Formale Systeme II: Theorie

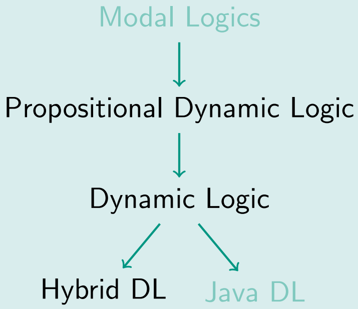
Dynamic Logic:
Uninterpreted and Interpreted First Order DL

SS 2022

Prof. Dr. Bernhard Beckert · Dr. Mattias Ulbrich
Slides partially by Prof. Dr. Peter H. Schmitt

Roadmap

Overview – a family of logics



Motivation

First Order Dynamic Logic

Atomic programs are refined to **assignments**.

Motivation

Karlsruhe Institute of Technology

First Order Dynamic Logic

Atomic programs are refined to **assignments**.

Example Formula

$$x_0 = x \land y_0 = y \to [x := x + y; y := x - y; x := x - y]\,\varphi$$

First Order Dynamic Logic

Inherit from FOL:

- Terms over function symbols and variables
- Predicate symbols
- Quantification over variables

Inherit from PDL

- Modalities
- Composite program constructors

Refine PDL

Unspecified atomic programs replaced by assignments $\mathit{var} := \mathit{term}$

Syntax

Syntactical material

$\Sigma = (F, P, \alpha)$ ... signature

F ... function symbols

P ... predicate symbols

$\alpha : F \cup P \to \mathbb{N}$ ... arity function

Var ... set of variables

- No atomic programs like in PDL
- Same as for FOL

Syntax

As abstract grammar:

$$
\begin{aligned}
term \quad &::= \quad var \mid f(term_1, \ldots, term_{\alpha(f)}) \\
\\
fml \quad &::= \quad true \mid false \mid p(term_1, \ldots, term_{\alpha(p)}) \mid term_1 = term_2 \\
&\mid \quad \neg fml \mid fml_1 \wedge fml_2 \mid fml_1 \vee fml_2 \mid fml_1 \rightarrow fml_2 \\
&\mid \quad \exists var.\ fml \mid \forall var.\ fml \\
&\mid \quad [prog]fml \mid \langle prog \rangle fml \\
\\
prog \quad &::= \quad var := term \\
&\mid \quad var := * \\
&\mid \quad prog_1 \,;\, prog_2 \mid prog_1 \cup prog_2 \mid prog^*
\end{aligned}
$$

for $var \in Var, f \in F, p \in P$

Semantics – Kripke Structures

First Order Structure (D, I)

D ... set of objects (domain) I ... Interpretation

$I(f) : D^{\alpha(f)} \rightarrow D$ for function symbol $f \in F$

$I(p) \subseteq D^{\alpha(p)}$ for predicate symbol $p \in P$

Semantics – Kripke Structures

First Order Structure (D, I)

D ... set of objects (domain) I ... Interpretation

$I(f) : D^{\alpha(f)} \to D$ for function symbol $f \in F$

$I(p) \subseteq D^{\alpha(p)}$ for predicate symbol $p \in P$

Kripke Frame (S, ρ)

S ... set of states $\rho : \mathtt{prog} \to 2^{S \times S}$ ... accessibility relation

Semantics – Kripke Structures

First Order Structure (D, I)

D ... set of objects (domain) I ... Interpretation

$I(f) : D^{\alpha(f)} \to D$ for function symbol $f \in F$

$I(p) \subseteq D^{\alpha(p)}$ for predicate symbol $p \in P$

Kripke Frame (S, ρ)

S ... set of states $\rho : \mathtt{prog} \to 2^{S \times S}$ ... accessibility relation

FODL: Fixed Kripke Frame $\mathcal{K}_D = (S_D, \rho_D)$

which depends on the domain D

Semantics – Kripke Structures

The set of states $\mathcal{K}_D$ is the set of assignments of elements in the universe D to variables in Var:

$$S_D = Var \to D$$

Semantics – Kripke Structures

The set of states $\mathcal{K}_D$ is the set of assignments of elements in the universe D to variables in Var:

$$S_D = Var \to D$$

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

$$val_{D,I,s}(t)$$

the usual first-order evaluation of t in (D, I);
variables are interpreted via s.

Function Update Notation

Notation: for $s \in S_D$, $x \in Var$, $a \in D$

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

Semantics of Programs

Binary Relation

$\rho : \texttt{prog} \rightarrow S_D \times S_D$ assigns accessiblity to programs

Semantics of Programs

Binary Relation

$\rho : \texttt{prog} \to S_D \times S_D$ assigns accessiblity to programs

$$\rho(x := v) \quad = \quad \{(s, t) \mid t = s[x/val_{D,I,s}(v)]\}$$

Semantics of Programs

Binary Relation

$\rho : \texttt{prog} \to S_D \times S_D$ assigns accessiblity to programs

$$\rho(x := v) \quad = \quad \{(s,t) \mid t = s[x/val_{D,I,s}(v)]\}$$

$$\rho(x := *) \quad = \quad \{(s,t) \mid \text{ex. } a \in D \text{ with } t = s[x/a]\}$$

Semantics of Programs

Binary Relation

$\rho : \texttt{prog} \to S_D \times S_D$ assigns accessiblity to programs

$$\rho(x := v) \quad = \quad \{(s, t) \mid t = s[x/val_{D,I,s}(v)]\}$$

$$\rho(x := *) \quad = \quad \{(s, t) \mid \text{ex. } a \in D \text{ with } t = s[x/a]\}$$

$$\rho(\pi_1 \cup \pi_2) \quad = \quad \rho(\pi_1) \cup \rho(\pi_2)$$

Semantics of Programs

Binary Relation

$\rho : \texttt{prog} \to S_D \times S_D$ assigns accessiblity to programs

$$\rho(x := v) = \{(s,t) \mid t = s[x/val_{D,I,s}(v)]\}$$

$$\rho(x := *) = \{(s,t) \mid \text{ex. } a \in D \text{ with } t = s[x/a]\}$$

$$\rho(\pi_1 \cup \pi_2) = \rho(\pi_1) \cup \rho(\pi_2)$$

$$\rho(\pi_1 \,;\, \pi_2) = \rho(\pi_1) \,;\, \rho(\pi_2)$$

; is forward composition

Semantics of Programs

Binary Relation

$\rho : \texttt{prog} \to S_D \times S_D$ assigns accessiblity to programs

$$\rho(x := v) \quad = \quad \{(s,t) \mid t = s[x/val_{D,I,s}(v)]\}$$

$$\rho(x := *) \quad = \quad \{(s,t) \mid \text{ex. } a \in D \text{ with } t = s[x/a]\}$$

$$\rho(\pi_1 \cup \pi_2) \quad = \quad \rho(\pi_1) \cup \rho(\pi_2)$$

$$\rho(\pi_1 \,;\, \pi_2) \quad = \quad \rho(\pi_1) \,;\, \rho(\pi_2) \qquad \text{; is forward composition}$$

$$= \quad \{(s,t) \mid \text{ex. } u \in S_D \text{ with } (s,u) \in \rho(\pi_1), (u,t) \in \rho(\pi_2)\}$$

Semantics of Programs

Binary Relation

$\rho : \mathtt{prog} \rightarrow S_D \times S_D$ assigns accessiblity to programs

$$
\begin{aligned}
\rho(x := v) &= \{(s,t) \mid t = s[x/val_{D,I,s}(v)]\} \\
\rho(x := *) &= \{(s,t) \mid \text{ex. } a \in D \text{ with } t = s[x/a]\} \\
\rho(\pi_1 \cup \pi_2) &= \rho(\pi_1) \cup \rho(\pi_2) \\
\rho(\pi_1 \,;\, \pi_2) &= \rho(\pi_1) \,;\, \rho(\pi_2) \qquad \text{; is forward composition} \\
&= \{(s,t) \mid \text{ex. } u \in S_D \text{ with } (s,u) \in \rho(\pi_1), (u,t) \in \rho(\pi_2)\} \\
\rho(\pi^*) &= \rho(\pi)^* \qquad \text{* is refl. transitive closure}
\end{aligned}
$$

Semantics of Programs

Binary Relation

$\rho : \texttt{prog} \to S_D \times S_D$ assigns accessiblity to programs

$$\rho(x := v) = \{(s,t) \mid t = s[x/val_{D,I,s}(v)]\}$$

$$\rho(x := *) = \{(s,t) \mid \text{ex. } a \in D \text{ with } t = s[x/a]\}$$

$$\rho(\pi_1 \cup \pi_2) = \rho(\pi_1) \cup \rho(\pi_2)$$

$$\rho(\pi_1 \,;\, \pi_2) = \rho(\pi_1) \,;\, \rho(\pi_2)$$; is forward composition

$$= \{(s,t) \mid \text{ex. } u \in S_D \text{ with } (s,u) \in \rho(\pi_1), (u,t) \in \rho(\pi_2)\}$$

$$\rho(\pi^*) = \rho(\pi)^*$$ * is refl. transitive closure

$$= \{(s_o, s_n) \mid \text{ex. } n \geq 0 \text{ with } (s_i, s_{i+1}) \in \rho(\pi) \text{ f.a. } i < n\}$$

Semantics of Programs

Binary Relation

$\rho : \texttt{prog} \to S_D \times S_D$ assigns accessiblity to programs

$$\rho(x := v) = \{(s,t) \mid t = s[x/val_{D,I,s}(v)]\}$$

$$\rho(x := *) = \{(s,t) \mid \text{ex. } a \in D \text{ with } t = s[x/a]\}$$

$$\rho(\pi_1 \cup \pi_2) = \rho(\pi_1) \cup \rho(\pi_2)$$

$$\rho(\pi_1 \,;\, \pi_2) = \rho(\pi_1) \,;\, \rho(\pi_2)$$

; is forward composition

$$= \{(s,t) \mid \text{ex. } u \in S_D \text{ with } (s,u) \in \rho(\pi_1), (u,t) \in \rho(\pi_2)\}$$

$$\rho(\pi^*) = \rho(\pi)^*$$

* is refl. transitive closure

$$= \{(s_o, s_n) \mid \text{ex. } n \geq 0 \text{ with } (s_i, s_{i+1}) \in \rho(\pi) \text{ f.a. } i < n\}$$

$$\rho(\mathbf{?}\varphi) = \{(s,s) \mid I, s \models \varphi\}$$

Semantics of Formulae

$$I, s \models p(t_1, \ldots, t_n) \quad \text{iff} \quad (val_{I,s}(t_1), \ldots, val_{I,s}(t_n)) \in I(p)$$

Semantics of Formulae

KIT Karlsruhe Institute of Technology

$$I, s \models p(t_1, \ldots, t_n) \quad \text{iff} \quad (val_{I,s}(t_1), \ldots, val_{I,s}(t_n)) \in I(p)$$

$$I, s \models t_1 = t_2 \quad \text{iff} \quad val_{I,s}(t_1) = val_{I,s}(t_2)$$

Semantics of Formulae

$$I, s \models p(t_1, \ldots, t_n) \quad \text{iff} \quad (val_{I,s}(t_1), \ldots, val_{I,s}(t_n)) \in I(p)$$

$$I, s \models t_1 = t_2 \quad \text{iff} \quad val_{I,s}(t_1) = val_{I,s}(t_2)$$

$$I, s \models [\pi]F \quad \text{iff} \quad I, s' \models F \text{ for all } s' \text{ with } (s, s') \in \rho(\pi)$$

Semantics of Formulae

$$I, s \models p(t_1, \ldots, t_n) \quad \text{iff} \quad (val_{I,s}(t_1), \ldots, val_{I,s}(t_n)) \in I(p)$$

$$I, s \models t_1 = t_2 \quad \text{iff} \quad val_{I,s}(t_1) = val_{I,s}(t_2)$$

$$I, s \models [\pi]F \quad \text{iff} \quad I, s' \models F \text{ for all } s' \text{ with } (s, s') \in \rho(\pi)$$

$$I, s \models \langle\pi\rangle F \quad \text{iff} \quad I, s' \models F \text{ for some } s' \text{ with } (s, s') \in \rho(\pi)$$

Semantics of Formulae

$I, s \models p(t_1, \ldots, t_n)$ iff $(val_{I,s}(t_1), \ldots, val_{I,s}(t_n)) \in I(p)$

$I, s \models t_1 = t_2$ iff $val_{I,s}(t_1) = val_{I,s}(t_2)$

$I, s \models [\pi]F$ iff $I, s' \models F$ for all s' with $(s, s') \in \rho(\pi)$

$I, s \models \langle\pi\rangle F$ iff $I, s' \models F$ for some s' with $(s, s') \in \rho(\pi)$

$\models$ is as expected for $\neg, \wedge, \vee, \rightarrow, \forall x, \exists x$.

Semantics of Formulae

Karlsruhe Institute of Technology

$$
\begin{array}{lll}
I, s \models p(t_1, \ldots, t_n) & \text{iff} & (val_{I,s}(t_1), \ldots, val_{I,s}(t_n)) \in I(p) \\
I, s \models t_1 = t_2 & \text{iff} & val_{I,s}(t_1) = val_{I,s}(t_2) \\
I, s \models [\pi]F & \text{iff} & I, s' \models F \text{ for all } s' \text{ with } (s, s') \in \rho(\pi) \\
I, s \models \langle\pi\rangle F & \text{iff} & I, s' \models F \text{ for some } s' \text{ with } (s, s') \in \rho(\pi)
\end{array}
$$

$\models$ is as expected for $\neg, \wedge, \vee, \rightarrow, \forall x, \exists x$.

We write $I \models \varphi$ iff $I, s \models \varphi$ for all $s \in S$.

Basic Observation

$\pi \in \texttt{prog}$ a program
$FV(\pi) = \{x \in Var \mid \text{ex. } t \text{ such that } x := t \text{ or } x := * \text{ occurs in } \pi\}$
$V(\pi) = \{x \in Var \mid x \text{ occurs in } \pi\}$

Basic Observation

$\pi \in \texttt{prog}$ a program
$FV(\pi) = \{x \in Var \mid \text{ex. } t \text{ such that } x := t \text{ or } x := * \text{ occurs in } \pi\}$
$V(\pi) = \{x \in Var \mid x \text{ occurs in } \pi\}$

1. If $(s, s_1) \in \rho(\pi)$ then $s(x) = s_1(x)$ for all $x \notin FV(\pi)$.
   i.e., program π only changes variables in $FV(\pi)$;

Basic Observation

$\pi \in \texttt{prog}$ a program
$FV(\pi) = \{x \in Var \mid \text{ex. } t \text{ such that } x := t \text{ or } x := * \text{ occurs in } \pi\}$
$V(\pi) = \{x \in Var \mid x \text{ occurs in } \pi\}$

1. If $(s, s_1) \in \rho(\pi)$ then $s(x) = s_1(x)$ for all $x \notin FV(\pi)$.
   i.e., program π only changes variables in $FV(\pi)$;

2. If $(s, s_1) \in \rho(\pi)$ then $(s[x/a], s_1[x/a]) \in \rho(\pi)$
   for $a \in D$, $x \notin V(\pi)$.
   i.e., variables outside $V(\pi)$ do not influence the program π;

Basic Observation

$\pi \in \texttt{prog}$ a program
$FV(\pi) = \{x \in Var \mid \text{ex. } t \text{ such that } x := t \text{ or } x := * \text{ occurs in } \pi\}$
$V(\pi) = \{x \in Var \mid x \text{ occurs in } \pi\}$

1. If $(s, s_1) \in \rho(\pi)$ then $s(x) = s_1(x)$ for all $x \notin FV(\pi)$.
   i.e., program π only changes variables in $FV(\pi)$;
2. If $(s, s_1) \in \rho(\pi)$ then $(s[x/a], s_1[x/a]) \in \rho(\pi)$
   for $a \in D$, $x \notin V(\pi)$.
   i.e., variables outside $V(\pi)$ do not influence the program π;
3. more general: If $(s, s_1) \in \rho(\pi)$ and $s' \in S_D$ such that
   $s'(y) = s(y)$ for all $y \in V(\pi)$ then there is s'_1 such that

Basic Observation

$\pi \in \texttt{prog}$ a program
$FV(\pi) = \{x \in Var \mid \text{ex. } t \text{ such that } x := t \text{ or } x := * \text{ occurs in } \pi\}$
$V(\pi) = \{x \in Var \mid x \text{ occurs in } \pi\}$

1. If $(s, s_1) \in \rho(\pi)$ then $s(x) = s_1(x)$ for all $x \notin FV(\pi)$.
   i.e., program π only changes variables in $FV(\pi)$;
2. If $(s, s_1) \in \rho(\pi)$ then $(s[x/a], s_1[x/a]) \in \rho(\pi)$
   for $a \in D$, $x \notin V(\pi)$.
   i.e., variables outside $V(\pi)$ do not influence the program π;
3. more general: If $(s, s_1) \in \rho(\pi)$ and $s' \in S_D$ such that $s'(y) = s(y)$ for all $y \in V(\pi)$ then there is s'_1 such that
   1. $(s', s'_1) \in \rho(\pi)$ and

Basic Observation

$\pi \in \texttt{prog}$ a program
$FV(\pi) = \{x \in Var \mid \text{ex. } t \text{ such that } x := t \text{ or } x := * \text{ occurs in } \pi\}$
$V(\pi) = \{x \in Var \mid x \text{ occurs in } \pi\}$

1. If $(s, s_1) \in \rho(\pi)$ then $s(x) = s_1(x)$ for all $x \notin FV(\pi)$.
   i.e., program π only changes variables in $FV(\pi)$;
2. If $(s, s_1) \in \rho(\pi)$ then $(s[x/a], s_1[x/a]) \in \rho(\pi)$
   for $a \in D$, $x \notin V(\pi)$.
   i.e., variables outside $V(\pi)$ do not influence the program π;
3. more general: If $(s, s_1) \in \rho(\pi)$ and $s' \in S_D$ such that $s'(y) = s(y)$ for all $y \in V(\pi)$ then there is s'_1 such that
   1. $(s', s'_1) \in \rho(\pi)$ and
   2. $s'_1(x) = s'(x)$ for all $x \notin V(\pi)$

Basic Observation

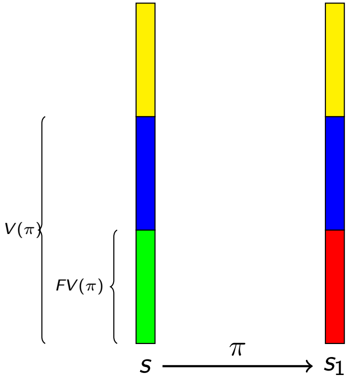
$\pi \in \texttt{prog}$ a program
$FV(\pi) = \{x \in Var \mid \text{ex. } t \text{ such that } x := t \text{ or } x := * \text{ occurs in } \pi\}$
$V(\pi) = \{x \in Var \mid x \text{ occurs in } \pi\}$

1. If $(s, s_1) \in \rho(\pi)$ then $s(x) = s_1(x)$ for all $x \notin FV(\pi)$.
   i.e., program π only changes variables in $FV(\pi)$;
2. If $(s, s_1) \in \rho(\pi)$ then $(s[x/a], s_1[x/a]) \in \rho(\pi)$
   for $a \in D$, $x \notin V(\pi)$.
   i.e., variables outside $V(\pi)$ do not influence the program π;
3. more general: If $(s, s_1) \in \rho(\pi)$ and $s' \in S_D$ such that $s'(y) = s(y)$ for all $y \in V(\pi)$ then there is s'_1 such that
   1. $(s', s'_1) \in \rho(\pi)$ and
   2. $s'_1(x) = s'(x)$ for all $x \notin V(\pi)$
   3. $s'_1(y) = s_1(y)$ for all $y \in V(\pi)$.

Basic Observation

$(s, s_1) \in \rho(\pi)$ and s' with $s'(y) = s(y)$ for all $y \in V(\pi)$
then there is s'_1 with

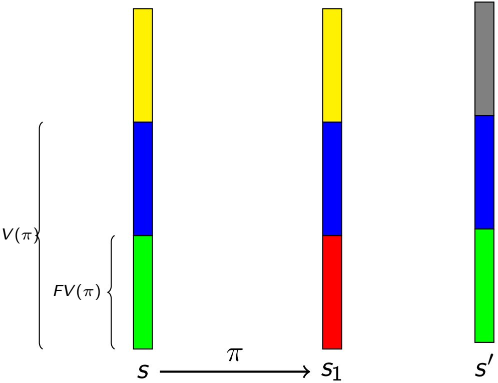
$$(s', s'_1) \in \rho(\pi), \qquad s'_1(x) = \begin{cases} s'(x) & \text{for all } x \notin V(\pi) \\ s_1(x) & \text{for all } x \in V(\pi) \end{cases}.$$



Basic Observation

$(s, s_1) \in \rho(\pi)$ and s' with $s'(y) = s(y)$ for all $y \in V(\pi)$
then there is s'_1 with

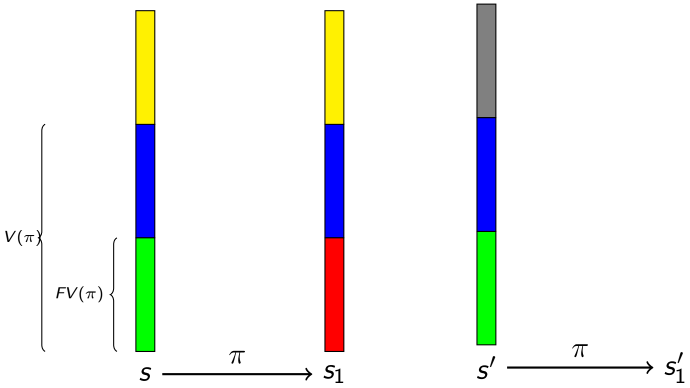
$$(s', s'_1) \in \rho(\pi), \qquad s'_1(x) = \begin{cases} s'(x) & \text{for all } x \notin V(\pi) \\ s_1(x) & \text{for all } x \in V(\pi) \end{cases}.$$



Basic Observation

$(s, s_1) \in \rho(\pi)$ and s' with $s'(y) = s(y)$ for all $y \in V(\pi)$ then there is s'_1 with

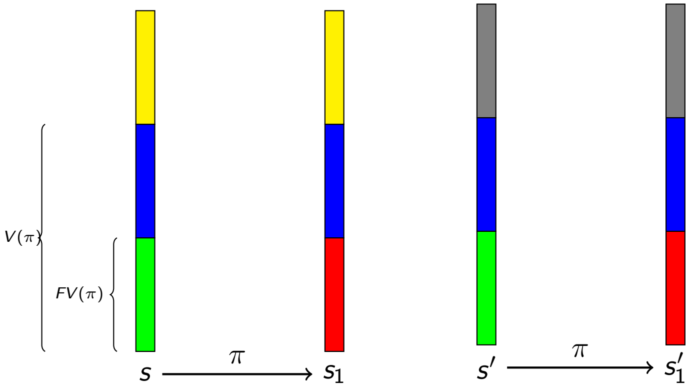
$$(s', s'_1) \in \rho(\pi), \qquad s'_1(x) = \begin{cases} s'(x) & \text{for all } x \notin V(\pi) \\ s_1(x) & \text{for all } x \in V(\pi) \end{cases}.$$



Basic Observation

$(s, s_1) \in \rho(\pi)$ and s' with $s'(y) = s(y)$ for all $y \in V(\pi)$
then there is s'_1 with

$$(s', s'_1) \in \rho(\pi), \qquad s'_1(x) = \begin{cases} s'(x) & \text{for all } x \notin V(\pi) \\ s_1(x) & \text{for all } x \in V(\pi) \end{cases}.$$



Interesting Tautologies

All PDL tautologies

e.g. $[\pi;\tau]\varphi \;\leftrightarrow\; [\pi][\tau]\varphi$

Interesting Tautologies

All PDL tautologies

e.g. $[\pi;\tau]\varphi \;\leftrightarrow\; [\pi][\tau]\varphi$

$$[x := t]\varphi \;\leftrightarrow\; \langle x := t\rangle\varphi$$

Interesting Tautologies

All PDL tautologies
e.g. $[\pi;\tau]\varphi \;\leftrightarrow\; [\pi][\tau]\varphi$

$$[x := t]\varphi \;\leftrightarrow\; \langle x := t\rangle\varphi$$

$$[x := *]\varphi \;\leftrightarrow\; \forall x.\varphi$$

Interesting Tautologies

All PDL tautologies

e.g. $[\pi;\tau]\varphi \leftrightarrow [\pi][\tau]\varphi$

$$[x := t]\varphi \leftrightarrow \langle x := t\rangle\varphi$$

$$[x := *]\varphi \leftrightarrow \forall x.\varphi$$

$$\langle x := *\rangle\varphi \leftrightarrow \exists x.\varphi$$

Interesting Tautologies

All PDL tautologies
e.g. $[\pi;\tau]\varphi \;\leftrightarrow\; [\pi][\tau]\varphi$

$$[x := t]\varphi \;\leftrightarrow\; \langle x := t\rangle\varphi$$

$$[x := *]\varphi \;\leftrightarrow\; \forall x.\varphi$$

$$\langle x := *\rangle\varphi \;\leftrightarrow\; \exists x.\varphi$$

φ a FO formula w/o quantification over x:
$[x := t]\varphi \;\leftrightarrow\; \varphi[x/t]$

Constant Domain Assumption

Is this a tautology?

$$\forall x.[\pi]\varphi \;\leftrightarrow\; [\pi]\forall x.\varphi \qquad \text{if } x \notin V(\pi)$$

Constant Domain Assumption

Karlsruhe Institute of Technology

Is this a tautology?

$$\forall x.[\pi]\varphi \;\leftrightarrow\; [\pi]\forall x.\varphi \qquad \text{if } x \notin V(\pi)$$

Here: Yes. Every state has the same set of objects (so-colled **constant domain assumption**).

Constant Domain Assumption

Karlsruhe Institute of Technology

Is this a tautology?

$$\forall x.[\pi]\varphi \;\leftrightarrow\; [\pi]\forall x.\varphi \qquad \text{if } x \notin V(\pi)$$

Here: Yes. Every state has the same set of objects (so-colled **constant domain assumption**).

But: In some languages, the set of objects can grow (object creation via command **new**)

$$[o := \text{new}]\forall x.\varphi \;\rightarrow\; \forall x.[o := \text{new}]\varphi$$

[To Be or Not To Be Created, "Abstract Object Creation in Dynamic Logic", Ahrendt et al., FM 2009]

Constant Domain Assumption

Is this a tautology? →"Barcan Formula"

$$\forall x.[\pi]\varphi \leftrightarrow [\pi]\forall x.\varphi \qquad \text{if } x \notin V(\pi)$$

Here: Yes. Every state has the same set of objects (so-colled **constant domain assumption**).

But: In some languages, the set of objects can grow (object creation via command **new**)

$$[o := \text{new}]\forall x.\varphi \;\rightarrow\; \forall x.[o := \text{new}]\varphi$$

[To Be or Not To Be Created, “Abstract Object Creation in Dynamic Logic”, Ahrendt et al., FM 2009]

Example

$$z = y \ \wedge \ \forall x.\ f(g(x)) = x \\ \quad \rightarrow \quad [(y := g(y))^*]\langle(y := f(y))^*\rangle y = z$$

Example

$$z = y \;\wedge\; \forall x.\ f(g(x)) = x \qquad \to \quad [(y := g(y))^*]\langle (y := f(y))^* \rangle y = z$$

$$z = y \;\wedge\; \forall x.\ f(g(x)) = x \qquad \to \quad [\textbf{while } p(y) \textbf{ do } y := g(y)]\langle \textbf{while } y \neq z \textbf{ do } y := f(y)\rangle \mathit{true}$$

Indeterminism

DL programs can be indeterminstic

Indeterminism

DL programs can be indeterminstic

Sources of indeterminsm

- Non-deterministic choice $\cup$

Indeterminism

DL programs can be indeterminstic

Sources of indeterminsm

- Non-deterministic choice $\cup$
- Non-deterministic iteration $*$

Indeterminism

DL programs can be indeterminstic

Sources of indeterminsm

- Non-deterministic choice $\cup$
- Non-deterministic iteration $*$
- Non-deterministic assignment $v := *$

Indeterminism

DL programs can be indeterminstic

Sources of indeterminsm

- Non-deterministic choice $\cup$
- Non-deterministic iteration $*$
- Non-deterministic assignment $v := *$

Indeterminism

DL programs can be indeterminstic

Sources of indeterminsm

- Non-deterministic choice $\cup$
- Non-deterministic iteration $*$
- Non-deterministic assignment $v := *$

Example for $v := *$:

$$\textbf{choose } x \textbf{ such that } p(x) \quad :\leftrightarrow \quad x := * \,;\, \textbf{?}p(x)$$

Deterministic programs

Definition

A DL program $\pi \in$ prog is called a while-program if:

1. $\cup$ occurs only within the patterns of **if**,
2. $*$ occurs only within the patterns of **while**,
3. $var := *$ does not occur for any variable $var \in Var$

Deterministic programs

Definition

A DL program $\pi \in$ prog is called a while-program if:

1. $\cup$ occurs only within the patterns of **if**,
2. $*$ occurs only within the patterns of **while**,
3. $var := *$ does not occur for any variable $var \in Var$

Reminder

if φ then α else β

Deterministic programs

Definition

A DL program $\pi \in$ prog is called a while-program if:

1. $\cup$ occurs only within the patterns of **if**,
2. $*$ occurs only within the patterns of **while**,
3. $var := *$ does not occur for any variable $var \in Var$

Reminder

$$\text{if } \varphi \text{ then } \alpha \text{ else } \beta \quad := \quad (\mathbf{?}\varphi \,;\, \alpha) \cup (\mathbf{?}\neg\varphi \,;\, \beta)$$

Deterministic programs

Karlsruhe Institute of Technology

Definition

A DL program $\pi \in$ prog is called a while-program if:

1. $\cup$ occurs only within the patterns of **if**,
2. $*$ occurs only within the patterns of **while**,
3. $var := *$ does not occur for any variable $var \in Var$

Reminder

$$\begin{aligned} \text{if } \varphi \text{ then } \alpha \text{ else } \beta \quad &:= \quad \big(?\varphi\,;\alpha\big) \cup \big(?\neg\varphi\,;\beta\big) \\ \text{while } \varphi \text{ do } \alpha \end{aligned}$$

Deterministic programs

Definition

A DL program $\pi \in$ prog is called a while-program if:

1. $\cup$ occurs only within the patterns of **if**,
2. $*$ occurs only within the patterns of **while**,
3. $var := *$ does not occur for any variable $var \in Var$

Reminder

$$\begin{aligned} \text{if } \varphi \text{ then } \alpha \text{ else } \beta \quad &:= \quad \big(\mathbf{?}\varphi\,;\alpha\big) \cup \big(\mathbf{?}\neg\varphi\,;\beta\big) \\ \text{while } \varphi \text{ do } \alpha \quad &:= \quad (\mathbf{?}\varphi\,;\alpha)^*\,;\mathbf{?}\neg\varphi \end{aligned}$$

Deterministic programs

Semantic Definition

A program $\pi \in \texttt{prog}$ is called deterministic if its accessibility relation is a partial function.

i.e., if $(s, t_1), (s, t_2) \in \rho(\pi) \implies t_1 = t_2$

Deterministic programs

Semantic Definition

A program $\pi \in \texttt{prog}$ is called deterministic if its accessibility relation is a partial function.

i.e., if $(s, t_1), (s, t_2) \in \rho(\pi) \implies t_1 = t_2$

Characterisation of deterministic programs

A program $\pi \in \texttt{prog}$ is deterministic iff $\langle\pi\rangle\varphi \rightarrow [\pi]\varphi$ is a tautology for every formula $\varphi \in \texttt{fml}$.

Deterministic programs

Semantic Definition

A program $\pi \in \texttt{prog}$ is called deterministic if its accessibility relation is a partial function.

i.e., if $(s, t_1), (s, t_2) \in \rho(\pi) \implies t_1 = t_2$

Characterisation of deterministic programs

A program $\pi \in \texttt{prog}$ is deterministic iff $\langle\pi\rangle\varphi \to [\pi]\varphi$ is a tautology for every formula $\varphi \in \texttt{fml}$.

Observation

While programs are deterministic.

Deterministic programs

For determinstic programs:

$[\pi]\varphi$ means "π is **partially** correct with respect to postcondition φ"

$\langle\pi\rangle\varphi$ means "π is **totally** correct with respect to postcondition φ" (i.e. π partially correct **and** π terminates)

Moreover:

Total correctness is partial correctness plus termination:

$$\models \langle\pi\rangle\varphi \;\leftrightarrow\; [\pi]\varphi \wedge \langle\pi\rangle \mathit{true}$$

Expressiveness

Expressiveness of uninterpreted FODL

First order dynamic logic is more expressive than first order logic.

Expressiveness

Expressiveness of uninterpreted FODL

First order dynamic logic is more expressive than first order logic.

Arithmetic cannot be axiomatised in FOL

a direct implication of Gödel's Incompleteness Theorem

Expressiveness

Karlsruhe Institute of Technology

Expressiveness of uninterpreted FODL

First order dynamic logic is more expressive than first order logic.

Arithmetic **cannot** be axiomatised in FOL

a direct implication of Gödel's Incompleteness Theorem

Arithmetic **can** be axiomatised in FODL

... we shall see how ...

Axiomatisation of natural arithmetic

Signature: Let Σ contain:

- constant o (the "zero")
- unary function s (the "successor")

Axiomatisation of natural arithmetic

Signature: Let Σ contain:

- constant o (the "zero")
- unary function s (the "successor")

Goal

Define a FODL formula $\varphi_{\mathbb{N}}$ over Σ s.t.
$D, I \models \varphi_{\mathbb{N}} \quad \text{iff} \quad (D, I(o), I(s)) \overset{\sim}{=} (\mathbb{N}, 0, +1)$

Axiomatisation of natural arithmetic

Signature: Let Σ contain:

- constant o (the "zero")
- unary function s (the "successor")

Goal

Define a FODL formula $\varphi_{\mathbb{N}}$ over Σ s.t.
$D, I \models \varphi_{\mathbb{N}} \quad \text{iff} \quad (D, I(o), I(s)) \cong (\mathbb{N}, 0, +1)$

Idea:

Formalise: "Every element can be reached by a number of loop iterations from zero."

Axiomatisation of natural arithmetic

Signature: Let Σ contain:

- constant o (the "zero")
- unary function s (the "successor")

Goal

Define a FODL formula $\varphi_{\mathbb{N}}$ over Σ s.t.
$D, I \models \varphi_{\mathbb{N}} \quad \text{iff} \quad (D, I(o), I(s)) \cong (\mathbb{N}, 0, +1)$

Idea:

Formalise: "Every element can be reached by a number of loop iterations from zero."

Solution:

Axiomatisation of natural arithmetic

Signature: Let Σ contain:

- constant o (the "zero")
- unary function s (the "successor")

Goal

Define a FODL formula $\varphi_{\mathbb{N}}$ over Σ s.t.
$D, I \models \varphi_{\mathbb{N}}$ iff $(D, I(o), I(s)) \stackrel{\sim}{=} (\mathbb{N}, 0, +1)$

Idea:

Formalise: "Every element can be reached by a number of loop iterations from zero."

Solution:

$$\begin{aligned} \varphi_{\mathbb{N}} \ := \quad & \forall y. \langle x := o; (x := s(x))^* \rangle x = y \\ \wedge \quad & \forall x, y. \ ((s(x) = s(y) \to x = y) \wedge \neg s(x) = o) \end{aligned}$$

Interpreted Dynamic Logic

Fix the first order structure and domain.

Interpreted Dynamic Logic

Fix the first order structure and domain.

In particular: consider

$$\Sigma_{\mathcal{N}} = (\{0, 1, -1, ..., +, *\}, \{<\}) \text{ and } \mathcal{N} = (\mathbb{N}, I_{\mathcal{N}})$$

s.t. $I_{\mathcal{N}}$ interprets the symbols "as expected".

Examples

Valid formulas:

- $3 < 5, x < x + 2, 0 * x = 0$

Examples

Valid formulas:

- $3 < 5, x < x + 2, 0 * x = 0$
- $(p(0) \land \forall x.(p(x) \rightarrow p(x+1))) \rightarrow \forall x.p(x)$

Examples

Valid formulas:

- $3 < 5, x < x + 2, 0 * x = 0$
- $(p(0) \wedge \forall x.(p(x) \to p(x+1))) \to \forall x.p(x)$
- $\neg \exists x(0 < x \wedge x < 1)$

Examples

Valid formulas:

- $3 < 5, x < x + 2, 0 * x = 0$
- $(p(0) \wedge \forall x.(p(x) \rightarrow p(x+1))) \rightarrow \forall x.p(x)$
- $\neg \exists x(0 < x \wedge x < 1)$
- $[y := x\,;(a := *\,; x := x + a)^*]x \geq y$

Examples

Valid formulas:

- $3 < 5, x < x + 2, 0 * x = 0$
- $(p(0) \wedge \forall x.(p(x) \rightarrow p(x+1))) \rightarrow \forall x.p(x)$
- $\neg \exists x(0 < x \wedge x < 1)$
- $[y := x\,;(a := *\,;x := x + a)^*]x \geq y$
- $x_0 = x \wedge y_0 = y$
  $\rightarrow [x := x + y; y := x - y; x := x - y]\, x = y_0 \wedge y = x_0$

Relative Completeness and Calculi

Preliminaries

Karlsruhe Institute of Technology

Encoding sequences (*Gödel*, ∼1930)

There exists a first-order definable function $\beta : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ with:
For every $n \in \mathbb{N}$ and every sequence $c_1, \ldots, c_n \in \mathbb{N}^*$ there exists some c such that $\beta(c, i) = c_i$ for $i = 0, \ldots n$.

Preliminaries

Encoding sequences (*Gödel*, $\sim$1930)

There exists a first-order definable function $\beta : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ with: For every $n \in \mathbb{N}$ and every sequence $c_1, \ldots, c_n \in \mathbb{N}^*$ there exists some c such that $\beta(c, i) = c_i$ for $i = 0, \ldots n$.

c is called the *Gödel number* for $c_1, \ldots, c_n$.
Notation: $c = \ulcorner c_1, \ldots, c_n \urcorner$

Preliminaries

Encoding sequences (*Gödel*, $\sim$1930)

There exists a first-order definable function $\beta : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ with:
For every $n \in \mathbb{N}$ and every sequence $c_1, \ldots, c_n \in \mathbb{N}^*$ there exists some c such that $\beta(c, i) = c_i$ for $i = 0, \ldots n$.

c is called the *Gödel number* for $c_1, \ldots, c_n$.
Notation: $c = \ulcorner c_1, \ldots, c_n \urcorner$

Example encoding:
$\ulcorner c_1, \ldots, c_n \urcorner := 2^{c_1+1} \cdot 3^{c_2+1} \cdot 5^{c_3+1} \cdot \ldots \cdot p_n^{1+c_n}$

Preliminaries

Encoding sequences (*Gödel*, ~1930)

There exists a first-order definable function $\beta : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ with:
For every $n \in \mathbb{N}$ and every sequence $c_1, \ldots, c_n \in \mathbb{N}^*$ there exists some c such that $\beta(c, i) = c_i$ for $i = 0, \ldots n$.

c is called the *Gödel number* for $c_1, \ldots, c_n$.
Notation: $c = \ulcorner c_1, \ldots, c_n \urcorner$

Example encoding:

$$\ulcorner c_1, \ldots, c_n \urcorner := 2^{c_1+1} \cdot 3^{c_2+1} \cdot 5^{c_3+1} \cdot \ldots \cdot p_n^{1+c_n}$$

$$\beta(c, i) = k \quad \Leftrightarrow \quad p_i^{k+1} \mid c \ \wedge \ p_i^{k+2} \nmid c$$

Preliminaries

Encoding sequences (*Gödel*, $\sim$1930)

There exists a first-order definable function $\beta : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ with:
For every $n \in \mathbb{N}$ and every sequence $c_1, \ldots, c_n \in \mathbb{N}^*$ there exists some c such that $\beta(c, i) = c_i$ for $i = 0, \ldots n$.

c is called the *Gödel number* for $c_1, \ldots, c_n$.
Notation: $c = \ulcorner c_1, \ldots, c_n \urcorner$

Example encoding:

$$\ulcorner c_1, \ldots, c_n \urcorner := 2^{c_1+1} \cdot 3^{c_2+1} \cdot 5^{c_3+1} \cdot \ldots \cdot p_n^{1+c_n}$$

$$\beta(c, i) = k \quad \Leftrightarrow \quad p_i^{k+1} \mid c \;\wedge\; p_i^{k+2} \nmid c$$

Example: $\ulcorner 2, 0, 1 \urcorner = 2^3 \cdot 3^1 \cdot 5^2 = 600$

Comparing logics

- **Uninterpreted FODL is more expressive than FOL.**
  There exists a FODL formula such that no FOL formula has the same models.

- **Is FODL over $\mathcal{N}$ more expressive than FOL over $\mathcal{N}$?**
  How can the compare expressiveness with a fixed interpretation?

Relative Completeness

Let L be a logic.
Let $T \subseteq Fml_L$ be a set of formulas (a *theory*).

Relative Completeness

Let L be a logic.
Let $T \subseteq Fml_L$ be a set of formulas (a *theory*).

Oracle

Function $O_T : Fml_L \to \{true, false\}$ with $\varphi \in T \Leftrightarrow O(\varphi) = true$ is called an **oracle** for T.

Relative Completeness

Let L be a logic.
Let $T \subseteq Fml_L$ be a set of formulas (a *theory*).

Oracle

Function $O_T : Fml_L \to \{true, false\}$ with $\varphi \in T \Leftrightarrow O(\varphi) = true$ is called an **oracle** for T.

Relative Completeness *(Cook, 1978)*

A logic is called complete relative to T if there exists a correct and complete calculus which may make use of oracle O_T.

Relative Completeness

Let L be a logic.
Let $T \subseteq Fml_L$ be a set of formulas (a *theory*).

Oracle

Function $O_T : Fml_L \to \{true, false\}$ with $\varphi \in T \Leftrightarrow O(\varphi) = true$ is called an **oracle** for T.

Relative Completeness *(Cook, 1978)*

A logic is called complete relative to T if there exists a correct and complete calculus which may make use of oracle O_T.

Note: T (resp. O_T) may not be computable!

Relative Completeness of FODL

Let $T_{\mathcal{N}} = \{\varphi \mid \mathcal{N} \models \varphi\}$ be the set of valid statements over $\mathbb{N}$.

Theorem

FODL is complete relative to $T_{\mathcal{N}}$.

Programs as Formulas

Programs representable

Every DL program π can be represented as a formula $\kappa(\pi) \in \mathit{Fml}_{FOL_{\mathcal{N}}}$

Programs as Formulas

Programs representable

Every DL program π can be represented as a formula $\kappa(\pi) \in Fml_{FOL_{\mathcal{N}}}$

Here: only one-variable-programs $V(\pi) = \{x\}$

(general case $\rightsquigarrow$ exercise)

Programs as Formulas

Programs representable

Every DL program π can be represented as a formula $\kappa(\pi) \in Fml_{FOL_{\mathcal{N}}}$

Here: only one-variable-programs $V(\pi) = \{x\}$

(general case $\rightsquigarrow$ exercise)

Predicate $\kappa(\pi)(x, x')$ has two free variables:

1. x for the pre-state,
2. x' for the post-state.

Programs as Formulas

Programs representable

Every DL program π can be represented as a formula $\kappa(\pi) \in Fml_{FOL_{\mathcal{N}}}$

Here: only one-variable-programs $V(\pi) = \{x\}$

(general case $\leadsto$ exercise)

Predicate $\kappa(\pi)(x, x')$ has two free variables:

1. x for the pre-state,
2. x' for the post-state.

Modelling goal:

$$s[x'/s'(x)] \models \kappa(\pi)(x, x') \iff (s, s') \in \rho(\pi)$$

Programs as Formulas (II)

$$\kappa(x := t)(x, x') \quad := \quad x' = t$$

Programs as Formulas (II)

$$\kappa(x := t)(x, x') \quad := \quad x' = t$$

$$\kappa(\pi_1 \cup \pi_2)(x, x') \quad := \quad \kappa(\pi_1)(x, x') \;\vee\; \kappa(\pi_2)(x, x')$$

Programs as Formulas (II)

$$\kappa(x := t)(x, x') \quad := \quad x' = t$$

$$\kappa(\pi_1 \cup \pi_2)(x, x') \quad := \quad \kappa(\pi_1)(x, x') \;\vee\; \kappa(\pi_2)(x, x')$$

$$\kappa(\pi_1 \,;\, \pi_2)(x, x') \quad := \quad \exists u.\ \kappa(\pi_1)(x, u) \wedge \kappa(\pi_2)(u, x')$$

Programs as Formulas (II)

$$\begin{aligned}
\kappa(x := t)(x, x') &:= x' = t \\
\kappa(\pi_1 \cup \pi_2)(x, x') &:= \kappa(\pi_1)(x, x') \lor \kappa(\pi_2)(x, x') \\
\kappa(\pi_1 \,;\, \pi_2)(x, x') &:= \exists u.\ \kappa(\pi_1)(x, u) \land \kappa(\pi_2)(u, x') \\
\kappa(\mathbf{?}\varphi)(x, x') &:= \varphi(x) \land x = x'
\end{aligned}$$

Programs as Formulas (II)

$$\kappa(x := t)(x, x') \quad := \quad x' = t$$

$$\kappa(\pi_1 \cup \pi_2)(x, x') \quad := \quad \kappa(\pi_1)(x, x') \ \vee \ \kappa(\pi_2)(x, x')$$

$$\kappa(\pi_1 \,;\, \pi_2)(x, x') \quad := \quad \exists u.\ \kappa(\pi_1)(x, u) \wedge \kappa(\pi_2)(u, x')$$

$$\kappa(\mathbf{?}\varphi)(x, x') \quad := \quad \varphi(x) \wedge x = x'$$

$$\kappa(\pi^*)(x, x') \quad := \quad \exists n. \exists \ulcorner x_1, \ldots, x_n \urcorner.\ x = x_1 \wedge x' = x_n \\ \wedge \forall i < n.\ \kappa(\pi)(x_i, x_{i+1})$$

Reduction of FODL$_{\mathcal{N}}$ to $FOL_{\mathcal{N}}$

Theorem

There is a function $\kappa : Fml_{FODL_{\mathcal{N}}} \to Fml_{FOL_{\mathcal{N}}}$ such that

- $\mathcal{N} \models \varphi \leftrightarrow \kappa(\varphi)$ and
- κ is computable.

Reduction of FODL$_{\mathcal{N}}$ to $FOL_{\mathcal{N}}$

Theorem

There is a function $\kappa : Fml_{FODL_{\mathcal{N}}} \to Fml_{FOL_{\mathcal{N}}}$ such that

- $\mathcal{N} \models \varphi \leftrightarrow \kappa(\varphi)$ and
- κ is computable.

Proof

by structural induction.

Interesting case:

$$\kappa([\pi]\varphi(x)) \;\leftrightarrow\; \forall x'.\, \kappa(\pi)(x, x') \to \kappa(\varphi(x'))$$

(Remainder left as exercise)

A practical calculus

Let φ be a FOL formula and π a program with only FOL tests.

Calculus

$$
\begin{aligned}
[x := t]\varphi &\rightsquigarrow \varphi[x/t] \\
[\pi_1 ; \pi_2]\varphi &\rightsquigarrow [\pi_1][\pi_2]\varphi \\
[\pi_1 \cup \pi_2]\varphi &\rightsquigarrow [\pi_1]\varphi \land [\pi_2]\varphi \\
[\mathbf{?}\psi]\varphi &\rightsquigarrow \psi \to \varphi \\
[\pi^*]\varphi &\rightsquigarrow \quad \mathit{INV} \\
&\qquad \land (\forall \bar{x}.\ \mathit{INV} \to [\pi]\mathit{INV}) \\
&\qquad \land (\forall \bar{x}.\ \mathit{INV} \to \varphi)
\end{aligned}
$$

for an arbitrary formula $\mathit{INV} \in \mathit{Fml}_{\mathit{FOL}}$. $\bar{x} = \mathit{FV}(\pi)$

The calculus allows reduction of FODL formulae to FOL formulae

Weakest Precondition Calculus

Let φ be a FOL formula and π a **while** program (with FOL tests).

Calculus

$$
\begin{array}{rcl}
[x := t]\varphi & \rightsquigarrow & \varphi[x/t] \\
[\pi_1 \,;\, \pi_2]\varphi & \rightsquigarrow & [\pi_1][\pi_2]\varphi \\
[\textbf{if } \psi \textbf{ then } \pi_1 \textbf{ else } \pi_2]\varphi & \rightsquigarrow & (\psi \to [\pi_1]\varphi) \land (\neg\psi \to [\pi_2]\varphi) \\
[\textbf{while } \psi \textbf{ do } \pi]\varphi & \rightsquigarrow & \quad \mathit{INV} \\
 & & \land (\forall \bar{x}.\ \mathit{INV} \land \psi \to [\pi]\mathit{INV}) \\
 & & \land (\forall \bar{x}.\ \mathit{INV} \land \neg\psi \to \varphi)
\end{array}
$$

for an arbitrary formula $\mathit{INV} \in \mathit{Fml}_{\mathit{FOL}}$. $\bar{x} = FV(\pi)$

This is the weakest-precondition calculus (*Dijkstra*, 1975)

Notation: $wlp(\pi, \varphi) = [\pi]\varphi, \quad wp(\pi, \varphi) = \langle\pi\rangle\varphi$

Properties

Let $[\pi]\varphi \rightsquigarrow^* \psi$ be the result of applying the calculus.

Properties

Let $[\pi]\varphi \rightsquigarrow^* \psi$ be the result of applying the calculus.

1. $\models \psi \to [\pi]\varphi$
   ψ is a precondition such that φ is guaranteed to hold after π.

Properties

Let $[\pi]\varphi \rightsquigarrow^* \psi$ be the result of applying the calculus.

1. $\models \psi \to [\pi]\varphi$
   ψ is a precondition such that φ is guaranteed to hold after π.

2. There exist loop invariants such that $\models \psi \leftrightarrow [\pi]\varphi$
   earlier defined $\kappa(\cdot)$ formulates **strongest loop invariants**
   Then ψ is the **weakest precondition**

Properties

Let $[\pi]\varphi \rightsquigarrow^* \psi$ be the result of applying the calculus.

1. $\models \psi \to [\pi]\varphi$
   ψ is a precondition such that φ is guaranteed to hold after π.

2. There exist loop invariants such that $\models \psi \leftrightarrow [\pi]\varphi$
   earlier defined $\kappa(\cdot)$ formulates **strongest loop invariants**
   Then ψ is the **weakest precondition**

3. If $\models \mathit{pre} \to \psi$, then also $\models \mathit{pre} \to [\pi]\varphi$
   Prove pre/post-condition contracts by applying calculus to program and postcondition and then showing implication from precondition.

Arithmetic Completeness

Axioms

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

Axioms for PDL

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

Rules

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

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

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

Arithmetic Completeness

Axioms

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

Axioms for PDL

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

Rules

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

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

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

Theorem

For **any** formula $\varphi \in Fml_{FODL}$: $\mathbb{N} \models \varphi \iff \vdash_{\mathbb{N}} \varphi$