Dynamic Logic

Principles and Applications

Wolfgang Ahrendt

22 April 2021

Outline

- Preamble: Modal Logics
- Propositional DL
- First-order DL
- Symbolic Execution
- DL at Scale
- Extending DL by Dynamic Domains
- Differential DL

Part I

Modal Logics

Modal Logic

- Pre-history: Aristotle, ..., W. of Ockham, ...
- Modern modal logic: C.I. Lewis (1912)
- Semantics (1950s): A. Prior, J. Hintikka, S. Kripke
- Modal logics come in many flavours: K, T, S4, S5, D, ...
  (vary in properties of reachibility relation)
- Application areas:
  philosophy of language, epistemology, metaphysics, computation
- Variants of modal logic with many applications to computation:
  - Temporal Logic
    (A.N. Prior 1957, A. Pnuelli 1977)
  - Dynamic Logic
    (V.R. Pratt 1976, "Semantical considerations on Floyd-Hoare Logic")

Modal Logics

- MLs represent statements about necessity and possibility
  - $\Box\varphi$ "φ in all states we can **reach from here**"
  - $\Diamond\varphi$ "φ in some state we can **reach from here**"
  - "reach" is *one* step in reachability relation
- Temporal logic (*endogenous* modal logic)
  - $\Box\varphi$ "φ in all states we can **reach by letting some time pass**"
  - $\Diamond\varphi$ "φ in some state we can **reach by letting some time pass**"
  - "letting some time pass" is *one* step in reachability relation
  - reachability is reflexive transitive closure of clock tick
  - (normally no defined like that, but with traces and multiple steps)
- Dynamic logic (*exogenous* modal logic)
  - $[\alpha]\varphi$ "φ in all states we can **reach by** α"
  - $\langle\alpha\rangle\varphi$ "φ in some state we can **reach by** α"
  - "reach by α" refers to *one* step in α-reachability relation

Part II

Propositional Dynamic Logic

Propositional Dynamic Logic (PDL)

- Normally defined for non-deterministic programs:
  - as a means of abstraction
  - to model an uncontrollable environment (later)

Propositional Dynamic Logic (PDL)

Propositional DL Formulas

(Assume sets of atomic formulas and programs.)

If φ, ψ are formulas, and α, β are programs, then

- $\neg\varphi$
- $\varphi \vee \psi$
- $\langle\alpha\rangle\varphi$ (*some execution of α leads to a state where φ*)

are also formulas,and

- $\alpha;\beta$ (*sequence*)
- $\alpha \cup \beta$ (*non-deterministic choice*)
- α^* (*execute α a finite, non-deterministic number of times*)
- $?\varphi$ (*test φ, proceed if true, fail if false*)

are also programs.

Semantics of PDL

Assume:

- atomic formulas: AF
- atomic programs: AP

Semantics of PDL Formulas

Kripke model $\mathcal{M} = (S, \mathcal{I})$ where

- Set of states $S = \{u, v, \ldots\}$
- Interpretation of atomic formulas $\mathcal{I}$: $AF \to 2^S$
- Interpretation of atomic formulas $\mathcal{I}$: $AP \to 2^{S \times S}$

Semantics of PDL

Let f be any atomic formula, p be any atomic program

Semantics of PDL Formulas

Meaning of formula $\varphi^{\mathcal{M}} \subseteq S$:

- $f^{\mathcal{M}} = \mathcal{I}(f)$
- $p^{\mathcal{M}} = \mathcal{I}(p)$
- $(\varphi \vee \psi)^{\mathcal{M}} = \varphi^{\mathcal{M}} \cup \psi^{\mathcal{M}}$
- $(\neg\varphi)^{\mathcal{M}} = S - \varphi^{\mathcal{M}}$

Semantics of PDL

Let f be any atomic formula, p be any atomic program

Semantics of PDL Formulas

Meaning of formula $\varphi^{\mathcal{M}} \subseteq S$, meaning of program $\alpha^{\mathcal{M}} \subseteq S \times S$:

- $(\langle\alpha\rangle\varphi)^{\mathcal{M}} = \{u \mid \exists v.\ (u, v) \in \alpha^{\mathcal{M}}$ and $v \in \varphi^{\mathcal{M}}\}$
- $(\alpha;\beta)^{\mathcal{M}} = \{(u, v) \mid \exists w.\ (u, w) \in \alpha^{\mathcal{M}}$ and $(w, v) \in \beta^{\mathcal{M}}\}$
- $(\alpha \cup \beta)^{\mathcal{M}} = \alpha^{\mathcal{M}} \cup \beta^{\mathcal{M}}$
- $(\alpha^*)^{\mathcal{M}} =$ “reflexive transitive closure of $\alpha^{\mathcal{M}}$”
- $(?\varphi)^{\mathcal{M}} = \{(u, u) \mid u \in \varphi^{\mathcal{M}}\}$

Derived Formulas and Programs

- $\wedge, \rightarrow, \leftrightarrow,$ *true*, *false* are defined from $\neg, \vee$
- $[\alpha]\varphi \equiv \neg\langle\alpha\rangle\neg\varphi$ (*all execution of α lead to a state where φ*)
- **skip** $\equiv$?*true*
- **fail** $\equiv$?*false*
- **if** φ **then** α **else** β **fi** $\equiv (?\varphi; \alpha) \cup (?\neg\varphi; \beta)$
- **while** φ **do** α **od** $\equiv (?\varphi; \alpha)^*; ?\neg\varphi$
- Hoare triples: $\{\varphi\}\alpha\{\psi\} \equiv \varphi \rightarrow [\alpha]\psi$
- Weakest liberal precondition: wlp$(\alpha, \varphi) \equiv [\alpha]\varphi$
- Weakest precondition: wlp$(\alpha, \varphi) \equiv \langle\alpha\rangle\varphi$

Some Valid PDL Formulas

- $\langle \alpha \cup \beta \rangle \varphi \;\leftrightarrow\; \langle \alpha \rangle \varphi \vee \langle \beta \rangle \varphi$
- $[\alpha \cup \beta] \varphi \;\leftrightarrow\; [\alpha] \varphi \wedge [\beta] \varphi$
- $\langle \alpha ; \beta \rangle \varphi \;\leftrightarrow\; \langle \alpha \rangle \langle \beta \rangle \varphi$
- $[\alpha ; \beta] \varphi \;\leftrightarrow\; [\alpha][\beta] \varphi$
- $\langle ?\psi \rangle \varphi \;\leftrightarrow\; \psi \wedge \varphi$
- $[?\psi] \varphi \;\leftrightarrow\; \psi \rightarrow \varphi$
- $(\varphi \rightarrow [\alpha]\varphi) \;\rightarrow\; (\varphi \rightarrow [\alpha^*]\varphi)$

Meta-properties of PDL

PDL is not compact

- $\{\neg\varphi, \neg\langle\alpha\rangle\varphi, \neg\langle\alpha;\alpha\rangle\varphi, \neg\langle\alpha;\alpha;\alpha\rangle\varphi, \ldots\} \cup \{\neg\langle\alpha^*\rangle\varphi\}$

  is finitely satisfiable, but not satisfiable.

PDL is complete

- The exists a proof system $\vdash$ such that: if $\models \varphi$ then $\vdash \varphi$.

PDL Complexity

- PDL satisfiability is *deterministic exponential time complete*. (Regardless of allowing $\langle\ \rangle$, $[\ ]$ inside ?-tests.)

Deterministic PDL

A program α is deterministic if describes a partial function:

$$\alpha^{\mathcal{M}} \in S \rightharpoonup S$$

Deterministic while programs

- $\cup$, * appear only to abbreviate **if** and **while**

In deterministic PDL:

- $[\alpha]\varphi$ is partial correctness
- $\langle\alpha\rangle\varphi$ is total correctness
- $\langle\alpha\rangle\varphi \to [\alpha]\varphi$ is valid

Part III

First-order Dynamic Logic

First-Order States

Signature

A first-order (DL) signature Σ consists of

- a set F_Σ of function symbols
- a set P_Σ of predicate symbols
- a set V_Σ of program variables

Definition (First-Order DL State)

Let $\mathcal{D}$ be a domain.

$$\mathcal{I}(f) : \mathcal{D} \times \cdots \times \mathcal{D} \to \mathcal{D} \qquad \text{(for each } f \in F_\Sigma\text{)}$$

$$\mathcal{I}(p) \subseteq \mathcal{D} \times \cdots \times \mathcal{D} \qquad \text{(for each } p \in P_\Sigma\text{)}$$

$$\mathcal{I}(v) \in \mathcal{D} \qquad \text{(for each } v \in V_\Sigma\text{)}$$

Then $s = (\mathcal{D}, \mathcal{I})$ is a first-order DL state.
S is the set of all first-order states.

Kripke Model

Definition (Kripke Model)

Kripke Model $K = (S, \rho)$

- States $(\mathcal{D}, \mathcal{I}) \in S$
- Transition relation $\rho : \mathit{Program} \to 2^{S \times S}$

$$(s, s') \in \rho(\alpha)$$

iff.

one execution of α starting in state s leads to final sate s'

- ρ is the semantics of programs $\in \mathit{Program}$
- For now, we assume $\mathcal{D}$, $\mathcal{I}(F_\Sigma)$, $\mathcal{I}(P_\Sigma)$ identical all states of S.
  $\Rightarrow$ States vary only on program variables $\mathcal{I}(V_\Sigma)$.

First-order Dynamic Logic (DL)

Changes to PDL:

- Atomic programs have forms:
  - $v := t$ (deterministic assignment)
  - $v := *$ (non-deterministic assignment)
- Atomic formulas are of the forms:
  - $p(t_1, \ldots, t_n)$
  - $t_1 = t_2$
- If φ is a DL formula, then so are $\exists x.\varphi$, $\forall x.\varphi$
- φ appearing in $?\varphi$ must be a quantifier-free first-order formula

Some Valid DL Formulas

- $[v := *]\varphi(v) \;\leftrightarrow\; \forall x.\varphi(x)$
- $\langle v := *\rangle\varphi(v) \;\leftrightarrow\; \exists x.\varphi(x)$
- $\langle v := t\rangle\varphi \;\leftrightarrow\; \varphi[v/t]$
  ($\varphi[v/t]$ result of substituting v by t)
  weakest precondition reasoning
- $[v := t]\varphi \;\leftrightarrow\; \varphi[v/t]$

Meta-properties of (first-order) DL

DL is in-complete

- The exists no proof system $\vdash$ such that:

  if $\models \varphi$ then $\vdash \varphi$.

DL is relatively complete

- Let $\mathcal{A}$ be an arithmetical structure.
- Assume $T_{\mathcal{A}}$ to be all theorems of $\mathcal{A}$.
- The exists a proof system $\vdash$ such that:

  if $\mathcal{A} \models \varphi$ then $T_{\mathcal{A}} \vdash \varphi$.

Part IV

Symbolic Execution

Motivation

Traditional reasoning about programs goes backwards:

$$\begin{array}{ll} & \langle v_1 := t_1; v_2 := t_2 \rangle \varphi \\ \leftrightarrow & \langle v_1 := t_1 \rangle \langle v_2 := t_2 \rangle \varphi \\ \leftrightarrow & \langle v_1 := t_1 \rangle \varphi[v_2/t_2] \\ \leftrightarrow & (\varphi[v_2/t_2])[v_1/t_1] \\ \equiv & \varphi[v_1/t_1, v_2/t_2[v_1/t_1]] \end{array}$$

Symbolic Execution of Programs

Symbolic Execution (King, late 60s)

- Follow the natural control flow when analysing a program

Notation for Symbolic State Changes: "updates"

- Symbolic execution should "walk" through program in natural forward direction
- Need succinct representation of state changes, effected by each symbolic execution step
- Want to simplify effects of program execution early
- Want to apply state changes late (to *branching conditions* and *post condition*)

Explicit State Updates

Explicit Substitutions: "Updates"

Extend DL by explicit substitution modalities, called updates.

Definition (Syntax of Updates, Updated Terms/Formulas)

If v is program variable, t FOL term (of right type), t' any FOL term, and φ any DL formula, then

- $\{\mathrm{v} := t\}$ is an update
- $\{\mathrm{v} := t\}t'$ is DL term
- $\{\mathrm{v} := t\}\varphi$ is DL formula

Computing Effect of Updates (Automated)

Rewrite rules for update followed by ...

program variable $\begin{cases} \{\mathrm{x} := t\}\mathrm{x} & \leadsto & t \\ \{\mathrm{x} := t\}\mathrm{y} & \leadsto & \mathrm{y} \end{cases}$

logical variable $\{\mathrm{x} := t\}w \leadsto w$

complex term $\{\mathrm{x} := t\}f(t_1, \dots, t_n) \leadsto f(\{\mathrm{x} := t\}t_1, \dots, \{\mathrm{x} := t\}t_n)$

atomic formula $\{\mathrm{x} := t\}p(t_1, \dots, t_n) \leadsto p(\{\mathrm{x} := t\}t_1, \dots, \{\mathrm{x} := t\}t_n)$

FOL formula $\begin{cases} \{\mathrm{x} := t\}(\varphi \;\&\; \psi) \leadsto \{\mathrm{x} := t\}\varphi \;\&\; \{\mathrm{x} := t\}\psi \\ \dots \\ \{\mathrm{x} := t\}(\forall\,\tau\; y;\, \varphi) \leadsto \forall\,\tau\; y;\, (\{\mathrm{x} := t\}\varphi) \end{cases}$

program formula No rewrite rule for $\{\mathrm{x} := t\}\langle \mathtt{prog} \rangle\varphi$

Substitution delayed until *prog* symbolically executed

Assignment Rule Using Updates

Symbolic execution of assignment using updates

$$\text{assign}\ \frac{\Gamma \vdash \{\mathtt{x} := t\}\langle \mathit{rest}\rangle\varphi, \Delta}{\Gamma \vdash \langle \mathtt{x\ =\ }t\mathtt{;}\ \ \mathit{rest}\rangle\varphi, \Delta}$$

- Works as long as *t* is 'simple' (has no side effects)

Parallel Updates

How to apply updates on updates?

Example

Symbolic execution of

```
t=x; x=y; y=t;
```

yields:

```
{t := x}{x := y}{y := t}
```

Need to compose three sequential state changes into a single one: parallel updates

Parallel Updates Cont'd

Definition (Parallel Update)

A parallel update has the form $\{v_1 := r_1 || \cdots || v_n := r_n\}$, where each $\{v_i := r_i\}$ is simple update

- All r_i computed in old state before update is applied
- Updates of all program variables v_i executed simultaneously
- Upon conflict $v_i = v_j,\ r_i \neq r_j$ later update ($\max\{i, j\}$) wins

Definition (Parallelising Updates, Conflict Resolution)

$$\{v_1 := r_1\}\{v_2 := r_2\} \ = \ \{v_1 := r_1 || v_2 := \{v_1 := r_1\}r_2\}$$

$$\{v_1 := r_1 || \cdots || v_n := r_n\}\mathtt{x} \ = \ \begin{cases} \mathtt{x} & \text{if } \mathtt{x} \notin \{v_1, \ldots, v_n\} \\ r_k & \text{if } \mathtt{x} = v_k,\ \mathtt{x} \notin \{v_{k+1}, \ldots, v_n\} \end{cases}$$

Symbolic Execution with Updates (by Example)

$$\texttt{x} < \texttt{y} \vdash \texttt{x} < \texttt{y}$$

$$\vdots$$

$$\texttt{x} < \texttt{y} \vdash \{\texttt{x:=y} \parallel \texttt{y:=x}\}\langle\rangle\, \texttt{y} < \texttt{x}$$

$$\vdots$$

$$\texttt{x} < \texttt{y} \vdash \{\texttt{t:=x} \parallel \texttt{x:=y} \parallel \texttt{y:=x}\}\langle\rangle\, \texttt{y} < \texttt{x}$$

$$\vdots$$

$$\texttt{x} < \texttt{y} \vdash \{\texttt{t:=x} \parallel \texttt{x:=y}\}\{\texttt{y:=t}\}\langle\rangle\, \texttt{y} < \texttt{x}$$

$$\vdots$$

$$\texttt{x} < \texttt{y} \vdash \{\texttt{t:=x}\}\{\texttt{x:=y}\}\langle\texttt{y=t;}\rangle\, \texttt{y} < \texttt{x}$$

$$\vdots$$

$$\texttt{x} < \texttt{y} \vdash \{\texttt{t:=x}\}\langle\texttt{x=y; y=t;}\rangle\, \texttt{y} < \texttt{x}$$

$$\vdots$$

$$\vdash \texttt{x} < \texttt{y} \texttt{ -> } \langle\texttt{t=x; x=y; y=t;}\rangle\, \texttt{y} < \texttt{x}$$

Part V

Dynamic Logic at Scale

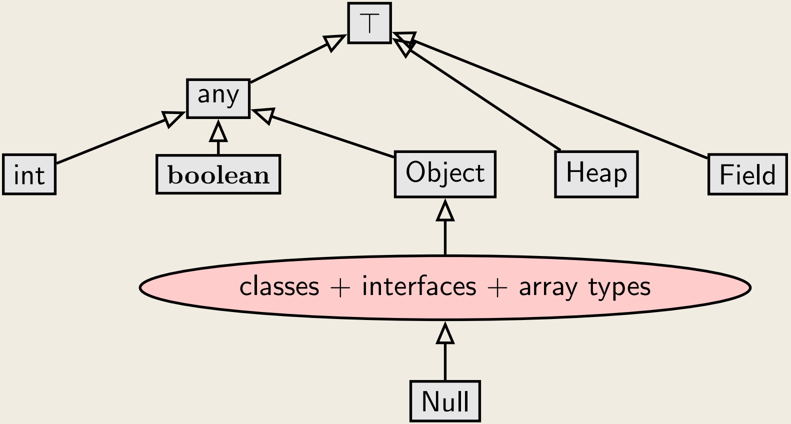
DL based Verification of a Real World Language

KeY verification approach and system, featuring:

- DL for full (sequential) Java
- Seqent Calculus for JavaDL
- Supporting specification language JML
- Translating JML + Java to JavaDL formulas
- KeY prover:
  - proof strategies for high automation
  - advanced GUI for proof interaction

Modelling Java in FOL: Fixing a Type Hierarchy

Signature based on Java's type hierarchy



Each interface and class in application and API becomes type with appropriate subtype relation

Java Features in Dynamic Logic: Complex Expressions

Complex expressions with side effects

- JAVA expressions may have side effects, due to method calls, increment/decrement operators, nested assignments
- FOL terms have no side effect on the state

Example (Complex expression with side effects in Java)

`int i = 0; if ((i=2)>= 2) i++;` value of `i` ?

Complex Expressions Cont'd

Decomposition of complex terms by symbolic execution

Follow the rules laid down in JAVA Language Specification

Local code transformations

$$\text{evalOrderIteratedAssgnmt}\ \frac{\Gamma \vdash \langle \texttt{y = t; x = y;}\ \omega\rangle\varphi, \Delta}{\Gamma \vdash \langle \texttt{x = y = t;}\ \omega\rangle\varphi, \Delta}\quad \texttt{t}\ \text{simple}$$

Temporary variables store result of evaluating subexpression

$$\text{ifEval}\ \frac{\Gamma \vdash \langle \mathbf{boolean}\ \texttt{v0; v0 = b;}\ \mathbf{if}\ \texttt{(v0) p;}\ \omega\rangle\varphi, \Delta}{\Gamma \vdash \langle \mathbf{if}\ \texttt{(b) p;}\ \omega\rangle\varphi, \Delta}\quad \text{b complex}$$

Java Features in Dynamic Logic: Abrupt Termination

Abrupt Termination: Exceptions and Jumps

Redirection of control flow via **return**, **break**, **continue**, exceptions

$$\langle \textbf{try}\ \{\texttt{p}\}\ \textbf{catch}(\texttt{T e})\ \{\texttt{q}\}\ \textbf{finally}\ \{\texttt{r}\}\ \omega\rangle\varphi$$

Rule tryThrow matches try–catch in pre-/postfix and active throw

$$\frac{\vdash\langle \textbf{if}\,(\texttt{e}\ \textbf{instanceof}\,\texttt{T})\{\textbf{try}\{\texttt{x=e};\texttt{q}\}\,\textbf{finally}\,\{\texttt{r}\}\}\textbf{else}\{\texttt{r};\textbf{throw}\ \texttt{e};\}\ \omega\rangle\varphi}{\vdash\ \langle \textbf{try}\ \{\ \textbf{throw}\ \texttt{e};\ \texttt{p}\}\ \textbf{catch}(\texttt{T x})\ \{\texttt{q}\}\ \textbf{finally}\ \{\texttt{r}\}\ \omega\rangle\varphi}$$

Field Update Assignment Rule

Changing the value of fields

How to (symbolically) execute assignment to field?

$$\frac{\Gamma, \mathrm{o} \neq \mathtt{\textbf{null}} \vdash \{\mathrm{o.f} := \mathrm{e}\}\langle\pi\ \ \omega\rangle\varphi, \Delta \qquad \Gamma, \mathrm{o} = \mathtt{\textbf{null}} \vdash \langle\pi\, \mathtt{\textbf{throw}}\ \mathtt{new\ NullPointerException();}\ \omega\rangle\varphi, \Delta}{\Gamma \vdash \langle\pi\, \mathrm{o.f} = \mathrm{e};\, \omega\rangle\varphi, \Delta}$$

π is the "inactive prefix", any number of opening try blocks: $(\mathbf{try}\{)^*$

Major Case Studies with KeY: TimSort

TimSort

Hybrid sorting algorithm (insertion sort + merge sort) optimised for partially sorted arrays (typical for real-world data).

Facts

- Designed by Tim Peters (for Python)
- Since Java 1.7 default algorithm for non-primitive arrays/collections

TimSort is used in

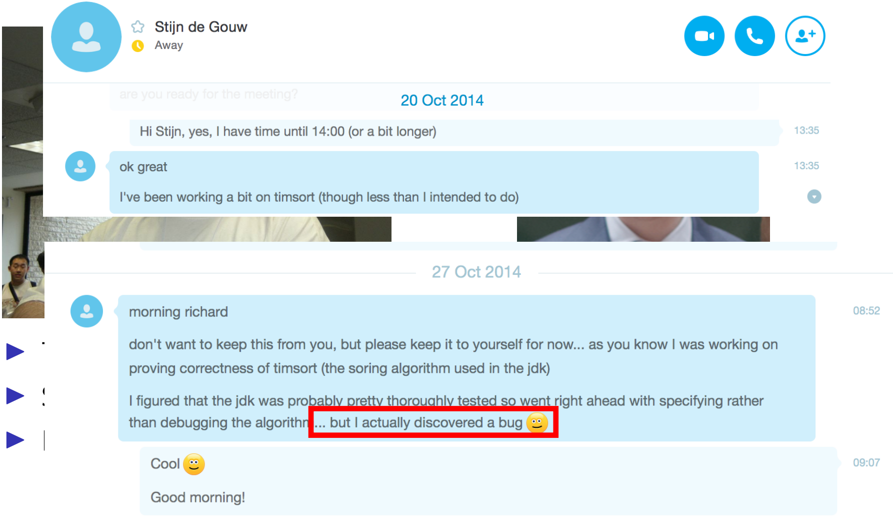
- Java (standard libraries OpenJDK, Oracle)
- Python (standard library)
- Android (standard library)
- ... and many more languages / frameworks!

TimSort: People

- Tim Peters
- Sorting Algorithm Designer
- Python Guru

- Stijn de Gouw
- Assistant Professor
- Formerly postman in the NL
- Interested in sorting for professional reasons

TimSort: People



are you ready for the meeting?

20 Oct 2014

Hi Stijn, yes, I have time until 14:00 (or a bit longer) 13:35

ok great 13:35

I've been working a bit on timsort (though less than I intended to do)

27 Oct 2014

morning richard 08:52

don't want to keep this from you, but please keep it to yourself for now... as you know I was working on proving correctness of timsort (the soring algorithm used in the jdk)

I figured that the jdk was probably pretty thoroughly tested so went right ahead with specifying rather than debugging the algorithm... but I actually discovered a bug 🙂

Cool 🙂 09:07

Good morning!

professional reasons

Major Case Studies with KeY

Found Bug in Java Libraries' main Sorting Method using KeY

- `java.util.Collections.sort` and `java.util.Arrays.sort` implement TimSort
- KeY verification of OpenJDK implementation revealed *bug*.
- Same bug present in Android SDK, Phyton library, Haskell library, ...

Verified Fix using KeY

- Fixing the implementation
- Verified absence of the bug in new version with KeY

Major Case Studies with KeY

Found Bug in Java Libraries' main Sort Method using KeY

- `java.util.Collections.so` `util.Arrays.sort` implement TimSort
- KeY verification evealed *bug*.
- Same b , Haskell library, …

Verified

- Fixing
- Verified bug in new version with KeY

Some researchers found an error in the logic of merge_collapse, explained here, and with corrected code shown in

…

It should be fixed anyway, and their suggested fix looks good to me.

Tim Peters via Python-Bugtracker

Major Case Studies with KeY

Found Bug in Java Libraries' main Sorti Method using KeY

- java.uti Collections.s util.Arrays.sort impleme
- KeY ve evealed *bug*.
- Same , Haskell library, ...

Verified

- Fixing
- Verified bug in new version n KeY

Some d an error in the
logic of m xplained here,
and with co own in
It should be fixe heir sug-
gested fix looks goo
Tim Peters via acker

Congratulations to Stijn de Gouw et al.
for finding and fixing a bug in TimSort
using formal methods!
Joshua Bloch via Twitter

KeY target languages

- Java
- ABS (distributed objects with asynchronous method calls)
- Solidity (smart contract language)

Remark:

If you thought $?\varphi$ is a purely theoretical concept:

Solidity command

require(e)

means *exactly*

?e

Part VI

Constant vs. Dynamic Domains

Kripke Model (recap)

Definition (Kripke Model)

Kripke Model $K = (S, \rho)$

- States $(\mathcal{D}, \mathcal{I}) \in S$
- Transition relation $\rho : \mathit{Program} \rightarrow (S \times S)$

$$(s, s') \in \rho(\alpha)$$

iff.

one execution of α starting in state s leads to final sate s'

- So far, we assumed $\mathcal{D}$, $\mathcal{I}(F_\Sigma)$, $\mathcal{I}(P_\Sigma)$ identical all states of S.
  $\Rightarrow$ States vary only on program variables $\mathcal{I}(V_\Sigma)$.
  $\Rightarrow$ Constant domain assumption.

Challenge the Constant Domains

Should the following be valid for all programs α?

$$(\forall x. \langle \alpha \rangle \varphi(x)) \overset{?}{\leftrightarrow} \langle \alpha \rangle \forall x. \varphi(x)$$

- When could this be a problem?
- What if α creates additional resources we can quantify over?
- E.g., object creation?
- Can we extend $\mathcal{D}$?
- What happens to $\mathcal{I}(F_\Sigma)$, $\mathcal{I}(P_\Sigma)$ on the new elements?

(External Slides)

Part VII

dL: Differential Dynamic Logic

Hybrid Systems

Mathematical model of systems combining:

- discrete dynamics
- continuous dynamics

Differential Equations

Example:

- $x' = v, v' = a$

Describes mutual dependency how *variables* x, v, a change over continuous time.

Why did I say *variable* instead of *function*?

Good fit to modal/temporal/dynamic logic.

"Talk" about flexible variables rather than functions (over states or time).

We identify "differential equation" and "differential equation system":

- $\begin{pmatrix} x' \\ v' \end{pmatrix} = \begin{pmatrix} v \\ a \end{pmatrix}$

Towards Differential Dynamic Logic

Continuous programs

$x' = f(x) \;\&\; Q$

with

- differential equation $x' = f(x)$
- evolution domain constraint Q

Intuitive meaning:
Variables 'follow' differential equation for any duration while satisfying Q.

Example

$x' = v, v' = a, cl' = 1 \;\&\; cl \leq eps$

Meaning:

- x, v, a follow Newton dynamics
- cl moves exactly with time (cl is a clock)
- system evolves *at most* until clock reaches *eps*

Semantics of Continuous Programs

Real Valued Program Variables

A state $s \in S$ is a mapping from program variables to real numbers:

$$s : V_\Sigma \to \mathbb{R}$$

Semantics of Continuous Programs (simplified)

State v is reachable from state u by $x_1' = e_1, \ldots, x_n' = e_n \ \& \ Q$ iff there is a *solution* σ and duration r s.t.:

- $\sigma : [0, r] \to S$
- $\sigma(0) = u$, $\sigma(r) = v$
- At each time $\tau \in [0, r]$:
  - $\dfrac{\mathrm{d}\sigma(t)(x_i)}{\mathrm{d}t}(\tau) = \sigma(\tau)(e^{\mathcal{M}})$
  - $\sigma(\tau) \in Q^{\mathcal{M}}$

Hybrid Programs

Hybrid Programs

$\alpha, \beta ::= x := t \mid x := * \mid ?\varphi \mid x' = f(x) \;\&\; Q \mid \alpha \cup \beta \mid \alpha ; \beta \mid \alpha^*$

Differential Dynamic Logic (dL)

Changes to first-order DL:

- Programs are hybrid programs.
- Add $t_1 \leq t_2$ to atomic formulas.

Examples

Safety of Controlled Plant

$$[(\mathit{ctrl}; \mathit{plant})^*]\mathit{safety\text{-}cond}$$

To Brake or Not To Break [Platzer 3.4.1]

$$[((?\varphi_A; a := A \cup ?\neg\varphi_A; a := B);$$
$$cl := 0;$$
$$\{x' = v, v' = a, cl' = 1 \;\&\; v \geq 0 \land cl \leq eps\})^*]\ \psi_{safe}$$

Literature

D. Kozen, J. Tiuryn
Logics of Programs
Handbook of Theoretical Computer Science, 1990.

W. Ahrendt, B. Beckert, R. Bubel, R. Hähnle, P. Schmitt, M. Ulbrich, editors.
Deductive Software Verification - The KeY Book
Vol 10001 of LNCS, Springer, 2016.

A. Platzer.
Logical Foundations of Cyber-Physical Systems
Springer, 2018.

W. Ahrendt, F. de Boer, I. Grabe
Abstract Object Creation in Dynamic Logic
— To Be or Not To Be Created
FM 2009, LNCS, Springer, 2009.