Formale Systeme II: Theorie

Dynamic Logic: Uninterpreted and Interpreted First Order DL

SS 2018

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

Overview – a family of logics

- Modal Logics
  - Propositional Dynamic Logic
    - Dynamic Logic
      - Hybrid DL
      - Java DL
Motivation

First Order Dynamic Logic

Atomic programs are refined to assignments.

Example Formula

\[ x_0 = x \land y_0 = y \rightarrow [x := x + y; y := x - y; x := x - y] \varphi \]
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 $\text{var} := \text{term}$
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:

\[
term ::= \text{var} | f(\text{term}_1, \ldots, \text{term}_{\alpha(f)})
\]

\[
fml ::= \text{true} | \text{false} | p(\text{term}_1, \ldots, \text{term}_{\alpha(p)}) | \text{term}_1 = \text{term}_2 \\
| \lnot \text{fml} | \text{fml}_1 \land \text{fml}_2 | \text{fml}_1 \lor \text{fml}_2 | \text{fml}_1 \rightarrow \text{fml}_2 \\
| \exists \text{var} \cdot \text{fml} | \forall \text{var} \cdot \text{fml} \\
| \langle \text{prog} \rangle \text{fml} | [\text{prog}] \text{fml}
\]

\[
prog ::= \text{var} := \text{term} \\
| \text{var} := * \\
| \text{prog}_1 ; \text{prog}_2 | \text{prog}_1 \cup \text{prog}_2 | \text{prog}^* \\
\]

for \( \text{var} \in \text{Var}, f \in F, p \in P \)
Semantics – Kripke Structures

First Order Structure \((D, I)\)

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

\[ I(f) : D^\alpha(f) \rightarrow D \text{ for function symbol } f \in F \]
\[ I(P) \subseteq D^\alpha(p) \text{ for predicate symbol } p \in P \]

Kripke Structure \((S, \rho)\)

- \(S\) ... set of states
- \(\rho : \text{prog} \rightarrow S \times S\) ... 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 $\text{Var}$:

$$S_D = \text{Var} \rightarrow D$$

For every $t \in \text{Term}_\Sigma$ we denote by $\text{val}_{D,I,s}(t)$ the usual first-order evaluation of $t$ in $(D, I)$; variables are interpreted via $s$.
**Function Update Notation**

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

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

Binary Relation

\( \rho : \text{prog} \rightarrow S_D \times S_D \) assigns accessibility to programs

\[ \rho(x := v) = \{(s, t) \mid t = s[x/val_D], l_s(v)\} \]

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

\[ \rho(\pi_1 \cup \pi_2) = \rho(\pi_1) \cup \rho(\pi_2) \]

\[ \rho(\pi_1 ; \pi_2) = \rho(\pi_1) ; \rho(\pi_2) \quad ; \text{is forward composition} \]

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

\[ \rho(\pi^*) = \rho(\pi)^* \quad * \text{ is refl. transitive closure} \]

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

\[ \rho(?\varphi) = \{(s, s) \mid l, s \models \varphi\} \]
Semantics of Formulae

\[ I, s \models p(t_1, \ldots, t_n) \iff (\text{val}_{I,s}(t_1), \ldots, \text{val}_{I,s}(t_n)) \in I(p) \]

\[ I, s \models t_1 = t_2 \iff \text{val}_{I,s}(t_1) = \text{val}_{I,s}(t_2) \]

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

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

\[ \models \text{ is homomorphic for } \neg, \land, \lor, \to, \forall x, \exists x. \]

We write \( I \models \varphi \) iff \( I, s \models \varphi \) for all \( s \in S \).
Basic Observation

\( \pi \in \text{prog} \) a program

\[ FV(\pi) = \{ x \in \text{Var} \mid \text{ex. } t \text{ such that } x := t \text{ occurs in } \pi \} \]

\[ V(\pi) = \{ x \in \text{Var} \mid x \text{ occurs in } \pi \} \]

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

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

\[(s', s'_1) \in \rho(\pi), \quad s'_1(x) = \begin{cases} s'(x) & \text{for all } x \notin V(\pi) \\ s_1(x) & \text{for all } x \in V(\pi) \end{cases}.\]
Interesting Tautologies

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

\([x := t]\varphi \leftrightarrow \langle x := t\rangle\varphi\)

\([x := \ast]\varphi \leftrightarrow \forall x.\varphi\)

\(\langle x := \ast\rangle\varphi \leftrightarrow \exists x.\varphi\)

\(\varphi\) a FO formula w/o quantification over \(x\):
\([x := t]\varphi \leftrightarrow \varphi[x/t]\)
Constant Domain Assumption

<table>
<thead>
<tr>
<th>Is this a tautology?</th>
<th>→”Barcan Formula”</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \forall x. [\pi] \varphi \leftrightarrow [\pi] \forall x. \varphi ) if ( x \not\in V(\pi) )</td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

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

\[ z = y \land \forall x. \ f(\ g(x)) = x \]

\[ \rightarrow \ [(y := g(y))^]* \langle (y := f(y))^* \rangle y = z \]

\[ z = y \land \forall x. \ f(\ g(x)) = x \]

\[ \rightarrow \ [\text{while } p(y) \text{ do } y := g(y)] \langle \text{while } y \neq z \text{ do } y := f(y) \rangle \text{true} \]
Indeterminism

DL programs can be indeterministic

Sources of indeterminism

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

Example for \( v := * \):

choose \( x \) such that \( p(x) \) :\( \leftrightarrow \) \( x := * ; ?p(x) \)
Deterministic programs

Definition

A DL program \( \pi \in \text{prog} \) is called a while-program if:

1. \( \cup \) occurs only within the patterns of \( \text{if} \),
2. \( \ast \) occurs only within the patterns of \( \text{while} \),
3. \( \text{var} \) :\( =\ast\) does not occur for any variable \( \text{var} \in \text{Var} \)

Reminder

\[
\begin{align*}
\text{if } \varphi \text{ then } \alpha \text{ else } \beta & := (\varphi ; \alpha) \cup (\neg \varphi ; \beta) \\
\text{while } \varphi \text{ do } \alpha & := (\varphi ; \alpha)^* ; \neg \varphi
\end{align*}
\]
Deterministic programs

Semantic Definition
A program \( \pi \in \text{prog} \) 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 \( \pi \in \text{prog} \) is deterministic iff \( \langle \pi \rangle \varphi \rightarrow [\pi] \varphi \) is a tautology for every formula \( \varphi \in \text{fml} \).

Observation
While programs are deterministic.
Deterministic programs

For deterministic programs:

\[
[\pi] \varphi \quad \text{means “} \pi \text{ is \textbf{partially} correct with respect to postcondition } \varphi \text{”}
\]

\[
\langle \pi \rangle \varphi \quad \text{means “} \pi \text{ is \textbf{totally} correct with respect to postcondition } \varphi \text{”}
\]
(i.e. \( \pi \) partially correct and \( \pi \) terminates)

Moreover:
Total correctness is partial correctness plus termination:

\[
\models \langle \pi \rangle \varphi \iff [\pi] \varphi \land \langle \pi \rangle \text{true}
\]
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

Arithmetic can be axiomatised in FODL
... we shall see how ...
Axiomatisation of natural arithmetic

**Signature:** Let $\Sigma$ contain:
- constant $o$ (the “zero”)
- unary function $s$ (the “successor”)

**Goal**
Define a FODL formula $\varphi_N$ over $\Sigma$ s.t.
$D, I \models \varphi_N$ iff $(D, I(n), I(s)) \simeq (\mathbb{N}, 0, +1)$

**Idea:**
Formalise: “Every element can be reached by a number of loop iterations from zero.”

**Solution:**
$$\varphi_N := \forall y. \langle x := o; (x := s(x))^\ast \rangle x = y \\
\land \forall x, y. ((s(x) = s(y) \rightarrow x = y) \land \neg s(x) = o)$$
Fix the first order structure and domain.

In particular: consider

$$\Sigma_N = (\{0, 1, -1, \ldots, +, \ast\}, \{<\})$$ and $$N = (\mathbb{N}, I_N)$$

s.t. $I_N$ interprets the symbols “as expected”.
Examples

Valid formulas:

- $3 < 5, x < x + 2, 0 \times x = 0$

- $(p(0) \land \forall x. (p(x) \rightarrow p(x + 1))) \rightarrow \forall x. p(x)$

- $\neg \exists x (0 < x \land x < 1)$

- $[y := x; (a := \times; x := x + a)^*]x \geq y$

- $x_0 = x \land y_0 = y$
  $\rightarrow [x := x + y; y := x - y; x := x - y]x = y_0 \land y = x_0$
Relative Completeness and Calculi
Encodings of sequences (Gödel, ~1930)

There exists a first-order definable function $\beta : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ 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: $c = \lceil c_1, \ldots, c_n \rceil$

**Example encoding:**

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

$\beta(c, i) = k \iff p_i^{k+1} | c \land p_i^{k+2} \not| c$

**Example:** $\lceil 2, 0, 1 \rceil = 2^3 \cdot 3^1 \cdot 5^2 = 600$
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).

Oracle

Function \( O_T : Fml_L \rightarrow \{\text{true}, \text{false}\} \) with \( \varphi \in T \iff O(\varphi) = \text{true} \) 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 $T_N = \{ \varphi \mid N \models \varphi \}$ be the set of valid statements over $\mathbb{N}$.

**Theorem**

FODL is complete relative to $T_N$.
Programs as Formulas

Programs representable

Every DL program $\pi$ can be represented as a formula $\kappa(\pi) \in Fml_{FOL_N}$

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.

Modelling goal:

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

\[ \kappa(x := t)(x, x') := x' = t \]

\[ \kappa(\pi_1 \cup \pi_2)(x, x') := \kappa(\pi_1)(x, x') \lor \kappa(\pi_2)(x, x') \]

\[ \kappa(\pi_1 ; \pi_2)(x, x') := \exists u. \ \kappa(\pi_1)(x, u) \land \kappa(\pi_2)(u, x') \]

\[ \kappa(\exists \varphi)(x, x') := \varphi(x) \land x = x' \]

\[ \kappa(\pi^*)(x, x') := \exists n. \exists x_1, \ldots, x_n . \ x = x_1 \land x' = x_n \]
\[ \land \forall i < n. \ \kappa(\pi)(x_i, x_{i+1}) \]
Reduction of \( \text{FODL}_\mathcal{N} \) to \( \text{FOL}_\mathcal{N} \)

**Theorem**

There is a function \( \kappa : \text{Fml}_{\text{FODL}_\mathcal{N}} \rightarrow \text{Fml}_{\text{FOL}_\mathcal{N}} \) such that

- \( \mathcal{N} \models \varphi \iff \kappa(\varphi) \) and
- \( \kappa \) is computable.

**Proof**

by structural induction.

Interesting case:

\[
\kappa([\pi] \varphi(x)) \iff \forall x'. \kappa(\pi)(x, x') \rightarrow \kappa(\varphi(x'))
\]

( Remainder left as exercise )
A practical calculus

Let $\varphi$ be a FOL formula and $\pi$ a program with only FOL tests.

Calculus

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

for an arbitrary formula $\text{INV} \in Fml_{\text{FOL}}$.  \(\bar{x} = \text{FV}(\pi)\)

The calculus allows reduction of FODL formulae to FOL formulae.
Weakest Precondition Calculus

Let $\varphi$ be a FOL formula and $\pi$ a while program (with FOL tests).

**Calculus**

$$[x := t] \varphi \rightsquigarrow \varphi[x/t]$$

$$[\pi_1 ; \pi_2] \varphi \rightsquigarrow [\pi_1][\pi_2] \varphi$$

$$[\text{if } \psi \text{ then } \pi_1 \text{ else } \pi_2] \varphi \rightsquigarrow (\psi \rightarrow [\pi_1] \varphi) \land (\neg \psi \rightarrow [\pi_2] \varphi)$$

$$[\text{while } \psi \text{ do } \pi] \varphi \rightsquigarrow INV$$

$$\land (\forall \bar{x}. INV \land \psi \rightarrow [\pi] INV)$$

$$\land (\forall \bar{x}. INV \land \neg \psi \rightarrow \varphi)$$

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

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.

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

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

3. If $\models pre \rightarrow \psi$, then also $\models pre \rightarrow [\pi]\varphi$
   
   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
\[ \langle x := t \rangle \varphi \leftrightarrow \varphi[x/t] \]

**Rules**
\[
\begin{align*}
F, F & \rightarrow G \\
\frac{}{G} \\
F & \frac{F}{[\pi]F} \\
F & \frac{\forall x F}{\forall \pi F}
\end{align*}
\]
(modus ponens)
(generalisations)

\[ \forall n(F(n + 1) \rightarrow \langle \pi \rangle F(n)) \]
\[ \forall n(F(n) \rightarrow \langle \pi^* \rangle F(0)) \]
(convergence)

**Theorem**
For any formula \( \varphi \in Fml_{FODL} \):
\[ \mathbb{N} \models \varphi \iff \vdash_{\mathbb{N}} \varphi \]