Non-classical logics

Lecture 19: Dynamic Logic

Viorica Sofronie-Stokkermans

sofronie@uni-koblenz.de

Motivation

A Simple Programming Language

Logical basis

Typed first-order predicate logic
(Types, variables, terms, formulas, . . .)

Assumption for examples

The signature contains a type Nat and appropriate symbols:

- function symbols $0, s, +, *$
  (terms s(0), s(s(0)), . . . written as 1,2, . . .)
- predicate symbols $\doteq, \leq, <, \geq, >$

NOTE: This is a “convenient assumption” not a definition

Motivation

Programs

- Assignments: $X := t$ $\quad X$: variable, t:term
- Test: if B then a else b fi

  B: quant.-free formula, a, b: programs
- Loop: while B do a od

  B: quantifier-free formula, a: program
- Composition: $a; b$ $\quad a, b$ programs

WHILE is computationally complete

Motivation

WHILE: Examples

Compute the square of X and store it in Y

$$Y := X * X$$

If X is positive then add one else subtract one

$$\text{if } X > 0 \text{ then } X := X + 1 \text{ else } X := X - 1 \text{ fi}$$

Motivation

WHILE: Example - Square of a Number

Compute the square of X (the complicated way)

Making use of: $n^2 = 1 + 3 + 5 + \cdots + (2 * n - 1)$

```
I := 0;
Y := 0;
while I < X do
Y :=Y +2*I+1;
I := I+1
od
```

Motivation

WHILE: Operational Semantics

Given

A (fixed) first-order structure $\mathcal{A}$ interpreting the function and predicate symbols in the signature

State

$s = (\mathcal{A}, \beta)$ where β is a variable assignment (i.e. function interpreting the variables)

Motivation

State update

$s[e/X] = (\mathcal{A}, \beta[X \mapsto e])$

with $\beta[X \mapsto e](Y) = \begin{cases} e & \text{if } Y = X \\ \beta(Y) & \text{otherwise} \end{cases}$

Motivation

Define the relation $R(\alpha)$ as follows (we write $s[\alpha]s'$ instead of $sR(\alpha)s'$):

- $s[X := t]s'$ iff $s' = s[s(t)/X]$
- $s[\text{if } B \text{ then } \alpha \text{ else } \beta \text{ fi}]s'$ iff $s \models B$ and $s[\alpha]s'$ or $s \models \neg B$ and $s[\beta]s'$.
- $s[\text{while } B \text{ do } \alpha \text{ od}]s'$ iff there are states $s = s_0, \ldots, s_t = s'$ s.t. $s_i \models B$ for $0 \leq i \leq t-1$ and $s_t \models \neg B$ and $s_0[\alpha]s_1, s_1[\alpha]s_2, \ldots, s_{t-1}[\alpha]s_t$
- $s[\alpha; \beta]s'$ iff there is a state s'' such that $s[\alpha]s''$ and $s''[\beta]s'$

If α is a deterministic program, $[\alpha]$ is a partial function

Motivation

A Different Approach to WHILE

Programs

- $X := t$ (atomic program)
- $\alpha ; \beta$ (sequential composition)
- $\alpha \cup \beta$ (non-deterministic choice)
- α^* (non-deterministic iteration, n times for some $n \geq 0$)
- $F?$ (test)
  remains in initial state if F is true,
  does not terminate if F is false

Motivation

Restriction to deterministic programs

Non-deterministic program constructors may only be used in

if B then α else β fi $\equiv (B?; \alpha) \cup ((\neg B)?; \beta)$

while B do α od $\equiv (B?; \alpha)^*; (\neg B)?$

Motivation

Expressing Program Properties

Logic for expressing properties

Full first-order logic (usually with arithmetic)

Partial correctness assertion (Hoare formula)

$$\{P\}\alpha\{Q\}$$

Meaning:
If α is started in a state satisfying P and terminates, then its final state satisfies Q

Formally:
$\{P\}\alpha\{Q\}$ is valid iff for all states s, s', if $s \models P$ and $s[a]s'$, then $s' \models Q$

Examples

$\{X > 0\}X := X + 1\{X > 1\}$

$\{\text{even}(X)\}X := X + 2\{\text{even}(X)\}$
where $\text{even}(X) \equiv \exists Z(X = 2 * Z)$

$\{\mathit{true}\}\alpha_{\text{square}}\{Y = X * X\}$

Examples

$\{X > 0\}X := X + 1\{X > 1\}$

$\{\text{even}(X)\}X := X + 2\{\text{even}(X)\}$

where $\text{even}(X) \equiv \exists Z(X = 2 * Z)$

$\{\mathit{true}\}\alpha_{\text{square}}\{Y = X * X\}$

Verification: Use annotation of programs with "invariants"

Dynamic Logic

The idea of dynamic logic

- Annotated programs use formulas within programs
- Dynamic Logic uses programs within formulas
- Instead of "assert F" after program segment α, write: $[\alpha]F$

$\mapsto$ multi-modal logic

Dynamic Logic

Dynamic logic is a language for specifying programming languages.

The original work on dynamic logic is by Vaughan Pratt (1976) and by David Harel (1979).

Propositional Dynamic Logic

Propositional dynamic logic (PDL) is a multi-modal logic with structured modalities.

For each program α, there is:

- a box-modality $[\alpha]$ and
- a diamond modality $\langle\alpha\rangle$.

PDL was developed from first-order dynamic logic by Fischer-Ladner (1979) and has become popular recently.

Here we consider regular PDL.

Propositional Dynamic Logic

Syntax

Prog set of programs

$\mathsf{Prog}_0 \subseteq \mathsf{Prog}$: set of atomic programs

Π: set of propositional variables

The set of formulae $\mathsf{Fma}^{PDL}_{\mathsf{Prog},\Pi}$ of (regular) propositional dynamic logic and the set of programs P_0 are defined by simultaneous induction as follows:

PDL: Syntax

Formulae:

$$
\begin{array}{lcll}
F, G, H & ::= & \bot & \text{(falsum)} \\
 & \mid & \top & \text{(verum)} \\
 & \mid & p & p \in \Pi_0 \text{ (atomic formula)} \\
 & \mid & \neg F & \text{(negation)} \\
 & \mid & (F \wedge G) & \text{(conjunction)} \\
 & \mid & (F \vee G) & \text{(disjunction)} \\
 & \mid & (F \rightarrow G) & \text{(implication)} \\
 & \mid & (F \leftrightarrow G) & \text{(equivalence)} \\
 & \mid & [\alpha]F & \text{if } \alpha \in \text{Prog} \\
 & \mid & \langle\alpha\rangle F & \text{if } \alpha \in \text{Prog}
\end{array}
$$

Programs:

$$
\begin{array}{lcll}
\alpha, \beta, \gamma & ::= & \alpha_0 & \alpha_0 \in \text{Prog}_0 \text{ (atomic program)} \\
 & \mid & F? & F \text{ formula (test)} \\
 & \mid & \alpha ; \beta & \text{(sequential composition)} \\
 & \mid & \alpha \cup \beta & \text{(non-deterministic choice)} \\
 & \mid & \alpha^* & \text{(non-deterministic repetition)}
\end{array}
$$

Semantics

A PDL structure $\mathcal{K} = (S, R(), I)$ is a multimodal Kripke structure with an accessibility relation for each atomic program. That is it consists of:

- a non-empty set S of states
- an interpretation $R() : \mathsf{Prog}_0 \to S \times S$ of atomic programs that assigns a transition relation $R(\alpha)$ to each atomic program α
- an interpretation $I : \Pi \times S \to \{0, 1\}$

PDL: Semantics

The interpretation of PDL relative to a PDL structure $\mathcal{K} = (S, R(), I)$ is defined by extending $R()$ to Prog and extensing I to $\text{Fma}^{PDL}_{\text{Prop}_0}$ by the following simultaneously inductive definition:

Interpretation of formulae/programs

$$
\begin{array}{lll}
val_{\mathcal{K}}(p, s) & = & I(p, s) \\
val_{\mathcal{K}}(\neg F, s) & = & \neg_{\text{Bool}} val_{\mathcal{K}}(F, s) \\
val_{\mathcal{K}}(F \wedge G, s) & = & val_{\mathcal{K}}(F, s) \wedge_{\text{Bool}} val_{\mathcal{K}}(G, s) \\
val_{\mathcal{K}}(F \vee G, s) & = & val_{\mathcal{K}}(F, s) \vee_{\text{Bool}} val_{\mathcal{K}}(G, s) \\
val_{\mathcal{K}}([\alpha]F, s) = 1 & iff & \text{ for all } t \in S \text{ with } (s, t) \in R(\alpha), val_{\mathcal{K}}(F, t) = 1 \\
val_{\mathcal{K}}(\langle\alpha\rangle\, F, s) = 1 & iff & \text{ for some } t \in S \text{ with } (s, t) \in R(\alpha), val_{\mathcal{K}}(F, t) = 1 \\
R([F?]) & = & \{(s, s) \mid val_{\mathcal{K}}(F, s) = 1\} \\
 & & (F\text{? has the same meaning as: if } F \text{ then skip else do not terminate}) \\
R(\alpha \cup \beta) & = & R(\alpha) \cup R(\beta) \\
R(\alpha; \beta) & = & \{(s, t) \mid \text{ there exists } u \in S\ s.t.(s, u) \in R(\alpha) \text{ and } (u, t) \in R(\beta)\} \\
R(\alpha^*) & = & \{(s, t) \mid \text{ there exists } n \geq 0 \text{ and there exist } u_0, \dots, u_n \in S \text{ with} \\
 & & s = u_0, y = u_n, (u_0, u_1), \dots, (u_{n-1}, u_n) \in R(\alpha)\}
\end{array}
$$

Interpretation of formulae/programs

- $(\mathcal{K}, s)$ satisfies F (notation $(\mathcal{K}, s) \models F$) iff $val_{\mathcal{K}}(F, s) = 1$.
- F is valid in $\mathcal{K}$ (notation $\mathcal{K} \models F$) iff $(\mathcal{K}, s) \models F$ for all $s \in S$.
- F is valid (notation $\models F$) iff $\mathcal{K} \models F$ for all PDL-structures $\mathcal{K}$.

Axiom system for PDL

Camp :	$[\alpha;\beta]A \leftrightarrow [\alpha][\beta]A,$
Alt :	$[\alpha \cup \beta]A \leftrightarrow [\alpha]A \wedge [\beta]A,$
Mix :	$[\alpha^*]A \rightarrow A \wedge [\alpha][\alpha^*]A,$
Ind :	$[\alpha^*](A \rightarrow [a]A) \rightarrow (A \rightarrow [\alpha^*]A),$
Test :	$[A?]B \leftrightarrow (A \rightarrow B).$

We will show that PDL is determined by PDL structures, and has the finite model property.

Soundness and Completeness of PDL

Proof similar to the proof in the case of the modal system K (with small differences)

Theorem. If the formula F is provable in the inference system for PDL then F is valid in all PDL structures.

Proof: The axioms are valid in every PDL structure. Easy computation (examples on the blackboard).

Soundness and Completeness of PDL

Theorem. If the formula F is is valid in all PDL structures then F is provable in the inference system for PDL.

Proof

Idea:
Assume that F is not provable in the inference system for PDL.

We show that:

(1) $\neg F$ is consistent with the set L of all theorems of PDL

(2) We can construct a "canonical" PDL structure $\mathcal{K}_L$ and a state w in this PDL structure such that $(\mathcal{K}, w) \models \neg F$.

Contradiction!

Consistent sets of formulae

Let L be a set of PDL formulae which:
(1) contains all propositional tautologies
(2) contains axiom PDL
(3) is closed under modus ponens and generalization
(4) is closed under instantiation

Definition. A subset $F \subseteq L$ is called L-inconsistent iff there exist formulae $A_1, \ldots, A_n \in F$ such that

$$(\neg A_1 \vee \cdots \vee \neg A_n) \in L$$

F is called L-consistent iff it is not L-inconsistent.

Definition. A consistent set F of PDL formulae is called maximal L-consistent if for every formula A wither $A \in F$ or $\neg A \in F$.

Consistent sets of formulae

Let L be as before. In what follows we assume that L is consistent.

Theorem. Let F be a maximal L-consistent set of formulae. Then:

(1) For every formula A, either $A \in F$ or $\neg A \in F$, but not both.

(2) $A \vee B \in F$ iff $A \in F$ or $B \in F$

(3) $A \wedge B \in F$ iff $A \in F$ and $B \in F$

(4) $L \subseteq F$

(5) F is closed under Modus Ponens

Theorem. Every consistent set F of formulae is contained in a maximally consistent set of formulae.

Proofs: As for modal logic.

Canonical models

Goal: Assume F is not a theorem. Construct a PDL structure $\mathcal{K}$ and a state w of $\mathcal{K}$ such that $(\mathcal{K}, w) \models \neg F$.

States:

State of $\mathcal{K}$: maximal consistent set of formulae.

Intuition: $(\mathcal{K}, W) \models F$ iff $F \in W$.

Accessibility relation:

Intuition:
$(\mathcal{K}, W) \models [\alpha]F$ iff for all $W', ((W, W') \in R(\alpha) \rightarrow (\mathcal{K}, W') \models F)$

Canonical models

Goal: Assume F is not a PDL theorem. Construct a PDL structure $\mathcal{K}$ and a state w of $\mathcal{K}$ such that $(\mathcal{K}, w) \models \neg F$.

States:

State of $\mathcal{K}$: maximal consistent set of formulae.

Intuition: $(\mathcal{K}, W) \models F$ iff $F \in W$.

Accessibility relation:

Intuition:
$(\mathcal{K}, W) \models [\alpha]F$ iff for all $W', ((W, W') \in R(\alpha) \rightarrow (\mathcal{K}, W') \models F$
$[\alpha]F \in W$ iff for all $W', ((W, W') \in R(\alpha) \rightarrow F \in W')$

$(W, W') \in R(\alpha)$ iff $W' \supseteq \{F \mid [\alpha]F \in W\}$

Canonical models

Theorem. $\mathcal{K}$ satisfies all PDL structure conditions except $R(\alpha^*) \subseteq (R(\alpha))^*$.

Proof: By direct checking.

Example: $R(\alpha;\beta) \subseteq R(\alpha) \circ R(\beta)$
Assume $(W, W') \in R(\alpha;\beta)$. Then $W' \subseteq \{F \mid [\alpha;\beta]F \in W\}$.

We want to show that there exists W_0 with $(W, W_0) \in R(\alpha)$ and $(W_0, W') \in R(\beta)$.

It to show that $W_0 = \{B \mid [\alpha]B \in W\} \cup \{\neg[\beta]D \mid D \notin W'\}$ is PDL-consistent. For this, the PDL-theorem $[\alpha][\beta]A \to [\alpha][\beta]A$ is used.

Canonical models

Theorem. Assume F is not a PDL theorem. We can construct a PDL structure $\mathcal{K}'$ and a state w of $\mathcal{K}'$ such that $(\mathcal{K}', w) \models \neg F$.

Proof. To obtain a PDL structure that falsifies F we will collapse $\mathcal{K}$ by a suitable Γ that contains F. The closure rules for Γ that will be needed are:

- Γ is closed under subformulae;
- $[B?]D \in \Gamma$ implies $B \in \Gamma$;
- $[\alpha;\beta]B \in \Gamma$ implies $[\alpha][\beta]B \in \Gamma$;
- $[\alpha \cup \beta]B \in \Gamma$ implies $[\alpha]B, [\beta]B \in \Gamma$;
- $[\alpha^*]B \in \Gamma$ implies $[\alpha][\alpha^*]B \in \Gamma$

A set Γ satisfying these conditions will be called closed.

Completeness/Decidability of PDL

Theorem. If Γ is the smallest closed set containing a given formula F, then Γ is finite.

Proof. The point is to show that closing Subformulae(F) under the above rules produces only finitely many new formulae.

Define a formula to be boxed if it is prefixed by a modal connective, i.e. is of the form $[\alpha]B$ for some α and B. Each time we apply a closure rule, new boxed formulae appear on the right side of the rule, and further rules may apply to these new formulae.

But observe that the programs α indexing prefixes $[\alpha]$ on the right side are in all cases shorter in length than those indexing the prefix on the left of the rule in question. Hence we will eventually produce only atomic prefixes on the right, and run out of rules to apply.

Completeness/Decidability of PDL

Having determined that Γ, the smallest closed set containing F, is finite, we perform a Γ- filtration of $\mathcal{K}$.

Completeness/Decidability of PDL

Reminder (modal logic K):
Fix a model $\mathcal{K} = (S, R, I)$ and a set $\Gamma \subseteq Fma_\Sigma$ that is closed under subformulae, i.e. $B \in \Gamma$ implies Subformulae$(B) \subseteq \Gamma$.

For each $s \in S$, define

$$\Gamma_s = \{B \in \Gamma \mid (\mathcal{K}, s) \models B\}$$

and put $s \sim_\Gamma t$ iff $\Gamma_s = \Gamma_t$,

Then $s \sim_\Gamma t$ iff for all $B \in \Gamma$, $(\mathcal{K}, s) \models B$ iff $(\mathcal{K}, t) \models B$.

Fact: $\sim_\Gamma$ is an equivalence relation on S.

Let $[s] = \{t \mid s \sim_\Gamma t\}$ be the $\sim_\Gamma$-equivalence class of s.

Let $S_\Gamma := \{[s] \mid s \in S\}$ be the set of all such equivalence classes.

Decidability/Completeness

Goal: $(\mathcal{K}, s) \models A \quad \mapsto \quad (\mathcal{K}', s') \models A, \mathcal{K}' = (S', R', I')$.

Step 1: $S' := S_\Gamma$, where $\Gamma = \text{Subformulae}(S)$

Step 2: $I' : (\Pi \cap \Gamma) \times S' \rightarrow \{0, 1\}$ def. by $I'(P, [s]) = I(P, s)$

Step 3: R' def. e.g. by: $([s], [t]) \in R'$ iff $\exists s' \in [s], \exists t' \in [t]$: $(s', t') \in R$

Same construction for PDL (only we need to define a relation for each program).

Theorem: $\mathcal{K}'$ is a PDL structure (and a filtration of $\mathcal{K}$.

If $(\mathcal{K}, s) \not\models F$ then $(\mathcal{K}', [s]) \models \neg F$.
$\mapsto$ completeness.

Lemma. If Γ is finite, then S_Γ is finite and has at most 2^n elements, where n is the number of elements of Γ.
$\mapsto$ decidability

Conclusions

Although PDL appears to be more expressive than modal logic, it is still decidable (it has the finite model propety).

Proof calculi for PDL exist.

For really reasoning about programs, often *first order dynamic logic* is needed (undecidable)