A Theorem Proving Approach to Analysis of Secure Information Flow

A Paper by *Ádám Darvas*, *Rainer Hänle*, and *David Sands*

Georg Jung

jung@cis.ksu.edu

Department of Computing and Information Sciences, Kansas State University

Overview

Overview

- Introduction to Dynamic Logic

Overview

- Introduction to Dynamic Logic
- The KeY Theorem Prover

Overview

- Introduction to Dynamic Logic
- The KeY Theorem Prover
- Application of the Framework

Overview

- Introduction to Dynamic Logic
- The KeY Theorem Prover
- Application of the Framework

On Wednesday I hope to be able to present

Overview

- Introduction to Dynamic Logic
- The KeY Theorem Prover
- Application of the Framework

On Wednesday I hope to be able to present

- The theorem prover running “life”

Overview

- Introduction to Dynamic Logic
- The KeY Theorem Prover
- Application of the Framework

On Wednesday I hope to be able to present

- The theorem prover running “life”
- A meaningful example

Structure of the Dynamic Logic

Basics of Dynamic Logic

Dynamic Logic (**DL**) (David Harel, Dexter Kozen, Jerzy Tiuryn, 1984) was designed to reason about programs.

Basics of Dynamic Logic

Dynamic Logic (**DL**) (David Harel, Dexter Kozen, Jerzy Tiuryn, 1984) was designed to reason about programs. **DL** combines

Basics of Dynamic Logic

Dynamic Logic (**DL**) (David Harel, Dexter Kozen, Jerzy Tiuryn, 1984) was designed to reason about programs. **DL** combines

- First Order Predicate Logic

Basics of Dynamic Logic

Dynamic Logic (**DL**) (David Harel, Dexter Kozen, Jerzy Tiuryn, 1984) was designed to reason about programs. **DL** combines

- First Order Predicate Logic
- Modal Logic

Basics of Dynamic Logic

Dynamic Logic (**DL**) (David Harel, Dexter Kozen, Jerzy Tiuryn, 1984) was designed to reason about programs. **DL** combines

- First Order Predicate Logic
- Modal Logic
- Algebra of Linear Events

Basics of Dynamic Logic

Dynamic Logic (**DL**) (David Harel, Dexter Kozen, Jerzy Tiuryn, 1984) was designed to reason about programs. **DL** combines

- First Order Predicate Logic
- Modal Logic
- Algebra of Linear Events

Usually, the truth value of a formula ϕ is determined by a valuation of the free variables. The valuation though is *immutable*.

Basics of Dynamic Logic

Dynamic Logic (**DL**) (David Harel, Dexter Kozen, Jerzy Tiuryn, 1984) was designed to reason about programs.
DL combines

- First Order Predicate Logic
- Modal Logic
- Algebra of Linear Events

Usually, the truth value of a formula ϕ is determined by a valuation of the free variables. The valuation though is *immutable*.
DL instead features syntactic constructs to explicitly change a valuation. These constructs are referred to as *programs*.

Basics of Dynamic Logic

Dynamic Logic (**DL**) (David Harel, Dexter Kozen, Jerzy Tiuryn, 1984) was designed to reason about programs. **DL** combines

- First Order Predicate Logic
- Modal Logic
- Algebra of Linear Events

Usually, the truth value of a formula ϕ is determined by a valuation of the free variables. The valuation though is *immutable*.

DL instead features syntactic constructs to explicitly change a valuation. These constructs are referred to as *programs*.

In this presentation I describe a special **DL** called JAVA CARD DL, which is introduced by the referred paper.

Modalities of JAVA CARD DL

DL offers two modalities for every program `p`:

Modalities of JAVA CARD DL

DL offers two modalities for every program p:

- $\langle \mathtt{p} \rangle$ refers to the state (if p terminates) that is reached by running p.
  The *program formula* $\langle \mathtt{p} \rangle \phi$ hence expresses that p terminates in a state in which ϕ holds.

Modalities of JAVA CARD DL

DL offers two modalities for every program $\mathtt{p}$:

- $\langle \mathtt{p} \rangle$ refers to the state (if $\mathtt{p}$ terminates) that is reached by running $\mathtt{p}$.
  The *program formula* $\langle \mathtt{p} \rangle \phi$ hence expresses that $\mathtt{p}$ terminates in a state in which ϕ holds.
- Its dual $[\mathtt{p}]\phi \equiv \neg\langle \mathtt{p} \rangle \neg\phi$ expresses that $\langle \mathtt{p} \rangle$ either diverges or terminates in a state in which ϕ holds.

Modalities of JAVA CARD DL

DL offers two modalities for every program $\mathtt{p}$:

- $\langle \mathtt{p} \rangle$ refers to the state (if $\mathtt{p}$ terminates) that is reached by running $\mathtt{p}$.
  The *program formula* $\langle \mathtt{p} \rangle \phi$ hence expresses that $\mathtt{p}$ terminates in a state in which ϕ holds.
- Its dual $[\mathtt{p}]\phi \equiv \neg\langle \mathtt{p} \rangle \neg\phi$ expresses that $\langle \mathtt{p} \rangle$ either diverges or terminates in a state in which ϕ holds.

The $[.]$ modality is not yet implemented in the presented analysis system's backbone, the *KeY* Theorem Prover.

Modalities of JAVA CARD DL

DL offers two modalities for every program $\mathtt{p}$:

- $\langle \mathtt{p} \rangle$ refers to the state (if $\mathtt{p}$ terminates) that is reached by running $\mathtt{p}$.
  The *program formula* $\langle \mathtt{p} \rangle \phi$ hence expresses that $\mathtt{p}$ terminates in a state in which ϕ holds.
- Its dual $[\mathtt{p}]\phi \equiv \neg\langle \mathtt{p} \rangle \neg\phi$ expresses that $\langle \mathtt{p} \rangle$ either diverges or terminates in a state in which ϕ holds.

The $[.]$ modality is not yet implemented in the presented analysis system's backbone, the *KeY* Theorem Prover.

- For technical reasons there is also the *update* $\{\mathtt{loc} := val\}$ which has the same semantics as $\langle \mathtt{loc} = \mathtt{val}; \rangle$, only that the evaluation of val cannot have side effects.

Variable Types in DL

Variable Types in DL

- *program variables* are simply the locations used in the programs. Program variables can be updated, their interpretation varies with the program state.

Variable Types in DL

- *program variables* are simply the locations used in the programs. Program variables can be updated, their interpretation varies with the program state.
- *logic variables* the quantified variables. Logic variables cannot be updated.

Variable Types in DL

- *program variables* are simply the locations used in the programs. Program variables can be updated, their interpretation varies with the program state.
- *logic variables* the quantified variables. Logic variables cannot be updated.
- *metavariables* are introduced in the *skolemization* process.

Variable Types in DL

- *program variables* are simply the locations used in the programs. Program variables can be updated, their interpretation varies with the program state.
- *logic variables* the quantified variables. Logic variables cannot be updated.
- *metavariables* are introduced in the *skolemization* process.

Note: quantification over program variables is illegal in JAVA CARD DL.

Variable Types in DL

- *program variables* are simply the locations used in the programs. Program variables can be updated, their interpretation varies with the program state.
- *logic variables* the quantified variables. Logic variables cannot be updated.
- *metavariables* are introduced in the *skolemization* process.

Note: quantification over program variables is illegal in JAVA CARD DL.
To express the quantification $\forall \mathtt{x}.\langle \mathtt{p}(\mathtt{x})\rangle \psi(\mathtt{x})$,

Variable Types in DL

- *program variables* are simply the locations used in the programs. Program variables can be updated, their interpretation varies with the program state.
- *logic variables* the quantified variables. Logic variables cannot be updated.
- *metavariables* are introduced in the *skolemization* process.

Note: quantification over program variables is illegal in JAVA CARD DL.
To express the quantification $\forall \mathtt{x}.\langle \mathtt{p(x)}\rangle\psi(\mathtt{x})$, we hence have to write

$$\langle \mathtt{int\ x;}\rangle(\forall l : int.\{\mathtt{x} := l\}\langle \mathtt{p(x)}\rangle\psi(l, \mathtt{x}))$$

Comparison to Hoare Logic

The **DL** formula $\phi \rightarrow \langle \mathtt{p} \rangle \psi$ is similar to the *total correctness* Hoare triple $\{\phi\}\mathtt{p}\{\psi\}$ (C. A. R. Hoare. An axiomatic basis for computer programming. ACM 1969)

Comparison to Hoare Logic

The **DL** formula $\phi \rightarrow \langle \mathrm{p} \rangle \psi$ is similar to the *total correctness* Hoare triple $\{\phi\}\mathrm{p}\{\psi\}$ (C. A. R. Hoare. An axiomatic basis for computer programming. ACM 1969)

In Hoare Logic, quantifiers are only allowed within the assertions, quantification over program variables is not possible.

Comparison to Hoare Logic

The **DL** formula $\phi \rightarrow \langle \mathrm{p} \rangle \psi$ is similar to the *total correctness* Hoare triple $\{\phi\}\mathrm{p}\{\psi\}$ (C. A. R. Hoare. An axiomatic basis for computer programming. ACM 1969)

In Hoare Logic, quantifiers are only allowed within the assertions, quantification over program variables is not possible.

Hoare Logic does not provide for characterization of termination behavior.

Example 1

Consider two variables `h` and `l`. We want to express that no information flows from `h` to `l`:

Example 1

Consider two variables `h` and `l`. We want to express that no information flows from `h` to `l`:

$$\langle \texttt{int l;} \quad \texttt{int h;} \rangle (\forall x : int . \exists r : int . \forall y : int . \{\texttt{l} := x\}\{\texttt{h} := y\}\langle \texttt{p} \rangle r = \texttt{l})$$

Example 1

Consider two variables `h` and `l`. We want to express that no information flows from `h` to `l`:

$$\langle \texttt{int l}; \quad \texttt{int h}; \rangle (\forall x : int. \exists r : int. \forall y : int. \{\texttt{l} := x\}\{\texttt{h} := y\}\langle \texttt{p} \rangle r = \texttt{l})$$

Remember that the update has pure technical reasons. For the presentation we can write:

$$\forall \texttt{l}. \exists r. \forall \texttt{h}. \langle \texttt{p} \rangle r = \texttt{l}$$

Example 2

In addition to the independence of the result value r from any validation of the variable `h` we want to express that no information about `h` leaks from the termination behavior of $\langle \mathtt{p} \rangle$.

Example 2

In addition to the independence of the result value r from any validation of the variable $\mathtt{h}$ we want to express that no information about $\mathtt{h}$ leaks from the termination behavior of $\langle \mathtt{p} \rangle$.

$$\forall \mathtt{l}. \exists r. \forall \mathtt{h}. \langle \mathtt{p} \rangle r = \mathtt{l}$$

Example 2

In addition to the independence of the result value r from any validation of the variable $\mathtt{h}$ we want to express that no information about $\mathtt{h}$ leaks from the termination behavior of $\langle \mathtt{p} \rangle$.

$$\forall \mathtt{l}.(\exists \mathtt{h}.\langle \mathsf{p} \rangle true \rightarrow \exists r.\forall \mathtt{h}.\langle \mathtt{p} \rangle r = \mathtt{l})$$

Example 2

In addition to the independence of the result value r from any validation of the variable $\mathtt{h}$ we want to express that no information about $\mathtt{h}$ leaks from the termination behavior of $\langle \mathtt{p} \rangle$.

$$\forall \mathtt{l}.(\exists \mathtt{h}.\langle \mathsf{p} \rangle true \rightarrow \exists r.\forall \mathtt{h}.\langle \mathtt{p} \rangle r = \mathtt{l})$$

Note that this formula contains an implicit use of the $[.]$ modality and can therefore not be handled by the present implementation of KeY.

The *KeY* Theorem Prover

The KeY Theorem Prover

- Available at **`www.key-project.org`**

The KeY Theorem Prover

- Available at **`www.key-project.org`**
- User guided proof search

The KeY Theorem Prover

- Available at **www.key-project.org**
- User guided proof search
- Extensible with user defined *taclets*
  - Combination of *primitive rules* and *tactics*.
  - Can easily be added to the system.

The KeY Theorem Prover

- Available at **`www.key-project.org`**
- User guided proof search
- Extensible with user defined *taclets*
  - Combination of *primitive rules* and *tactics*.
  - Can easily be added to the system.
- Proof *goals* are implicitly negated. If all goals can be completed, then the security is proven, if goals remain open, then the program is insecure.

The KeY Theorem Prover

- Available at **`www.key-project.org`**
- User guided proof search
- Extensible with user defined *taclets*
  - Combination of *primitive rules* and *tactics*.
  - Can easily be added to the system.
- Proof *goals* are implicitly negated. If all goals can be completed, then the security is proven, if goals remain open, then the program is insecure.
- Existentially quantified variables (which become universally quantified through implicit negation) are replaced by *meta variables*, so that they can be instantiated later (*delayed instantiation*).

Application of the Framework

A Toy Example

To prove the formula $\forall \mathtt{l}.\exists r.\forall \mathtt{h}.\langle \mathtt{p}\rangle r = \mathtt{l}$ it has to be negated:

A Toy Example

To prove the formula $\forall \mathtt{l}.\exists r.\forall \mathtt{h}.\langle \mathtt{p}\rangle r = \mathtt{l}$ it has to be negated:

$$\exists \mathtt{l}.\forall r.\exists \mathtt{h}.\langle \mathtt{p}\rangle r \neq \mathtt{l}$$

A Toy Example

To prove the formula $\forall \mathtt{l}.\exists r.\forall \mathtt{h}.\langle \mathtt{p}\rangle r = \mathtt{l}$ it has to be negated:

$$\exists \mathtt{l}.\forall r.\exists \mathtt{h}.\langle \mathtt{p}\rangle r \neq \mathtt{l}$$

The paper claims that the program "$\mathtt{l} = \mathtt{h};$" can be proven insecure in 14 steps without user interaction. I was not yet able to reproduce the proof and the paper does not give any further information but the goal which fails:

$$\exists r.\forall \mathtt{h}.r = \mathtt{h}$$

More Realistic

The following example is considered in the paper:

```
class Account {
  private int balance;
  public boolean extraService;

  private void writeBalance (int amount) {
    if (amount >= 10000) extraService = true;
      else extraService = false;
    balance = amount; }
...
```

More Realistic

```
...
  private int readBalance () {
    return balance; }

  public boolean readExtra () {
    return extraService; }
}
```

More Realistic

We formalize the goal:

More Realistic

We formalize the goal:

$\langle \texttt{Account o} = \texttt{new Account}(); \texttt{int amount}; \texttt{boolean result}; \rangle$

More Realistic

We formalize the goal:

$\langle \texttt{Account o} = \texttt{new Account}(); \texttt{int amount}; \texttt{boolean result};\rangle$

$\forall e : boolean.\exists r : boolean.\forall a : int.$

More Realistic

We formalize the goal:

$\langle \texttt{Account o = new Account ();\ int amount;\ boolean result;} \rangle$

$\forall e : boolean . \exists r : boolean . \forall a : int .$
$\{\texttt{o.extraService} := e\}\{\texttt{amount} := a\}$

More Realistic

We formalize the goal:

$$\langle \texttt{Account o = new Account (); int amount; boolean result;} \rangle$$

$$\forall e : boolean . \exists r : boolean . \forall a : int .$$
$$\{\texttt{o.extraService} := e\}\{\texttt{amount} := a\}$$
$$\langle \texttt{o.writeBalance(amount); result = o.readExtra();} \rangle$$

More Realistic

We formalize the goal:

$\langle \texttt{Account o} = \texttt{new Account}(); \texttt{ int amount}; \texttt{ boolean result};\rangle$

$\forall e : boolean. \exists r : boolean. \forall a : int.$
$\{\texttt{o.extraService} := e\}\{\texttt{amount} := a\}$
$\langle \texttt{o.writeBalance}(\texttt{amount}); \texttt{ result} = \texttt{o.readExtra}();\rangle$
$r = \texttt{result}$

Exceptions

The following example is considered in the paper:

```
l = true;
try {
  if (h) trow new Exception ();
  l = false;
}
catch (Exception e) {}
```

Exceptions

Again, we formalize the goal:

Exceptions

Again, we formalize the goal:

$\langle$`boolean l; boolean h;`$\rangle$

Exceptions

Again, we formalize the goal:

$\langle$`boolean l;` `boolean h;`$\rangle$

$\forall x : boolean. \exists r : boolean. \forall a : boolean.$

Exceptions

Again, we formalize the goal:

$\langle \texttt{boolean l; boolean h;} \rangle$

$\forall x : boolean . \exists r : boolean . \forall a : boolean .$

$\{\texttt{l} := x\}\{\texttt{h} := y\}$

Exceptions

Again, we formalize the goal:

$\langle$`boolean l; boolean h;`$\rangle$

$\forall x : boolean.\exists r : boolean.\forall a : boolean.$

$\{\mathtt{l} := x\}\{\mathtt{h} := y\}$

$\langle$`p`$\rangle$

Exceptions

Again, we formalize the goal:

$\langle \texttt{boolean l; boolean h;} \rangle$

$\forall x : boolean. \exists r : boolean. \forall a : boolean.$

$\{\texttt{l} := x\}\{\texttt{h} := y\}$

$\langle \texttt{p} \rangle$

$r = \texttt{l}$