \ 0 \le x \wedge x \le 1\ \{g(f(x)) := x\}$
$r = min\ x.\ 0 \le x \wedge x \le 1 \wedge f(x) \doteq 0 \quad (R20)(R17)$

1	$\{u\}\ g(0)$	
2	$\equiv \textsc{non-rec}(u, g, \{u\}\ 0)$	$(R2)$
3	$\equiv \textsc{non-rec}(u, g, 0)$	*short*
4	$\equiv \textsc{non-rec}(\{x/r\}u_0, g, 0)$	$(R15)$
5	$\equiv \textbf{if 1}$	Def.of r
	$\quad \textbf{then}\ \textsc{non-rec}(g(f(r)) := r, g, 0)$	
	$\quad \textbf{else}\ g(0)$	$(R14)$
6	$\equiv \textbf{if}\ f(r) \doteq 0\ \textbf{then}\ r\ \textbf{else}\ g(0)$	$(R11)$

Another Example

$u = \texttt{for}\ x\ \{u_0\},\ u_0 = \texttt{if}\ \ 0 \leq x \wedge x \leq 1\ \{g(f(x)) := x\}$
$r = min\ x.\ 0 \leq x \wedge x \leq 1 \wedge f(x) \doteq 0 \quad (R20)(R17)$

1	$\{u\}\ g(0)$	
2	$\equiv \textsc{non-rec}(u, g, \{u\}\ 0)$	$(R2)$
3	$\equiv \textsc{non-rec}(u, g, 0)$	*short*
4	$\equiv \textsc{non-rec}(\{x/r\}u_0, g, 0)$	$(R15)$
5	$\equiv$ **if 1**	Def.of r
	then $\textsc{non-rec}(g(f(r)) := r, g, 0)$	
	else $g(0)$	$(R14)$
6	$\equiv$ **if** $f(r) \doteq 0$ **then** r **else** $g(0)$	$(R11)$
7	**if** $f(0) \doteq 0$ **then** 0	
	else if $f(1) \doteq 0$ **then** 1 **else** $g(0)$	

Soundness of Rewrite Rules

For α_i terms or updates, we define

$$\alpha_1 \equiv \alpha_2$$

to hold true if for all $\mathcal{B}$ and β:

$$val_{\mathcal{B}}(\beta, \alpha_1) = val_{\mathcal{B}}(\beta, \alpha_2)$$

In case $\alpha_1 \equiv \alpha_2$ we say α_1 and α_2 are equivalent.

It can be proved that for all rewrite rules $\alpha_1 \rightarrow \alpha_2$ we get

$$\alpha_1 \equiv \alpha_2$$

Note $\{f(a) := 1\} \not\equiv \{f(a) := 1; f(b) := f(b)\}$

A Normal Form for Updates

Normalisation Theorem

For every update u there is an equivalent update of the form

$$\begin{array}{l} \quad \texttt{for}\ x_{1,1}\ \{\texttt{for}\ x_{1,2}\ \{\texttt{for}\ \ldots\ \{\texttt{if}\ \ \phi_1\ \{t_1 := s_1\}\}\}\} \\ \|\ \ldots \\ \|\ \texttt{for}\ x_{k,1}\ \{\texttt{for}\ x_{k,2}\ \{\texttt{for}\ \ldots\ \{\texttt{if}\ \ \phi_k\ \{t_k := s_k\}\}\}\} \end{array}$$

Laws for Commuting and Distributing Updates

For α a term, a formula, or an update:

$$
\begin{array}{rcll}
\{u_1\}\{u_2\}\alpha & \equiv & \{u_1\,;u_2\}\alpha & (R51) \\
u_1 \,||\, (u_2 \,||\, u_3) & \equiv & (u_1 \,||\, u_2) \,||\, u_3 & (R52) \\
u_1; (u_2; u_3) & \equiv & (u_1; u_2); u_3) & (R53) \\
u_1 \,||\, u_2 & \equiv & \mathrm{REJECT}(u_1, u_2) \,||\, u_2 & (R54) \\
u_1 \,||\, u_2 & \equiv & u_2 \,||\, \mathrm{REJECT}(u_1, u_2) & (R55) \\
u & \equiv & \mathtt{if}\ \ \phi\ \{u\} & (R56) \\
u_1 & \equiv & u_1 \,||\, \mathrm{REJECT}(u_1, u_2) & (R57)
\end{array}
$$

where

$$u_1 \equiv u_2 \text{ iff } \text{ for all } \mathcal{A} \text{ and } \beta : val_{\mathcal{A}}(\beta, u_1) = val_{\mathcal{A}}(\beta, u_2)$$

Laws for Commuting and Distributing Updates

Continuation I

$$\begin{array}{rcll}
\texttt{if}\ \ \phi\ \{u_1 \,||\, u_2\} & \equiv & \texttt{if}\ \ \phi\ \{u_1\} \,||\, \texttt{if}\ \ \phi\ \{u_2\} & \\
\texttt{if}\ \ \phi_1\ \{\texttt{if}\ \ \phi_2\ \{u\}\} & \equiv & \texttt{if}\ \ \phi_1 \wedge \phi_2\ \{u\} & \\
\texttt{for}\ x\ \{\texttt{if}\ \ \phi\ \{u\}\} & \equiv & \texttt{if}\ \ \phi\ \{\texttt{for}\ x\ \{u\}\} & (x \notin fv(\phi)) \\
\texttt{for}\ x\ \{\texttt{if}\ \ \phi\ \{u\}\} & \equiv & \texttt{if}\ \ \exists x \phi\ \{u\} & (x \notin fv(u)) \\
\texttt{for}\ x\ \{u_1 \,||\, u_2\} & \equiv & \texttt{for}\ x\ \{u_1\} \,||\, u_2 & (x \notin fv(u_2))
\end{array}$$

$u = \texttt{for}\ z\ \{\texttt{if}\ \ z < x\ \{\{x := z\}u_1\}\}$ with $z \neq x, z \notin fv(u_1)$

$$\begin{array}{rcl}
\texttt{for}\ x\ \{u_1\} & \equiv & \texttt{for}\ x\ \{\mathrm{REJECT}(u_1, u)\} \\
\texttt{for}\ x\ \{u_1 \,||\, u_2\} & \equiv & \texttt{for}\ x\ \{u_1\} \,||\, \texttt{for}\ x\ \{\mathrm{REJECT}(u_2, u)\}
\end{array}$$

$u = \texttt{for}\ z\ \{\texttt{if}\ \ z < x\ \{\{x := z\}\texttt{for}\ y\ \{u_1\}\}\}$
with $card(\{x, y, z\}) = 3, z \notin fv(u_1)$

$$\texttt{for}\ x\ \{\texttt{for}\ y\ \{u_1\}\} \equiv \texttt{for}\ y\ \{\texttt{for}\ x\ \{\mathrm{REJECT}(u_1, u)\}\}$$

Rewrite Rules For $\text{REJECT}(u_1, u_2)$ and ;

$$
\begin{array}{rcll}
\text{REJECT}(\textbf{skip}, u) & \rightarrow & \textbf{skip} & (R22) \\
\text{REJECT}(f(\bar{s}) := t, u) & \rightarrow & \textbf{if } \neg\text{IN-DOM}(f, \bar{s}, u)\ \{f(\bar{s}) := t\} & (R23) \\
\text{REJECT}(u_1 \,||\, u_2, u) & \rightarrow & \text{REJECT}(u_1, u) \,||\, \text{REJECT}(u_2, u) & (R24) \\
\text{REJECT}(\texttt{if}\ \ \phi\ \{u_1\}, u) & \rightarrow & \texttt{if}\ \ \phi\ \{\text{REJECT}(u_1, u)\} & (R25) \\
\text{REJECT}(\texttt{for}\ x\ \{u_1\}, u) & \rightarrow & \texttt{for}\ x\ \{\text{REJECT}(u_1, u)\} & x \notin fv(u) \\
 & & & (R26)
\end{array}
$$

$$
\begin{array}{rclll}
u_1; u_2 & \rightarrow & u_1 \,||\, \{u_1\}\ u_2 & & (R45) \\
\{u\}\ \textbf{skip} & \rightarrow & \textbf{skip} & & (R46) \\
\{u\}\ f(\bar{s}) := t & \rightarrow & f(\{u\}\ \bar{s}) := \{u\}\ t & & (R47) \\
\{u\}\ u_1 \,||\, u_2 & \rightarrow & \{u\}\ u_1 \,||\, \{u\}\ u_2 & & (R48) \\
\{u\}\ \texttt{if}\ \ \phi\ \{u_1\} & \rightarrow & \texttt{if}\ \ \{u\}\ \phi\ \{\{u\}\ u_1\} & & (R49) \\
\{u\}\ \texttt{for}\ x\ \{u_1\} & \rightarrow & \texttt{for}\ x\ \{\{u\}\ u_1\} & x \notin fv(u) & (R50)
\end{array}
$$

Example

$u = \texttt{for}\ x\ \{u_0\}, u_0 = f(x) := 1 \,||\, f(b) := 2$
$u_1 = \texttt{for}\ z\ \{\texttt{if}\ \ z < x\ \{f(z) := 1\}\}$

$$
\begin{array}{lll}
1 & u; \texttt{if}\ \ a \neq b\ \{f(a) := 0\} & \\
2 & \equiv u \,||\, \{u\}\ \texttt{if}\ \ a \neq b\ \{f(a) := 0\} & (R45) \\
3 & \equiv u \,||\, \texttt{if}\ \ a \neq b\ \{f(\{u\}\ a) := 0\} & (R47, 49) \\
4 & \equiv u \,||\, \texttt{if}\ \ a \neq b\ \{f(a) := 0\} & \textit{short} \\
5 & \equiv \quad \texttt{for}\ x\ \{f(x) := 1\} & \\
  & \quad || \texttt{for}\ x\ \{\text{REJECT}(f(b) := 2, u_1)\} & (R64) \\
  & \quad || \texttt{if}\ \ a \neq b\ \{f(a) := 0\} & \\
6 & \equiv \quad \texttt{for}\ x\ \{f(x) := 1\} & \\
  & \quad || \texttt{for}\ x\ \{\texttt{if}\ \ \neg\text{IN-DOM}(f, b, u_1)\ \{f(b) := 2\}\} & (R23) \\
  & \quad || \texttt{if}\ \ a \neq b\ \{f(a) := 0\} & \\
7 & \equiv \quad \texttt{for}\ x\ \{f(x) := 1\} & \\
  & \quad || \texttt{for}\ x\ \{\texttt{if}\ \ \forall z(z < x \rightarrow z \neq b)\ \{f(b) := 2\}\} & (R21) \\
  & \quad || \texttt{if}\ \ a \neq b\ \{f(a) := 0\} &
\end{array}
$$

References

- The Abstract State Machine (ASM) specification language uses a very similiar concept of updates.
  R. Stärk, S. Nachan, *A logic for for abstract state machines*
  J.Universal Computer Science, 7 (2001), 981–1006.
- Generalised Substitutions in the **B** language have a character similar to updates.
- Guarded command languages share some similarities with updates but als cover loop or other interation constructs.
- The rewrite calculus presented here is taken from: Ph.Rümmer, Licentiate Thesis, Chalmers, 2006

Dynamic Logic
Lecture 6: Dynamic Logic for Javacard

Prof. P.H. Schmitt

Institut für Theoretische Informatik
Fakultät für Informatik
Universität Karlsruhe (TH)

Logic Summer School, Canberra, February, 2009

Drawbacks of the Assignment Axiom

$\langle x := t \rangle F \leftrightarrow F[x/t]$

- leads easily to big formulas by performing substitutions
- only applicable for first-order formula F, thus preventing symbolic execution of programs
- not pratical in realistic context with array variable, aliasing, exception handling etc.

Differentiating Variables

Logical Variables

- can be quantified
- never occur in programs

Program Variables

- can not be quantified
- may occur in programs and formulas

$\forall x \langle x := x + x \rangle even(x)$ ~~$\forall x \langle x := x + x \rangle even(x)$~~ no longer possible.

Instead:

$\forall x (\{a := x\} \langle a := a + a \rangle even(a))$

The construct $\{a := x\}$ is called an update and establishes the link between pure formulas and programs.

Use of Updates

Assignment Rule

Previous Version w/o Updates

$$\frac{\Gamma(z/x), x \doteq t(z/x) \quad \Rightarrow \quad F, \Delta(z/x)}{\Gamma \quad \Rightarrow \quad \langle x = t \rangle F, \Delta}$$

Version with Updates

$$\frac{\Gamma \quad \Rightarrow \quad \{x := t\} F, \Delta}{\Gamma \quad \Rightarrow \quad \langle x = t \rangle F, \Delta}$$

Use of Updates

Example Proof with Updates

$$\Rightarrow \langle f(a) = 1; f(a) = 2; f(b) = 3; \rangle p(f(a))$$

$$\Rightarrow \{f(a) := 1\} \langle f(a) = 2; f(b) = 3; \rangle p(f(a))$$

$$\Rightarrow \{f(a) := 1; f(a) := 2\} \langle f(b) = 3; \rangle p(f(a))$$

$$\Rightarrow \{f(a) := 2\} \langle f(b) = 3; \rangle p(f(a))$$

$$\Rightarrow \{f(a) := 2; f(b) := 3\} \langle \rangle p(f(a))$$

$$\Rightarrow p\big(if(a \doteq b)then(3)else(2)\big)$$

Use of Updates

Roadmap of a Proof with Updates

1. "collect effects" of a statement in an update
2. repeat 1 till the program is completely "executed"
3. syntactically apply the collected update to the postcondition
4. prove the resulting first order formula

Benefits of Updates

- means to describe state transitions independent of a prog. language
- advantageous for handling "aliasing" effects

Dynamic Logic For Realistic Program Verification

What is Missing?

- Treatment of arrays
- Dynamic Logic for object-oriented programs
  We will present $JavaDL$, a Dynamic Logic for sequential Java.
- Integration into software development process
- Higher levels of specification

This is easy.

- the first-order part is the typed first-order logic from lecture 4.
- as programs in Π_{JavaDL} all parsable sequential Java programs are allowed.

$JavaDL$ Semantics

First Version

Previous definition

For every first-order structure $\mathcal{M} = (M, val_{\mathcal{M}})$
$\mathcal{K}_{\mathcal{M}} = (S, \rho, \models)$
is the Kripke structure with computation domain $\mathcal{M}$

$S = Var \to M$

$\rho : \Pi \to S \times S$
the accessibility relations

$\models \ \subseteq S \times Fml_{\Sigma}$
the evaluation relation

$JavaDL$ modifications

For every typed first-order structure $\mathcal{M} = (\mathcal{D}, \delta, \mathcal{I})$
$\mathcal{K}_{\mathcal{M}} = (S, \rho, \models)$.
is the Kripke structure with Kripke seed $\mathcal{M}$

S = the set of all typed structures extending $\mathcal{M}$

$\rho : \Pi \to S \times S$
the accessibility relations

$\models \ \subseteq S \times Fml_{\Sigma}$
the evaluation relation

Example

When reasoning about the program

```
public class Point{
 int x,y;
 public void move(int dx, int dy){
     x = x + dx;
     y = y + dy;
   }
 public boolean equals(Object other){
   if (other != null && other instanceof Point)
      {Point p = (Point) other;
       return (x == p.x && y == p.y);
       }
   else {return false;}
 }}
```

the Kripke seed would would be the integers $\mathbb{Z}$ with $+$
the set S of states would be the set of all two-sorted structures

- with sort int always interpreted as $(\mathbb{Z}, +)$
- sort $Point$ interpeted as an arbitrary set
- and arbitrary interpretations of the unary functions x and y of sort $Point \to int$.

$JavaDL$ Semantics

Updates

Previous definition	$JavaDL$ modifications
For every variable x and term t	For every function symbol f and terms t, s
$x := t$	$t.f := s$

From the program logic point of view this is the fundamental difference between imperative and object-oriented programming languages.

JavaDL Semantics

Creating new objects

The Kripke seed $\mathcal{M}$ contains for every type A a universe $\mathcal{D}^A$ of all potential objects of type A.

$\mathcal{D}^A$ is the same for all states in $\mathcal{K}_{\mathcal{M}}$
fixed domain semantics

There is an implicit Boolean field $iscreated$ such that for any state s the set of existing objects in s is

$$\{a \in \mathcal{D}^A \mid s \models a.iscreated = \mathbf{1}\}.$$

Creating an object amounts to updating $a.iscreated = \mathbf{0}$ to $a.iscreated = \mathbf{1}$.

Technically, are reference to the next object o be created is necessary.

$JavaDL$ Semantics

Assignments with Side Effects

$$\frac{\Gamma \Rightarrow \langle y = y + 1; x = y; \alpha\rangle F, \Delta}{\Gamma \Rightarrow \langle x = {+}{+}y; \alpha\rangle F, \Delta}$$

$$\frac{\Gamma \Rightarrow \langle z = y; y = y + 1; x = z; \alpha\rangle F, \Delta \qquad z \text{ a new variable}}{\Gamma \Rightarrow \langle x = y{+}{+}; \alpha\rangle F, \Delta}$$

An Incorrect Post-Increment Rule

$$\frac{\Gamma \Rightarrow \langle x = y; y = y + 1; \alpha\rangle F, \Delta}{\Gamma \Rightarrow \langle x = y{+}{+}; \alpha\rangle F, \Delta}$$

The problem occurs when $x \equiv y$.

$$\frac{x = 5 \Rightarrow \langle x = x; x = x + 1\rangle x = 6}{x = 5 \Rightarrow \langle x = x{+}{+};\rangle x = 6}$$

According to the Java semantics the conclusion is false.
Yet, its premisse is true, showing the unsoundness of the rule.

A While Rule

$$\frac{\Gamma \Rightarrow I, \Delta \quad I, F_0 \Rightarrow [\pi] I \quad I, \neg F_0 \Rightarrow F}{\Gamma \Rightarrow [\mathrm{while}(F_0)\{\pi\}] F, \Delta}$$

I is called a loop invariant.

The resulting proof obligations are called:

Invariant initially valid
Preservation of invariant
Use invariant

An Incorrect While Rule

$$\frac{\Gamma \Rightarrow I, \Delta \qquad I, F_0 \Rightarrow [\pi] I, \Delta \qquad I, \neg F_0 \Rightarrow F, \Delta}{\Gamma \Rightarrow [\mathrm{while}(F_0)\{\pi\}] F, \Delta}$$

Instantiating:

$\Gamma = empty$, $I = true$, $F \equiv F_0 \equiv x \neq 1$, $\Delta = \{x = 1\}$

We obtain:

$$\frac{\emptyset \Rightarrow true \qquad x \neq 1 \Rightarrow [x = 1]\, true \qquad x = 1 \Rightarrow x \neq 1, x = 1}{\emptyset \Rightarrow [\mathrm{while}\ \ (x \neq 1)\{x = 1\}] x \neq 1, x = 1}$$

A While Rule with Termination

$$\frac{\Gamma \Rightarrow t \geq 0, T, \Delta \quad t \geq 0, I, F_0 \Rightarrow \langle\pi\rangle t > \backslash old(t), I \quad t \geq 0, I, \neg F_0 \Rightarrow F}{\Gamma \Rightarrow \langle \mathrm{while}(F_0)\{\pi\}F\rangle, \Delta}$$

I is called a loop invariant.

t a term of sort int is called the loop variant.

Unfolding `while` without Labels

$$\frac{\Gamma \Rightarrow \langle if(c)\{p; \quad while(c)\{p\}\}\ \omega\rangle\ \phi}{\Gamma \Rightarrow \langle while(c)\{p\}\ \omega\rangle\ \phi}$$

Unfolding `while` Loops

$$\frac{\Gamma \Rightarrow \langle \pi\ if(c) l^1 : \{ l^2 : \{ p' \};\ \ l_1 : \ldots l_n : while(c)\{p\}\}\ \omega \rangle\ \phi}{\Gamma \Rightarrow \langle \pi\ l_1 : \ldots l_n : while(c)\{p\}\ \omega \rangle\ \phi}$$

with

- l^1, l^2 are new labels
- p' is the result of simultaneously replacing:
  - $break\ l_i$ by $break\ l^1$
  - $break$ not nested by $break\ l^1$
  - $continue\ l_i$ by $break\ l^2$
  - $continue$ not nested by $break\ l^2$

An Example with Loop

```
public class Break{
    int i;
    /*@   public normal_behavior
      @   requires i<=10;
      @   assignable i;
      @   ensures i==10;
      @*/
 public void loop(){
     /*@ loop_invariant
       @  i<=10;
       @ assignable i;
       @ decreases 10-i;
       @*/
   while (true) {
       if (i==10) break;
        i++;
}}}
```

Is This Contract Satisfied?

```
public class SimpleWhile0 {
 int a,b,r;
 /*@ public normal_behavior
   @ requires a >= 0 && b >= 0;
   @ ensures \result == \old(a)*\old(b);
   @ ensures a == -1;
   @ diverges false;
   @*/
 int simplemult0(){ int r = 0; int a0ld = a;
/*@loop_invariant
  @ 0 <= a && r == (a0ld-a)*b;
  @ decreases a;
  @*/
      while (0<a--) {r = r + b;}
      return r;
    }}
```

Improved Loop Specification

```
public class SimpleWhile {
 int a,b,r;
  as before
 int simplemult(){
   int r = 0; int aOld = a;
/*@loop_invariant
  @ 0 <= a && r == (aOld-a)*b;
  @ decreases a;
  @ assignable a, r;
  @*/
      while (0<a--) {r = r + b;}
      return r;
    }}
```

Specifications Involving Integers

The following JML specification for the integer square root method can be found in the frist version of the JML manual by Gary T. Leavens, Albert L. Baker, and Clyde Ruby from 2003:

```
/*@ requires y >= 0;
  @ ensures
  @  \result * \result <= y &&
  @ y < (abs(\result)+1) * (abs(\result)+1);
  @ */
  public static int isqrt(int y)
```

Patrice Chalin pointed out the following flaw: For $y = 1$ and $\backslash result = 1073741821 = \frac{1}{2}(max\_int - 5)$ the above postcondition is true, but 1073741821 is not square root of 1.

Specifications Involving Integers

What is the Problem?

```
/*@ requires y >= 0;
  @ ensures
  @  \result * \result <= y &&
  @ y < (abs(\result)+1) * (abs(\result)+1);
  @ */
  public static int isqrt(int y)
```

The above postcondition is satisfied by $\backslash result = 1073741821$ is not square root of $y = 1$.

The problem arises since JML uses the JAVA semantics of integers which yields

$$\begin{aligned} 1073741821 * 1073741821 &= -2147483639 \\ 1073741822 * 1073741822 &= 4 \end{aligned}$$

DEMO

Programs Used in Demo

```
public class ISQRT{
 static int y;
 /*@   public normal_behavior
   @   requires y>=0;
   @   ensures \result*\result <= y &&
   @   \result>=0 &&
   @   (\result+1)*(\result+1) > y;
   @*/
 static public int isqrt(int y){
 int x = 0;
  /*@ loop_invariant
    @  x*x <= y && x>=0;
    @ assignable x;
    @ decreases  y-x;
    @*/
while((x+1)*(x+1)<=y && 0<=(x+1)*(x+1)){x=x+1};
   return x;}}
```

Programs Used in Demo

```
public class ISQRT2{
 static int y;
 /*@   public normal_behavior
   @   requires y>=0;
   @   ensures \result*\result <= y &&
   @    \result>=0 &&
   @    (\result+1)*(\result+1) > y;
   @*/
 static public int isqrt(int y){
   int x = 0;
 /*@ loop_invariant
   @  x*x <= y &&  x>=0;
   @ assignable x;
   @ decreases  y-x;
   @*/
 while ((2*x + 1)<=(y - x*x)){x=x+1;};
   return x;}}
```

Programs Used in Demo

```
\javaSource "...";
\programVariables {
int y,_y.result,java.lang.Exception exc;
}

\problem {
   inReachableState & inInt(y) & y >= (jint)(0)
-> {_y:=y}
  \<{
    exc=null;
    try {result=ISQRT3.isqrt(_y)@ISQRT3;}
    catch (java.lang.Throwable e) {exc=e;}
       }\> ( (  result*result) <= y
            &   result >= 0
            &   (result+1)*(result+1) >  y
            &   exc = null)
}
```

For Further Information Consult

THE END