Formal Specification and Verification

Propositional Dynamic Logic 1

30.01.2017

Viorica Sofronie-Stokkermans

e-mail: sofronie@uni-koblenz.de

Overview

- **Model checking:**

  Finite transition systems / CTL properties

  States are "entities" (no precise description, except for labelling functions)

  No precise description of actions (only $\rightarrow$ important)

Overview

- **Model checking:**

  Finite transition systems / CTL properties

  States are "entities" (no precise description, except for labelling functions)

  No precise description of actions (only $\rightarrow$ important)

Extensions in two possible directions:

- More precise description of the actions/events
  - Propositional Dynamic Logic
  - Hoare logic
- More precise description of states (and possibly also of actions)
  - succinct representation: formulae represent a set of states
  - deductive verification

Motivation

Example Program: Square

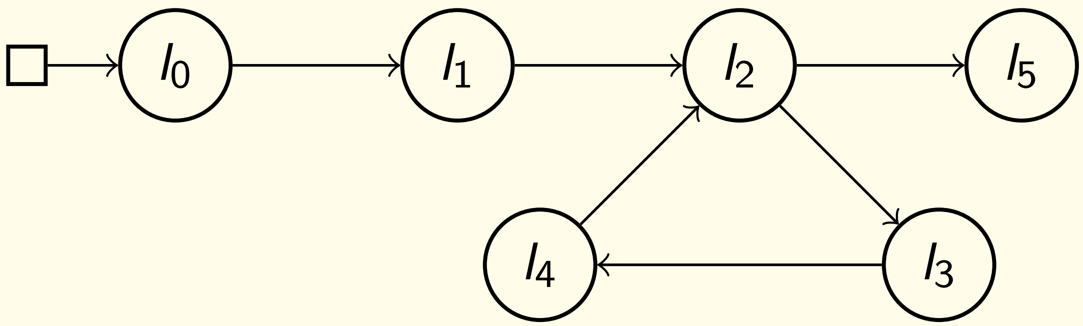
```
I := 0;
Y := 0;
while I < X do
  Y := Y+2*I+1;
  I := I+1
od
```

We would like to prove something like "$A\Diamond(\text{terminated} \wedge \texttt{Y=X*X})$".

Motivation

Example Program: Square

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

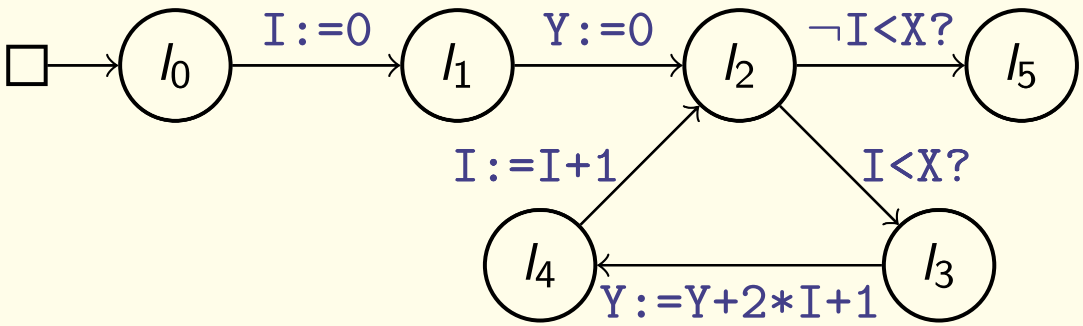


We would like to prove something like "$A\Diamond$(terminated $\wedge$ `Y=X*X`)".

Motivation

Example Program: Square

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



We would like to prove something like "$A\Diamond$(terminated $\wedge$ `Y=X*X`)".

CTL* too weak: Transitions carry meaning.

Dynamic Logic: [prog]`Y=X*X`

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: $X := t$ 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: $a; b$ a, b programs

WHILE is computationally complete

Motivation

WHILE: Examples

Assignment: Compute the square of X and store it in Y

$$Y := X * X$$

Test: 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

Program with a while loop and composition:

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

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

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

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 $s[\alpha]s'$, 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

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

Π: set of propositional variables

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

PDL: Syntax

Formulae:

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

Programs:

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