Dynamic Logic for Practical Program Verification

Set 2

Wolfgang Ahrendt

Chalmers University of Technology, Gothenburg, Sweden

VTSA Summer School, Luxembourg, 2024

Part I

Dynamic Logic at Scale: The KeYsystem

KeY

KeY is an approach and tool for the

- Formal specification
- Deductive verification

of

- Software *source code*

using

- *Dynamic Logic*

KeY

KeY is an approach and tool for the

- Formal specification
- Deductive verification

of

- Software *source code*

using

- *Dynamic Logic*

Versions of KeY support verification of

- sequential Java
- ABS (executable modelling language for distributed objects)
- Solidity (Smart Contracts, *work in progress*)

KeY Project Partners

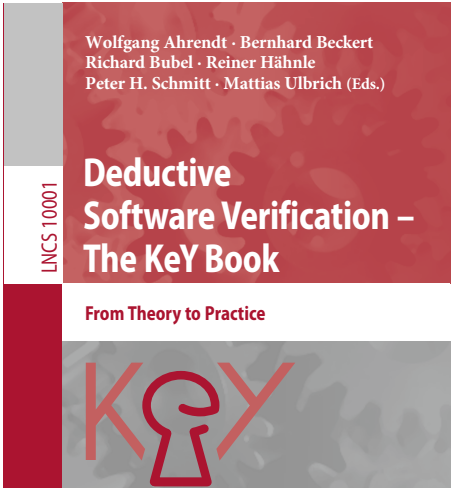
Karlsruhe Institute of Technology
Bernhard Beckert, Mattias Ulbrich, Peter H. Schmitt

Technical University Darmstadt
Reiner Hähnle, Richard Bubel

Chalmers University of Technology
Wolfgang Ahrendt

+ post-docs, PhD students, students, collaborators, alumni

The KeY Book


Wolfgang Ahrendt · Bernhard Beckert
Richard Bubel · Reiner Hähnle
Peter H. Schmitt · Mattias Ulbrich (Eds.)
LNCS 10001
Deductive
Software Verification –
The KeY Book
From Theory to Practice
KeY

KeY in 30 seconds

- Dynamic logic as program logic
- Calculus follows symbolic execution paradigm
- Sequent calculus
- Prover is interactive + automated
- most elaborate KeY instance: KeY-Java
  - Java as target language
  - Supports specification language JML (Java Modeling Language)

JML example

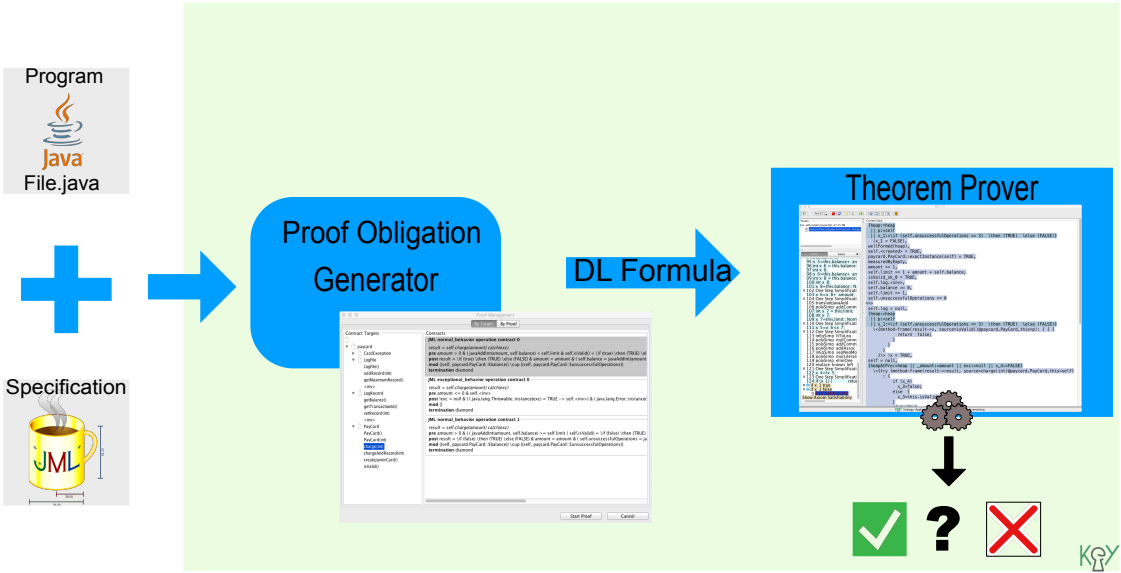
```
/*@ public normal_behavior
  @ requires a != null;
  @ ensures (\forall int j; j >= 0 && j < a.length;
  @                         \result >= a[j]);
  @ ensures a.length > 0 ==>
  @         (\exists int j; j >= 0 && j < a.length;
  @                         \result == a[j]);
  @*/
public static /*@ pure @*/ int max(int[] a) {
    int max = a[0], i = 1;
    while ( i < a.length ) {
      if ( a[i] > max ) max = a[i];
      ++i;
    }
    return max;}
```

JML to Dynamic Logic

Major components of KeY-Java

- Proof Obligation Generator
  - input: Java files containing JML specs
  - output: proof obligations in Dynamic Logic (DL) for Java
- KeY Prover
  - constructing proofs in sequent calculus for DL
  - designed for interplay of interaction and automated strategies

Typical KeY Workflow



Dynamic Logic for (almost) full Java

KeY supports full sequential Java, with some limitations:

- No generics
- No I/O
- No dynamic class loading or reflection
- API method calls: need either JML contract or implementation

Dynamic Logic for (almost) full Java

KeY supports full sequential Java, with some limitations:

- No generics
- No I/O
- No dynamic class loading or reflection
- API method calls: need either JML contract or implementation

- Recently added: support for floating-point numbers/arithemic

Major Case Study with KeY: Timsort

Timsort

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

Major Case Study 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)
- In Java libraries: default algorithm for non-primitive arrays/collections

Major Case Study 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)
- In Java libraries: default algorithm for non-primitive arrays/collections

Timsort is used in standard libraries of:

- Java
- Python
- Android
- Haskell [... and many more languages / frameworks]

Timsort: People

- Tim Peters

Timsort: People

- Tim Peters
- Sorting Algorithm Designer

Timsort: People

- Tim Peters
- Sorting Algorithm Designer
- Python Guru

Timsort: People

- Tim Peters
- Sorting Algorithm Designer
- Python Guru

- Stijn de Gouw

Timsort: People

- Tim Peters
- Sorting Algorithm Designer
- Python Guru

- Stijn de Gouw
- Postman in the NL

Timsort: People

- Tim Peters
- Sorting Algorithm Designer
- Python Guru

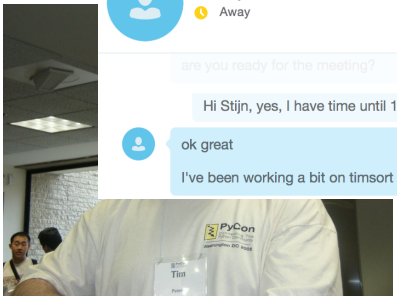
- Stijn de Gouw
- Postman in the NL
- Interested in sorting for professional reasons

Timsort: People

- Tim Peters
- Sorting Algorithm Designer
- Python Guru

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

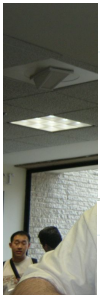
Timsort: People

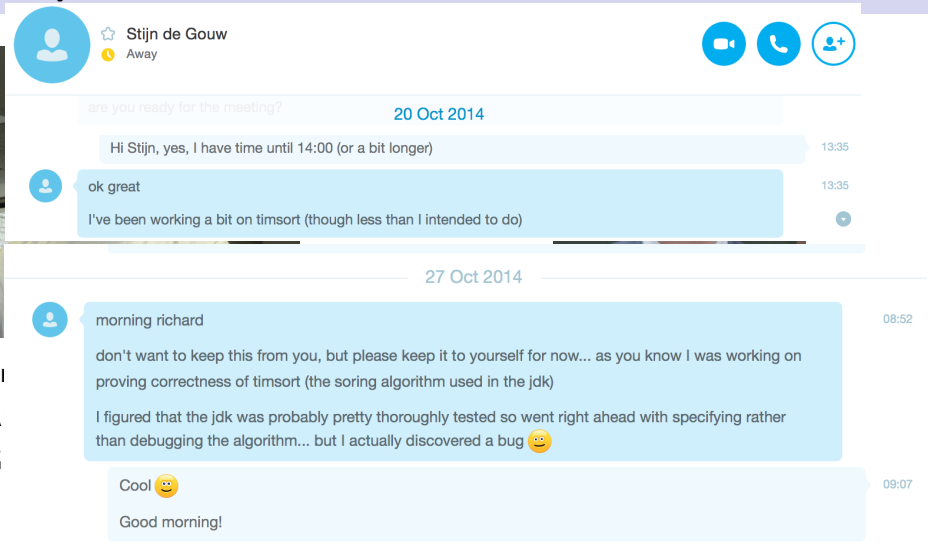


- Tim Peters
- Sorting Algorithm Designer
- Python Guru

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

Timsort: People

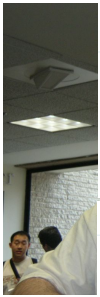




- Tim Peter
- Sorting A
- Python G

Assistant Professor

Timsort: People



- Tim Peter
- Sorting A
- Python G

Stijn de Gouw
Away

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!

Major Case Study 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*.

Major Case Study 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, Oracle's JDK, Python library, and ... Haskell library

Major Case Study 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, Oracle's JDK, Python library, and ... Haskell library

Verified Fix using KeY

- Fixing the (OpenJDK) implementation
- Verified new version with KeY (no `IndexOutOfBoundsException`)

Major Case Study with KeY

Found Bug in Java Libraries' main Sorting Method ... KeY

- `java.util.Collections.sort` and `java.u... sort` implement Timsort
- KeY verification of OpenJDK i...
- Same bug present in A... v, and ... Haskell library

Verified Fix using KeY

- Fixing the (OpenJDK...
- Verified new version w... `ArrayIndexOutOfBoundsException`)

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 Study with KeY

Found Bug in Java Libraries' main Sorting Method ... KeY

- `java.util.Collection...sort` and `java...sort` implement Timsort
- KeY verification of O...
- Same bug present in ... ..., and ... Haskell library

Verified Fix using KeY

- Fixing the (OpenJDK...
- Verified new version w... `...tion`)

Some ... an error in the logic of ... explained here, and with co... ...own in ... ...eir suggested fix looks goo... ...

It should be fixed ... ...

Tim Peters via ...

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

Joshua Bloch via Twitter

...acker

Fixed and Verified the Implementation

Verification with KeY

- Adding support for bitwise operations to KeY
- 460 lines of specification vs. 928 lines of code
- Fixed version formally verified (absence of exceptions)
- Whole project 2.5 person month

Aftermath

- Java community choose sub-optimal fix (increasing stack size)

Aftermath

- Java community choose sub-optimal fix (increasing stack size)
- Python community adopted KeYfix

Aftermath

- Java community choose sub-optimal fix (increasing stack size)
- Python community adopted KeYfix
- Bug affected
  - Java
  - Android
  - Python
  - Apache: Lucene, Hadoop, Spark++
  - Go, D, Haskell

Language	Min. array length req. to trigger error
Android	65.536 (2^{16})
Java	67.108.864 (2^{26})
Python	562.949.953.421.312 (2^{49})

Aftermath

- Java community choose sub-optimal fix (increasing stack size)
- Python community adopted KeYfix
- Bug affected
  - Java
  - Android
  - Python
  - Apache: Lucene, Hadoop, Spark++
  - Go, D, Haskell

Language	Min. array length req. to trigger error
Android	65.536 (2^{16})
Java	67.108.864 (2^{26})
Python	562.949.953.421.312 (2^{49})

- blog with $>$3 million page views
- top news on ycombinator, reddit, Hacker News etc.

References

- Stijn de Gouw, Jurriaan Rot, Frank S. de Boer, Richard Bubel, Reiner Hähnle
  OpenJDK's Java.utils.Collection.sort() Is Broken: The Good, the Bad and the Worst Case
  CAV 2015

Part II

KeY Prover Intro: First-Order Sequent Calculus

Motivation for Introducing First-Order Logic

We verify JAVA programs using First-Order Dynamic Logic

First-order Java DL combines

- First-Order Logic (FOL)
- JAVA programs

First-Order Formulas

Formulas

- each atomic formula is a formula
- with φ and ψ formulas, x a variable, and τ a type, the following are also formulas:
  - $\neg\varphi$ *("not φ")*
  - $\varphi \wedge \psi$ *("φ and ψ")*
  - $\varphi \vee \psi$ *("φ or ψ")*
  - $\varphi \rightarrow \psi$ *("φ implies ψ")*
  - $\varphi \leftrightarrow \psi$ *("φ is equivalent to ψ")*

First-Order Formulas

Formulas

- each atomic formula is a formula
- with φ and ψ formulas, x a variable, and τ a type, the following are also formulas:
  - $\neg\varphi$ *("not φ")*
  - $\varphi \wedge \psi$ *("φ and ψ")*
  - $\varphi \vee \psi$ *("φ or ψ")*
  - $\varphi \rightarrow \psi$ *("φ implies ψ")*
  - $\varphi \leftrightarrow \psi$ *("φ is equivalent to ψ")*
  - $\forall\ \tau\ x;\ \varphi$ *("for all x of type τ holds φ")*
  - $\exists\ \tau\ x;\ \varphi$ *("there exists an x of type τ such that φ")*

First-Order Formulas

Formulas

- each atomic formula is a formula
- with φ and ψ formulas, x a variable, and τ a type, the following are also formulas:
  - $\neg\varphi$ *("not φ")*
  - $\varphi \wedge \psi$ *("φ and ψ")*
  - $\varphi \vee \psi$ *("φ or ψ")*
  - $\varphi \rightarrow \psi$ *("φ implies ψ")*
  - $\varphi \leftrightarrow \psi$ *("φ is equivalent to ψ")*
  - $\forall\ \tau\ x;\ \varphi$ *("for all x of type τ holds φ")*
  - $\exists\ \tau\ x;\ \varphi$ *("there exists an x of type τ such that φ")*

In $\forall\ \tau\ x;\ \varphi$ and $\exists\ \tau\ x;\ \varphi$ the variable x is 'bound' (i.e., 'not free').

First-Order Formulas

Formulas

- each atomic formula is a formula
- with φ and ψ formulas, x a variable, and τ a type, the following are also formulas:
  - $\neg\varphi$ *("not φ")*
  - $\varphi \wedge \psi$ *("φ and ψ")*
  - $\varphi \vee \psi$ *("φ or ψ")*
  - $\varphi \rightarrow \psi$ *("φ implies ψ")*
  - $\varphi \leftrightarrow \psi$ *("φ is equivalent to ψ")*
  - $\forall\ \tau\ x;\ \varphi$ *("for all x of type τ holds φ")*
  - $\exists\ \tau\ x;\ \varphi$ *("there exists an x of type τ such that φ")*

In $\forall\ \tau\ x;\ \varphi$ and $\exists\ \tau\ x;\ \varphi$ the variable x is 'bound' (i.e., 'not free').

Formulas with no free variable are 'closed'.

Reasoning by Syntactic Transformation

Prove validity of φ by **syntactic** transformation of φ

Reasoning by Syntactic Transformation

Prove validity of φ by **syntactic** transformation of φ

Sequent Calculus based on notion of sequent:

$$\underbrace{\psi_1, \ldots, \psi_m}_{\text{antecedent}} \quad \Longrightarrow \quad \underbrace{\varphi_1, \ldots, \varphi_n}_{\text{succedent}}$$

Reasoning by Syntactic Transformation

Prove validity of φ by **syntactic** transformation of φ

Sequent Calculus based on notion of sequent:

$$\underbrace{\psi_1, \ldots, \psi_m}_{\text{antecedent}} \quad \Longrightarrow \quad \underbrace{\varphi_1, \ldots, \varphi_n}_{\text{succedent}}$$

has same meaning as

$$(\psi_1 \wedge \cdots \wedge \psi_m) \quad \rightarrow \quad (\varphi_1 \vee \cdots \vee \varphi_n)$$

Reasoning by Syntactic Transformation

Prove validity of φ by **syntactic** transformation of φ

Sequent Calculus based on notion of sequent:

$$\underbrace{\psi_1, \ldots, \psi_m}_{\text{antecedent}} \quad \Longrightarrow \quad \underbrace{\varphi_1, \ldots, \varphi_n}_{\text{succedent}}$$

has same meaning as

$$(\psi_1 \wedge \cdots \wedge \psi_m) \quad \rightarrow \quad (\varphi_1 \vee \cdots \vee \varphi_n)$$

which (for closed formulas ψ_i, φ_i) is equivalent to

$$\{\psi_1, \ldots, \psi_m\} \quad \models \quad \varphi_1 \vee \cdots \vee \varphi_n$$

Notation for Sequents

$$\psi_1, \ldots, \psi_m \quad \Longrightarrow \quad \varphi_1, \ldots, \varphi_n$$

Consider antecedent/succedent as sets of formulas (possibly empty)

Notation for Sequents

$$\psi_1, \ldots, \psi_m \quad \Longrightarrow \quad \varphi_1, \ldots, \varphi_n$$

Consider antecedent/succedent as sets of formulas (possibly empty)

Schema Variables

$\varphi, \psi, \ldots$ match formulas.
$\Gamma, \Delta, \ldots$ match sets of formulas.
Characterize infinitely many sequents with single schematic sequent, e.g.,

$$\Gamma \quad \Longrightarrow \quad \varphi \wedge \psi, \Delta$$

matches any sequent with occurrence of conjunction in succedent.

Here, we call $\varphi \wedge \psi$ main formula and Γ, Δ side formulas of sequent

Sequent Calculus Rules

Write syntactic transformation schema for sequents,
reflecting semantics of connectives

$$\text{RuleName}\ \frac{\overbrace{\Gamma_1 \Longrightarrow \Delta_1 \quad \cdots \quad \Gamma_r \Longrightarrow \Delta_r}^{\text{premisses}}}{\underbrace{\Gamma \Longrightarrow \Delta}_{\text{conclusion}}}$$

Meaning: For proving the conclusion, it suffices to prove all premisses.

Sequent Calculus Rules

Write syntactic transformation schema for sequents,
reflecting semantics of connectives

$$\text{RuleName}\ \frac{\overbrace{\Gamma_1 \Longrightarrow \Delta_1 \quad \cdots \quad \Gamma_r \Longrightarrow \Delta_r}^{\text{premisses}}}{\underbrace{\Gamma \Longrightarrow \Delta}_{\text{conclusion}}}$$

Meaning: For proving the conclusion, it suffices to prove all premisses.

Example

$$\text{andRight}\ \frac{\Gamma \Longrightarrow \varphi, \Delta \qquad \Gamma \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \varphi \wedge \psi, \Delta}$$

Sequent Calculus Rules

Write syntactic transformation schema for sequents,
reflecting semantics of connectives

$$\text{RuleName}\ \frac{\overbrace{\Gamma_1 \Longrightarrow \Delta_1 \quad \cdots \quad \Gamma_r \Longrightarrow \Delta_r}^{\text{premisses}}}{\underbrace{\Gamma \Longrightarrow \Delta}_{\text{conclusion}}}$$

Meaning: For proving the conclusion, it suffices to prove all premisses.

Example

$$\text{andRight}\ \frac{\Gamma \Longrightarrow \varphi, \Delta \qquad \Gamma \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \varphi \wedge \psi, \Delta}$$

Admissible to have no premisses (then the rule is called 'axiom').

A rule is sound (correct) iff the validity of all premisses implies the validity of the conclusion.

‘Propositional’ Sequent Calculus Rules

$$\text{close}\ \frac{}{\Gamma, \varphi \Longrightarrow \varphi, \Delta}$$

‘Propositional’ Sequent Calculus Rules

$$\text{close}\ \frac{}{\Gamma, \varphi \Longrightarrow \varphi, \Delta} \qquad \text{true}\ \frac{}{\Gamma \Longrightarrow \mathrm{true}, \Delta}$$

‘Propositional’ Sequent Calculus Rules

$$\text{close}\ \frac{}{\Gamma, \varphi \Longrightarrow \varphi, \Delta} \qquad \text{true}\ \frac{}{\Gamma \Longrightarrow \mathrm{true}, \Delta} \qquad \text{false}\ \frac{}{\Gamma, \mathrm{false} \Longrightarrow \Delta}$$

‘Propositional’ Sequent Calculus Rules

close $\dfrac{}{\Gamma, \varphi \Longrightarrow \varphi, \Delta}$ true $\dfrac{}{\Gamma \Longrightarrow \mathrm{true}, \Delta}$ false $\dfrac{}{\Gamma, \mathrm{false} \Longrightarrow \Delta}$

	left side (antecedent)	right side (succedent)
not	$\dfrac{\Gamma \Longrightarrow \varphi, \Delta}{\Gamma, \neg\varphi \Longrightarrow \Delta}$	$\dfrac{\Gamma, \varphi \Longrightarrow \Delta}{\Gamma \Longrightarrow \neg\varphi, \Delta}$

‘Propositional’ Sequent Calculus Rules

close $\dfrac{}{\Gamma, \varphi \Longrightarrow \varphi, \Delta}$ true $\dfrac{}{\Gamma \Longrightarrow \mathrm{true}, \Delta}$ false $\dfrac{}{\Gamma, \mathrm{false} \Longrightarrow \Delta}$

	left side (antecedent)	right side (succedent)
not	$\dfrac{\Gamma \Longrightarrow \varphi, \Delta}{\Gamma, \neg\varphi \Longrightarrow \Delta}$	$\dfrac{\Gamma, \varphi \Longrightarrow \Delta}{\Gamma \Longrightarrow \neg\varphi, \Delta}$
and	$\dfrac{\Gamma, \varphi, \psi \Longrightarrow \Delta}{\Gamma, \varphi \wedge \psi \Longrightarrow \Delta}$	$\dfrac{\Gamma \Longrightarrow \varphi, \Delta \qquad \Gamma \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \varphi \wedge \psi, \Delta}$

‘Propositional’ Sequent Calculus Rules

close $\dfrac{}{\Gamma, \varphi \Longrightarrow \varphi, \Delta}$ true $\dfrac{}{\Gamma \Longrightarrow \mathrm{true}, \Delta}$ false $\dfrac{}{\Gamma, \mathrm{false} \Longrightarrow \Delta}$

	left side (antecedent)	right side (succedent)
not	$\dfrac{\Gamma \Longrightarrow \varphi, \Delta}{\Gamma, \neg\varphi \Longrightarrow \Delta}$	$\dfrac{\Gamma, \varphi \Longrightarrow \Delta}{\Gamma \Longrightarrow \neg\varphi, \Delta}$
and	$\dfrac{\Gamma, \varphi,\ \psi \Longrightarrow \Delta}{\Gamma, \varphi \wedge \psi \Longrightarrow \Delta}$	$\dfrac{\Gamma \Longrightarrow \varphi, \Delta \qquad \Gamma \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \varphi \wedge \psi, \Delta}$
or	$\dfrac{\Gamma, \varphi \Longrightarrow \Delta \qquad \Gamma, \psi \Longrightarrow \Delta}{\Gamma, \varphi \vee \psi \Longrightarrow \Delta}$	$\dfrac{\Gamma \Longrightarrow \varphi,\ \psi, \Delta}{\Gamma \Longrightarrow \varphi \vee \psi, \Delta}$

‘Propositional’ Sequent Calculus Rules

close $\dfrac{}{\Gamma, \varphi \Longrightarrow \varphi, \Delta}$ true $\dfrac{}{\Gamma \Longrightarrow \mathrm{true}, \Delta}$ false $\dfrac{}{\Gamma, \mathrm{false} \Longrightarrow \Delta}$

	left side (antecedent)	right side (succedent)
not	$\dfrac{\Gamma \Longrightarrow \varphi, \Delta}{\Gamma, \neg\varphi \Longrightarrow \Delta}$	$\dfrac{\Gamma, \varphi \Longrightarrow \Delta}{\Gamma \Longrightarrow \neg\varphi, \Delta}$
and	$\dfrac{\Gamma, \varphi,\ \psi \Longrightarrow \Delta}{\Gamma, \varphi \wedge \psi \Longrightarrow \Delta}$	$\dfrac{\Gamma \Longrightarrow \varphi, \Delta \qquad \Gamma \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \varphi \wedge \psi, \Delta}$
or	$\dfrac{\Gamma, \varphi \Longrightarrow \Delta \qquad \Gamma, \psi \Longrightarrow \Delta}{\Gamma, \varphi \vee \psi \Longrightarrow \Delta}$	$\dfrac{\Gamma \Longrightarrow \varphi,\ \psi, \Delta}{\Gamma \Longrightarrow \varphi \vee \psi, \Delta}$
imp	$\dfrac{\Gamma \Longrightarrow \varphi, \Delta \qquad \Gamma, \psi \Longrightarrow \Delta}{\Gamma, \varphi \to \psi \Longrightarrow \Delta}$	

‘Propositional’ Sequent Calculus Rules

close $\dfrac{}{\Gamma, \varphi \Longrightarrow \varphi, \Delta}$ true $\dfrac{}{\Gamma \Longrightarrow \mathrm{true}, \Delta}$ false $\dfrac{}{\Gamma, \mathrm{false} \Longrightarrow \Delta}$

	left side (antecedent)	right side (succedent)
not	$\dfrac{\Gamma \Longrightarrow \varphi, \Delta}{\Gamma, \neg\varphi \Longrightarrow \Delta}$	$\dfrac{\Gamma, \varphi \Longrightarrow \Delta}{\Gamma \Longrightarrow \neg\varphi, \Delta}$
and	$\dfrac{\Gamma, \varphi,\ \psi \Longrightarrow \Delta}{\Gamma, \varphi \wedge \psi \Longrightarrow \Delta}$	$\dfrac{\Gamma \Longrightarrow \varphi, \Delta \qquad \Gamma \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \varphi \wedge \psi, \Delta}$
or	$\dfrac{\Gamma, \varphi \Longrightarrow \Delta \qquad \Gamma, \psi \Longrightarrow \Delta}{\Gamma, \varphi \vee \psi \Longrightarrow \Delta}$	$\dfrac{\Gamma \Longrightarrow \varphi,\ \psi, \Delta}{\Gamma \Longrightarrow \varphi \vee \psi, \Delta}$
imp	$\dfrac{\Gamma \Longrightarrow \varphi, \Delta \qquad \Gamma, \psi \Longrightarrow \Delta}{\Gamma, \varphi \to \psi \Longrightarrow \Delta}$	$\dfrac{\Gamma, \varphi \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \varphi \to \psi, \Delta}$

Sequent Calculus Proofs

Goal to prove: $\mathcal{G} = \quad \psi_1, \ldots, \psi_m \implies \varphi_1, \ldots, \varphi_n$

- find rule $\mathcal{R}$ whose conclusion matches $\mathcal{G}$
- instantiate $\mathcal{R}$ such that its conclusion is identical to $\mathcal{G}$
- apply that instantiation to all premisses of $\mathcal{R}$, resulting in new goals $\mathcal{G}_1, \ldots, \mathcal{G}_r$
- recursively find proofs for $\mathcal{G}_1, \ldots, \mathcal{G}_r$
- tree structure with goal as root
- close proof branch when applying rule without premiss

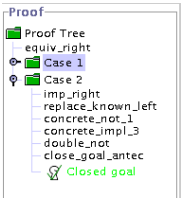
Sequent Calculus Proofs

Goal to prove: $\mathcal{G} = \quad \psi_1, \ldots, \psi_m \implies \varphi_1, \ldots, \varphi_n$

- find rule $\mathcal{R}$ whose conclusion matches $\mathcal{G}$
- instantiate $\mathcal{R}$ such that its conclusion is identical to $\mathcal{G}$
- apply that instantiation to all premisses of $\mathcal{R}$, resulting in new goals $\mathcal{G}_1, \ldots, \mathcal{G}_r$
- recursively find proofs for $\mathcal{G}_1, \ldots, \mathcal{G}_r$
- tree structure with goal as root
- close proof branch when applying rule without premiss

Goal-directed proof search

- Paper proofs: root at bottom, grow upwards
- KeY tool proofs: root at top, grow downwards



A Simple Proof

$$\dfrac{\dfrac{\dfrac{}{}\qquad\dfrac{}{}}{}}{\Longrightarrow (p \wedge (p \to q)) \to q}$$

A Simple Proof

$$\dfrac{\dfrac{\dfrac{}{}\qquad\dfrac{}{}}{p \wedge (p \to q) \Longrightarrow q}}{\Longrightarrow (p \wedge (p \to q)) \to q}$$

A Simple Proof

$$\dfrac{\dfrac{\dfrac{\dfrac{}{}\quad\dfrac{}{}}{p,\,(p \to q) \Longrightarrow q}}{p \wedge (p \to q) \Longrightarrow q}}{\Longrightarrow (p \wedge (p \to q)) \to q}$$

A Simple Proof

$$\dfrac{\dfrac{\dfrac{\dfrac{}{p \Longrightarrow p,\ q}\qquad\dfrac{}{p,\ q \Longrightarrow q}}{p,\ (p \to q) \Longrightarrow q}}{p \wedge (p \to q) \Longrightarrow q}}{\Longrightarrow (p \wedge (p \to q)) \to q}$$

A Simple Proof

$$\dfrac{\dfrac{\dfrac{\textsc{Close}\dfrac{*}{p \Longrightarrow p,\ q} \qquad \dfrac{*}{p,\ q \Longrightarrow q}\textsc{Close}}{p,\ (p \to q) \Longrightarrow q}}{p \wedge (p \to q) \Longrightarrow q}}{\Longrightarrow (p \wedge (p \to q)) \to q}$$

A Simple Proof

$$\dfrac{\dfrac{\dfrac{\dfrac{}{p \Longrightarrow p,\, q}\,{*}\,\textsc{Close} \qquad \dfrac{}{p,\, q \Longrightarrow q}\,{*}\,\textsc{Close}}{p,\, (p \to q) \Longrightarrow q}}{p \wedge (p \to q) \Longrightarrow q}}{\Longrightarrow (p \wedge (p \to q)) \to q}$$

A proof is closed iff all its branches are closed

Demo

`prop.key`

Proving Validity of First-Order Formulas

Proving a universally quantified formula

Claim: $\forall\, \tau\, x;\ \varphi$ is true

How is such a claim proved in Mathematics?

All even numbers are divisible by 2	$\forall\, \mathbf{int}\, x;\ (\mathsf{even}(x) \rightarrow \mathsf{divByTwo}(x))$
Let c be an arbitrary number	Declare "unused" constant $\mathbf{int}$ c
The even number c is divisible by 2	Prove $\mathsf{even}(c) \rightarrow \mathsf{divByTwo}(c)$

Proving Validity of First-Order Formulas

Proving a universally quantified formula

Claim: $\forall\, \tau\, x;\ \varphi$ is true

How is such a claim proved in Mathematics?

All even numbers are divisible by 2	$\forall\, \mathbf{int}\, x;\ (\mathsf{even}(x) \rightarrow \mathsf{divByTwo}(x))$
Let c be an arbitrary number	Declare "unused" constant $\mathbf{int}$ c
The even number c is divisible by 2	Prove $\mathsf{even}(c) \rightarrow \mathsf{divByTwo}(c)$

Sequent rule $\forall$-right

$$\mathsf{forallRight}\ \frac{\Gamma \Longrightarrow [x/c]\,\varphi, \Delta}{\Gamma \Longrightarrow \forall\, \tau\, x;\ \varphi, \Delta}$$

- $[x/c]\,\varphi$ is result of replacing each occurrence of x in φ with c
- c **new** constant of type τ

Proving Validity of First-Order Formulas Cont'd

Proving an existentially quantified formula

Claim: $\exists\, \tau\, x;\ \varphi$ is true

How is such a claim proved in Mathematics?

There is at least one prime number	$\exists\, \mathbf{int}\, x;\ \mathrm{prime}(x)$
Provide any "witness", say, 7	Use variable-free term $\mathbf{int}\ 7$
7 is a prime number	Prove prime(7)

Proving Validity of First-Order Formulas Cont'd

Proving an existentially quantified formula

Claim: $\exists\, \tau\, x;\ \varphi$ is true

How is such a claim proved in Mathematics?

There is at least one prime number	$\exists\, \mathbf{int}\, x;\ \mathrm{prime}(x)$
Provide any "witness", say, 7	Use variable-free term $\mathbf{int}\ 7$
7 is a prime number	Prove prime(7)

Sequent rule $\exists$-right

$$\text{existsRight}\ \frac{\Gamma \Longrightarrow [x/t]\,\varphi,\ \exists\, \tau\, x;\ \varphi, \Delta}{\Gamma \Longrightarrow \exists\, \tau\, x;\ \varphi, \Delta}$$

- t any variable-free term of type τ
- We might need other instances besides t! Keep $\exists\, \tau\, x;\ \varphi$

Proving Validity of First-Order Formulas Cont'd

Using a universally quantified formula

We assume $\forall\, \tau\, x;\ \varphi$ is true.

How is such a fact used in a Mathematical proof?

We know that all primes are odd	$\forall\, \mathbf{int}\, x;\ (\mathsf{prime}(x) \rightarrow \mathsf{odd}(x))$
In particular, this holds for 17	Use variable-free term $\mathbf{int}\ 17$
We know: if 17 is prime it is odd	$\mathsf{prime}(17) \rightarrow \mathsf{odd}(17)$

Proving Validity of First-Order Formulas Cont'd

Using a universally quantified formula

We assume $\forall\, \tau\, x;\ \varphi$ is true.

How is such a fact used in a Mathematical proof?

We know that all primes are odd	$\forall\, \mathbf{int}\, x;\ (\mathsf{prime}(x) \to \mathsf{odd}(x))$
In particular, this holds for 17	Use variable-free term $\mathbf{int}\ 17$
We know: if 17 is prime it is odd	$\mathsf{prime}(17) \to \mathsf{odd}(17)$

Sequent rule $\forall$-left

$$\mathsf{forallLeft}\ \frac{\Gamma, \forall\, \tau\, x;\ \varphi,\ [x/t]\,\varphi \Longrightarrow \Delta}{\Gamma, \forall\, \tau\, x;\ \varphi \Longrightarrow \Delta}$$

- t any variable-free term of type τ
- We might need other instances besides t! Keep $\forall\, \tau\, x;\ \varphi$

Proving Validity of First-Order Formulas Cont'd

Using an existentially quantified formula

We assume $\exists\, \tau\, x;\ \varphi$ is true

How is such a fact used in a Mathematical proof?

We know such an element exists. Let's give that element it a new name.

Proving Validity of First-Order Formulas Cont'd

Using an existentially quantified formula

We assume $\exists\, \tau\, x;\ \varphi$ is true

How is such a fact used in a Mathematical proof?

We know such an element exists. Let's give that element it a new name.

Sequent rule $\exists$-left

$$\text{existsLeft}\ \frac{\Gamma, [x/c]\,\varphi \Longrightarrow \Delta}{\Gamma, \exists\, \tau\, x;\ \varphi \Longrightarrow \Delta}$$

- c **new** constant of type τ

Proving Validity of First-Order Formulas Cont'd

Using an equation between terms

We assume $t = t'$ is true

How is such a fact used in a Mathematical proof?

$x = y-1 \Longrightarrow (x+1)/y = 1$

Use $x = y-1$ to modify $(x+1)/y$:
Replace x in succedent with right-hand side of antecedent

Proving Validity of First-Order Formulas Cont'd

Using an equation between terms

We assume $t = t'$ is true

How is such a fact used in a Mathematical proof?

$x = y-1 \Longrightarrow (x+1)/y = 1$

Use $x = y-1$ to modify $(x+1)/y$:
Replace x in succedent with right-hand side of antecedent

$x = y-1 \Longrightarrow (y-1+1)/y = 1$

Proving Validity of First-Order Formulas Cont'd

Using an equation between terms

We assume $t = t'$ is true

How is such a fact used in a Mathematical proof?

$x = y-1 \Longrightarrow (x+1)/y = 1$

Use $x = y-1$ to modify $(x+1)/y$:
Replace x in succedent with right-hand side of antecedent

$x = y-1 \Longrightarrow (y-1+1)/y = 1$

Sequent rule =-left

$$\text{applyEqL}\ \frac{\Gamma, t = t', [t/t']\,\varphi \Longrightarrow \Delta}{\Gamma, t = t', \varphi \Longrightarrow \Delta} \qquad \text{applyEqR}\ \frac{\Gamma, t = t' \Longrightarrow [t/t']\,\varphi, \Delta}{\Gamma, t = t' \Longrightarrow \varphi, \Delta}$$

- Always replace left- with right-hand side (use eqSymm if necessary)
- t,t' variable-free terms of the same type

Proving Validity of First-Order Formulas Cont'd

Closing a subgoal in a proof

- We derived a sequent that is trivially valid

$$\text{close}\ \frac{}{\Gamma, \varphi \Longrightarrow \varphi, \Delta} \qquad \text{true}\ \frac{}{\Gamma \Longrightarrow \mathrm{true}, \Delta} \qquad \text{false}\ \frac{}{\Gamma, \mathrm{false} \Longrightarrow \Delta}$$

- We derived an equation that is trivially valid

$$\text{eqClose}\ \frac{}{\Gamma \Longrightarrow t = t, \Delta}$$

Sequent Calculus for FOL at One Glance

	left side, antecedent	right side, succedent
$\forall$	$\dfrac{\Gamma, \forall\, \tau\, x;\ \varphi,\ [x/t']\,\varphi \Longrightarrow \Delta}{\Gamma, \forall\, \tau\, x;\ \varphi \Longrightarrow \Delta}$	$\dfrac{\Gamma \Longrightarrow [x/c]\,\varphi, \Delta}{\Gamma \Longrightarrow \forall\, \tau\, x;\ \varphi, \Delta}$
$\exists$	$\dfrac{\Gamma, [x/c]\,\varphi \Longrightarrow \Delta}{\Gamma, \exists\, \tau\, x;\ \varphi \Longrightarrow \Delta}$	$\dfrac{\Gamma \Longrightarrow [x/t']\,\varphi,\ \exists\, \tau\, x;\ \varphi, \Delta}{\Gamma \Longrightarrow \exists\, \tau\, x;\ \varphi, \Delta}$
$=$	$\dfrac{\Gamma, t = t' \Longrightarrow [t/t']\,\varphi, \Delta}{\Gamma, t = t' \Longrightarrow \varphi, \Delta}$ (+ application rule on left side)	$\dfrac{}{\Gamma \Longrightarrow t = t, \Delta}$

- $[t/t']\,\varphi$ is result of replacing each occurrence of t in φ with t'
- t,t' variable-free terms of type τ
- c **new** constant of type τ (occurs not on current proof branch)
- Equations can be reversed by commutativity

Recap: 'Propositional' Sequent Calculus Rules

main	left side (antecedent)	right side (succedent)
not	$\dfrac{\Gamma \Longrightarrow \varphi, \Delta}{\Gamma, \neg\varphi \Longrightarrow \Delta}$	$\dfrac{\Gamma, \varphi \Longrightarrow \Delta}{\Gamma \Longrightarrow \neg\varphi, \Delta}$
and	$\dfrac{\Gamma, \varphi, \psi \Longrightarrow \Delta}{\Gamma, \varphi \wedge \psi \Longrightarrow \Delta}$	$\dfrac{\Gamma \Longrightarrow \varphi, \Delta \qquad \Gamma \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \varphi \wedge \psi, \Delta}$
or	$\dfrac{\Gamma, \varphi \Longrightarrow \Delta \qquad \Gamma, \psi \Longrightarrow \Delta}{\Gamma, \varphi \vee \psi \Longrightarrow \Delta}$	$\dfrac{\Gamma \Longrightarrow \varphi, \psi, \Delta}{\Gamma \Longrightarrow \varphi \vee \psi, \Delta}$
imp	$\dfrac{\Gamma \Longrightarrow \varphi, \Delta \qquad \Gamma, \psi \Longrightarrow \Delta}{\Gamma, \varphi \to \psi \Longrightarrow \Delta}$	$\dfrac{\Gamma, \varphi \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \varphi \to \psi, \Delta}$

close $\dfrac{}{\Gamma, \varphi \Longrightarrow \varphi, \Delta}$ true $\dfrac{}{\Gamma \Longrightarrow \mathrm{true}, \Delta}$ false $\dfrac{}{\Gamma, \mathrm{false} \Longrightarrow \Delta}$

Proving Validity of First-Order Formulas Cont'd

Example (A simple theorem about binary relations)

$$\exists x;\ \forall y;\ p(x,y) \Longrightarrow \forall y;\ \exists x;\ p(x,y)$$

"Untyped" logic: let type of x and y be *any*

Proving Validity of First-Order Formulas Cont'd

Example (A simple theorem about binary relations)

$$\frac{\forall y;\ p(c, y) \Longrightarrow \forall y;\ \exists x;\ p(x, y)}{\exists x;\ \forall y;\ p(x, y) \Longrightarrow \forall y;\ \exists x;\ p(x, y)}$$

$\exists$-left: substitute new constant c for x

Proving Validity of First-Order Formulas Cont'd

Example (A simple theorem about binary relations)

$$\dfrac{\dfrac{\forall y;\ p(c,y) \Longrightarrow \exists x;\ p(x,d)}{\forall y;\ p(c,y) \Longrightarrow \forall y;\ \exists x;\ p(x,y)}}{\exists x;\ \forall y;\ p(x,y) \Longrightarrow \forall y;\ \exists x;\ p(x,y)}$$

$\forall$-right: substitute new constant d for y

Proving Validity of First-Order Formulas Cont'd

Example (A simple theorem about binary relations)

$$\dfrac{\dfrac{\dfrac{p(c,d),\ \forall y;\ p(c,y) \Longrightarrow \exists x;\ p(x,d)}{\forall y;\ p(c,y) \Longrightarrow \exists x;\ p(x,d)}}{\forall y;\ p(c,y) \Longrightarrow \forall y;\ \exists x;\ p(x,y)}}{\exists x;\ \forall y;\ p(x,y) \Longrightarrow \forall y;\ \exists x;\ p(x,y)}$$

$\forall$-left: free to substitute arbitrary term (of right type) for y, choose d

Proving Validity of First-Order Formulas Cont'd

Example (A simple theorem about binary relations)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{p(c,d),\ \forall y;\ p(c,y) \Longrightarrow p(c,d),\ \exists x;\ p(x,d)}{p(c,d),\ \forall y;\ p(c,y) \Longrightarrow \exists x;\ p(x,d)}}{\forall y;\ p(c,y) \Longrightarrow \exists x;\ p(x,d)}}{\forall y;\ p(c,y) \Longrightarrow \forall y;\ \exists x;\ p(x,y)}}{\exists x;\ \forall y;\ p(x,y) \Longrightarrow \forall y;\ \exists x;\ p(x,y)}$$

$\exists$-right: free to substitute arbitrary term (of right type) for x, choose c

Proving Validity of First-Order Formulas Cont'd

Example (A simple theorem about binary relations)

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{p(c,d),\ \forall y;\ p(c,y) \Longrightarrow p(c,d),\ \exists x;\ p(x,d)}\,*
}{p(c,d),\ \forall y;\ p(c,y) \Longrightarrow \exists x;\ p(x,d)}
}{\forall y;\ p(c,y) \Longrightarrow \exists x;\ p(x,d)}
}{\forall y;\ p(c,y) \Longrightarrow \forall y;\ \exists x;\ p(x,y)}
}{\exists x;\ \forall y;\ p(x,y) \Longrightarrow \forall y;\ \exists x;\ p(x,y)}
$$

Close

Proving Validity of First-Order Formulas Cont'd

Example (A simple theorem about binary relations)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\forall x;\ \exists y;\ p(x,y) \Longrightarrow \forall y;\ \exists x;\ p(x,y)}\ *}{\forall y;\ p(c,y) \Longrightarrow \forall y;\ \exists x;\ p(x,y)}}{\forall y;\ p(c,y) \Longrightarrow \exists x;\ p(x,d)}}{p(c,d),\ \forall y;\ p(c,y) \Longrightarrow \exists x;\ p(x,d)}}{p(c,d),\ \forall y;\ p(c,y) \Longrightarrow \exists x;\ p(c,d),\ \exists x;\ p(x,d)}$$

Demo

`relSimple.key`

Proving Validity of First-Order Formulas Cont'd

Using an existentially quantified formula

Proving Validity of First-Order Formulas Cont'd

Using an existentially quantified formula

Let x, y denote integer constants, both are not zero.

$$\neg(x = 0), \neg(y = 0) \qquad \Longrightarrow$$

Proving Validity of First-Order Formulas Cont'd

Using an existentially quantified formula

Let x, y denote integer constants, both are not zero. We know further that x divides y.

$$\neg(x = 0), \neg(y = 0), \exists\, \texttt{int}\ k;\ y = k * x \Longrightarrow$$

Proving Validity of First-Order Formulas Cont'd

Using an existentially quantified formula

Let x, y denote integer constants, both are not zero. We know further that x divides y.
Show: $(y/x) * x = y$
(equation not always true in integer division, e.g., $y = 1, x = 2$)

$$\neg(x = 0), \neg(y = 0), \exists\, \texttt{int}\ k;\ y = k * x \Longrightarrow (y/x) * x = y$$

Proving Validity of First-Order Formulas Cont'd

Using an existentially quantified formula

Let x, y denote integer constants, both are not zero. We know further that x divides y.
Show: $(y/x) * x = y$
(equation not always true in integer division, e.g., $y = 1, x = 2$)
Proof: We know x divides y, i.e. there exists a k such that $y = k * x$. Let now c denote such a k.

$$\frac{\neg(x = 0), \neg(y = 0), y = c * x \Longrightarrow (y/x) * x = y}{\neg(x = 0), \neg(y = 0), \exists\, \texttt{int}\ k;\ y = k * x \Longrightarrow (y/x) * x = y}$$

Proving Validity of First-Order Formulas Cont'd

Using an existentially quantified formula

Let x, y denote integer constants, both are not zero. We know further that x divides y.
Show: $(y/x) * x = y$
(equation not always true in integer division, e.g., $y = 1, x = 2$)
Proof: We know x divides y, i.e. there exists a k such that $y = k * x$. Let now c denote such a k. Hence we can replace y by $c * x$ on the right side.

$$\frac{\frac{\neg(x = 0), \neg(y = 0), y = c * x \Longrightarrow ((c * x)/x) * x = y}{\neg(x = 0), \neg(y = 0), y = c * x \Longrightarrow (y/x) * x = y}}{\neg(x = 0), \neg(y = 0), \exists\, \texttt{int}\ k;\ y = k * x \Longrightarrow (y/x) * x = y}$$

Proving Validity of First-Order Formulas Cont'd

Using an existentially quantified formula

Let x, y denote integer constants, both are not zero. We know further that x divides y.
Show: $(y/x) * x = y$
(equation not always true in integer division, e.g., $y = 1, x = 2$)
Proof: We know x divides y, i.e. there exists a k such that $y = k * x$. Let now c denote such a k. Hence we can replace y by $c * x$ on the right side. ... $\square$

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{\vdots}}{\neg(x = 0), \neg(y = 0), y = c * x \Longrightarrow ((c * x)/x) * x = y}}{\neg(x = 0), \neg(y = 0), y = c * x \Longrightarrow (y/x) * x = y}}{\neg(x = 0), \neg(y = 0), \exists\, \texttt{int}\ k;\ y = k * x \Longrightarrow (y/x) * x = y}$$

Features of the KeY Theorem Prover

Demo

`rel.key`, `twoInstances.key`

Feature List

- Can work on multiple proofs simultaneously (task list)
- Point-and-click navigation within proof
- Undo proof steps, prune proof trees
- Pop-up menu with proof rules applicable in pointer focus
- Preview of rule effect as tool tip
- Quantifier instantiation and equality rules by drag-and-drop
- Possible to hide (and unhide) parts of a sequent
- Saving and loading of proofs

Part III

Java Dynamic Logic

Java DL Example Formulas

$$(\texttt{balance} \geq c \land \texttt{amount} > 0) \rightarrow \langle \texttt{charge(amount);} \rangle \, \texttt{balance} > c$$

Java DL Example Formulas

$$(\texttt{balance} \geq c \land \texttt{amount} > 0) \rightarrow \langle \texttt{charge(amount);} \rangle \texttt{balance} > c$$

$$\texttt{o1.f} < \texttt{o2.f} \rightarrow [\texttt{t=o1.f; o1.f=o2.f; o2.f=t;}]\, \texttt{o2.f} < \texttt{o1.f}$$

Java DL Example Formulas

$$(\texttt{balance} \geq c \land \texttt{amount} > 0) \rightarrow \langle \texttt{charge(amount);} \rangle \texttt{balance} > c$$

$$\texttt{o1.f} < \texttt{o2.f} \rightarrow [\texttt{t=o1.f; o1.f=o2.f; o2.f=t;}]\, \texttt{o2.f} < \texttt{o1.f}$$

$$\texttt{o1.f} > 0 \rightarrow [\texttt{o1.f=o1.f+o2.f; o2.f=o1.f-o2.f; o1.f=o1.f-o2.f;}]\, \texttt{o2.f} > 0$$

Java DL Example Formulas

$$(\texttt{balance} \geq c \land \texttt{amount} > 0) \rightarrow \langle \texttt{charge(amount);} \rangle \texttt{balance} > c$$

$$\texttt{o1.f} < \texttt{o2.f} \rightarrow [\texttt{t=o1.f; o1.f=o2.f; o2.f=t;}]\, \texttt{o2.f} < \texttt{o1.f}$$

$$\texttt{o1.f} > 0 \rightarrow [\texttt{o1.f=o1.f+o2.f; o2.f=o1.f-o2.f; o1.f=o1.f-o2.f;}]\, \texttt{o2.f} > 0$$

Last formula not valid: If $\texttt{o1} = \texttt{o2}$ in prestate, then $\texttt{o2.f} = 0$ in poststate.

Java DL Example Formulas

$$(\texttt{balance} \geq c \land \texttt{amount} > 0) \rightarrow \langle \texttt{charge(amount);} \rangle \texttt{balance} > c$$

$$\texttt{o1.f} < \texttt{o2.f} \rightarrow [\texttt{t=o1.f; o1.f=o2.f; o2.f=t;}]\, \texttt{o2.f} < \texttt{o1.f}$$

$$\texttt{o1.f} > 0 \rightarrow [\texttt{o1.f=o1.f+o2.f; o2.f=o1.f-o2.f; o1.f=o1.f-o2.f;}]\, \texttt{o2.f} > 0$$

Last formula not valid: If $\texttt{o1} = \texttt{o2}$ in prestate, then $\texttt{o2.f} = 0$ in poststate.

Symbolic execution must take possible aliasing into account.

Java DL Example Formula

$$\forall\ \texttt{int}\ x;\ (\texttt{n} = x \land x >= 0 \rightarrow [\texttt{i = 0; r = 0; while (i<n) \{ i = i + 1; r = r + i;\} r=r+r-n;}]\ \texttt{r} = x * x)$$

Java DL Example Formula (in ASCII)

```
  a != null
->
  <
    int max = 0;
    if ( a.length > 0 ) max = a[0];
    int i = 1;
    while ( i < a.length ) {
      if ( a[i] > max ) max = a[i];
      ++i;
    }
  >
    \forall int j; (j >= 0 & j < a.length -> max >= a[j])
    &
    (a.length > 0 ->
      \exists int j; (j >= 0 & j < a.length & max = a[j]))
```

Assignment Rule Using Updates

Symbolic execution of assignment using updates

$$\text{assign}\ \frac{\Gamma \Longrightarrow \mathcal{U}\{x := t\}\langle \mathtt{rest}\rangle\varphi, \Delta}{\Gamma \Longrightarrow \mathcal{U}\langle x := t;\ \mathtt{rest}\rangle\varphi, \Delta}\quad t \text{ simple}$$

- $\mathcal{U}$: (nested) updates
- t simple: no side effects, no exceptions

Java DL Rules

Example rule

$$\text{if}\ \frac{\Gamma, \mathcal{U}(\texttt{b} = \mathit{true}) \Longrightarrow \mathcal{U}\langle \texttt{p};\ \mathit{rest}\rangle\varphi, \Delta \qquad \Gamma, \mathcal{U}(\texttt{b} = \mathit{false}) \Longrightarrow \mathcal{U}\langle \texttt{q};\ \mathit{rest}\rangle\varphi, \Delta}{\Gamma \Longrightarrow \mathcal{U}\langle \textbf{if}\ \texttt{(b) \{ p \} else \{ q \} ;}\ \mathit{rest}\rangle\varphi, \Delta}\ \text{b simple}$$

Symbolic execution must consider all possible execution branches

Java Dynamic Logic

Reasoning about Java programs requires extensions of FOL

- JAVA type hierarchy
- JAVA program variables
- JAVA heap for reference types

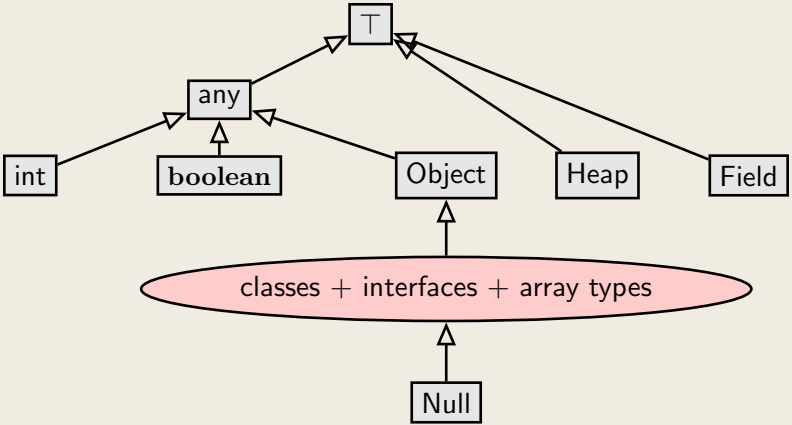
Type Hierarchy

Definition (Type Hierarchy)

- T_Σ is set of types
- Subtype relation $\sqsubseteq \subseteq T_\Sigma \times T_\Sigma$ with top element $\top$
  - $\tau \sqsubseteq \top$ for all $\tau \in T_\Sigma$

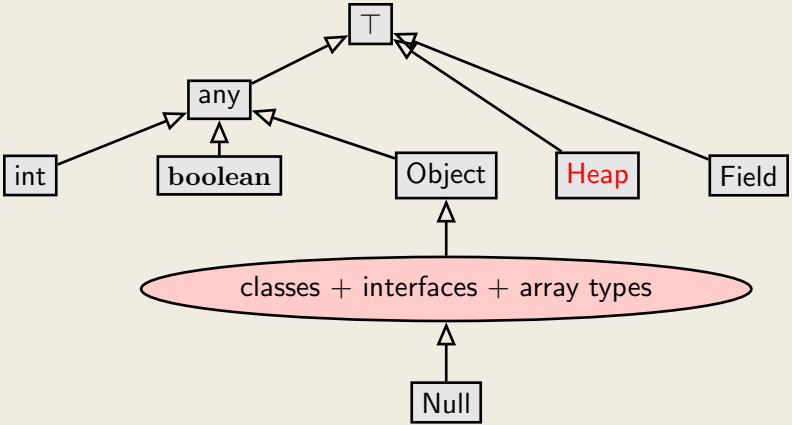
Modelling Java in FOL: Fixing a Type Hierarchy

T_Σ based on Java's type hierarchy (sketch)



Modelling Java in FOL: Fixing a Type Hierarchy

T_Σ based on Java's type hierarchy (sketch)



Modelling the Heap in FOL

The Java Heap

Objects are stored on (i.e., in) the heap.

- Status of heap changes during execution
- Each heap associates values to object/field pairs

Modelling the Heap in FOL

The Java Heap

Objects are stored on (i.e., in) the heap.

- Status of heap changes during execution
- Each heap associates values to object/field pairs

The Heap Model of KeY-DL

Each element of data type Heap represents a certain heap status.
Two functions involving heaps:

Modelling the Heap in FOL

The Java Heap

Objects are stored on (i.e., in) the heap.

- Status of heap changes during execution
- Each heap associates values to object/field pairs

The Heap Model of KeY-DL

Each element of data type Heap represents a certain heap status.
Two functions involving heaps:

- in F_Σ: Heap `store(Heap, Object, Field, any);`
  `store(`h, o, f, v`)` returns heap like h, but with v associated to $o.f$

Modelling the Heap in FOL

The Java Heap

Objects are stored on (i.e., in) the heap.

- Status of heap changes during execution
- Each heap associates values to object/field pairs

The Heap Model of KeY-DL

Each element of data type Heap represents a certain heap status.
Two functions involving heaps:

- in F_Σ: Heap `store(Heap, Object, Field, any);`
  `store(`h, o, f, v`)` returns heap like h, but with v associated to $o.f$
- in F_Σ: any `select(Heap, Object, Field);`
  `select(`h, o, f`)` returns value associated to $o.f$ in h

Modelling the Heap in FOL

Modelling instance fields

Person
`int age` `int id`
`int setAge(int newAge)` `int getId()`

- for each JAVA reference type C there is a type $C \in T_\Sigma$, e.g., Person

Modelling the Heap in FOL

Modelling instance fields

Person
int age **int** id
int setAge(**int** newAge) **int** getId()

- for each JAVA reference type C there is a type $C \in T_\Sigma$, e.g., Person
- for each field `f` there is a unique constant `f` of type Field, e.g., `id`

Modelling the Heap in FOL

Modelling instance fields

Person
int age **int** id
int setAge(**int** newAge) **int** getId()

- for each JAVA reference type C there is a type $C \in T_\Sigma$, e.g., Person
- for each field `f` there is a unique constant `f` of type Field, e.g., `id`
- heap maps pair (object, field) to value

Modelling the Heap in FOL

Modelling instance fields

Person
int age **int** id
int setAge(**int** newAge) **int** getId()

- for each JAVA reference type C there is a type $C \in T_\Sigma$, e.g., Person
- for each field `f` there is a unique constant `f` of type Field, e.g., `id`
- heap maps pair (object, field) to value

Modelling the Heap in FOL

Modelling instance fields

Person
int age **int** id
int setAge(**int** newAge) **int** getId()

- for each JAVA reference type C there is a type $C \in T_\Sigma$, e.g., Person
- for each field f there is a unique constant f of type Field, e.g., id
- heap maps pair (object, field) to value

Reading Field id of Person p

FOL notation $\texttt{select}(h, \texttt{p}, \texttt{id})$

KeY notation $\texttt{p.id@}h$ (abbreviating $\texttt{select}(h, \texttt{p}, \texttt{id})$)

Modelling the Heap in FOL

Modelling instance fields

Person
int age **int** id
int setAge(**int** newAge) **int** getId()

- for each JAVA reference type C there is a type $C \in T_\Sigma$, e.g., Person
- for each field f there is a unique constant f of type Field, e.g., id
- heap maps pair (object, field) to value

Reading Field id of Person p

FOL notation $\texttt{select}(h, \texttt{p}, \texttt{id})$

KeY notation $\texttt{p.id@}h$ (abbreviating $\texttt{select}(h, \texttt{p}, \texttt{id})$)

$\texttt{p.id}$ (abbreviating $\texttt{select}(\texttt{heap}, \texttt{p}, \texttt{id})$)[a]

[a] heap is special program variable for "current" heap; mostly implicit in $o.f$

Modelling the Heap in FOL

Modelling instance fields

Person
`int age` `int id`
`int setAge(int newAge)` `int getId()`

- for each JAVA reference type C there is a type $C \in T_\Sigma$, e.g., Person
- for each field `f` there is a unique constant `f` of type Field, e.g., `id`
- heap maps pair (object, field) to value

Writing to Field `id` of Person `p`

FOL notation $\texttt{store}(h, \texttt{p}, \texttt{id}, 6238)$

Modelling the Heap in FOL

Modelling instance fields

Person
`int age` `int id`
`int setAge(int newAge)` `int getId()`

- for each JAVA reference type C there is a type $C \in T_\Sigma$, e.g., Person
- for each field `f` there is a unique constant `f` of type Field, e.g., `id`
- heap maps pair (object, field) to value

Writing to Field `id` of Person `p`

FOL notation $\texttt{store}(h, \texttt{p}, \texttt{id}, 6238)$

KeY notation $h[\texttt{p.id} := 6238]$ (notation for `store`, not update)

The Algebra of Heaps

We do *not* formalise the *structure* (implementation) of heaps.
We formalise the *behaviour*, with an algebra of heap operations:

$$\texttt{select}(\texttt{store}(h, o, f, v), o, f) =$$

The Algebra of Heaps

We do *not* formalise the *structure* (implementation) of heaps.
We formalise the *behaviour*, with an algebra of heap operations:

$$\texttt{select}(\texttt{store}(h, o, f, v), o, f) = v$$

The Algebra of Heaps

We do *not* formalise the *structure* (implementation) of heaps.
We formalise the *behaviour*, with an algebra of heap operations:

$$\texttt{select}(\texttt{store}(h, o, f, v), o, f) = v$$

$$(o \neq o' \vee f \neq f') \;\rightarrow\; \texttt{select}(\texttt{store}(h, o, f, x), o', f') =$$

The Algebra of Heaps

We do *not* formalise the *structure* (implementation) of heaps.
We formalise the *behaviour*, with an algebra of heap operations:

$$\texttt{select}(\texttt{store}(h, o, f, v), o, f) = v$$

$$(o \neq o' \vee f \neq f') \;\rightarrow\; \texttt{select}(\texttt{store}(h, o, f, x), o', f') = \texttt{select}(h, o', f')$$

The Algebra of Heaps

We do *not* formalise the *structure* (implementation) of heaps.
We formalise the *behaviour*, with an algebra of heap operations:

$$\texttt{select}(\texttt{store}(h, o, f, v), o, f) = v$$

$$(o \neq o' \vee f \neq f') \;\rightarrow\; \texttt{select}(\texttt{store}(h, o, f, x), o', f') = \texttt{select}(h, o', f')$$

Example

$\texttt{select}(\texttt{store}(h, \texttt{o}, \texttt{f}, 15), \texttt{o}, \texttt{f}) \rightsquigarrow$
$\texttt{select}(\texttt{store}(h, \texttt{o}, \texttt{f}, 15), \texttt{o}, \texttt{g}) \rightsquigarrow$
$\texttt{select}(\texttt{store}(h, \texttt{o}, \texttt{f}, 15), \texttt{u}, \texttt{f}) \rightsquigarrow$

The Algebra of Heaps

We do *not* formalise the *structure* (implementation) of heaps.
We formalise the *behaviour*, with an algebra of heap operations:

$$\texttt{select}(\texttt{store}(h, o, f, v), o, f) = v$$

$$(o \neq o' \vee f \neq f') \;\rightarrow\; \texttt{select}(\texttt{store}(h, o, f, x), o', f') = \texttt{select}(h, o', f')$$

Example

$\texttt{select}(\texttt{store}(h, \texttt{o}, \texttt{f}, 15), \texttt{o}, \texttt{f}) \rightsquigarrow 15$
$\texttt{select}(\texttt{store}(h, \texttt{o}, \texttt{f}, 15), \texttt{o}, \texttt{g}) \rightsquigarrow$
$\texttt{select}(\texttt{store}(h, \texttt{o}, \texttt{f}, 15), \texttt{u}, \texttt{f}) \rightsquigarrow$

The Algebra of Heaps

We do *not* formalise the *structure* (implementation) of heaps.
We formalise the *behaviour*, with an algebra of heap operations:

$$\texttt{select}(\texttt{store}(h, o, f, v), o, f) = v$$

$$(o \neq o' \vee f \neq f') \;\rightarrow\; \texttt{select}(\texttt{store}(h, o, f, x), o', f') = \texttt{select}(h, o', f')$$

Example

$\texttt{select}(\texttt{store}(h, \texttt{o}, \texttt{f}, 15), \texttt{o}, \texttt{f}) \rightsquigarrow 15$
$\texttt{select}(\texttt{store}(h, \texttt{o}, \texttt{f}, 15), \texttt{o}, \texttt{g}) \rightsquigarrow \texttt{select}(h, \texttt{o}, \texttt{g})$
$\texttt{select}(\texttt{store}(h, \texttt{o}, \texttt{f}, 15), \texttt{u}, \texttt{f}) \rightsquigarrow$

The Algebra of Heaps

We do *not* formalise the *structure* (implementation) of heaps.
We formalise the *behaviour*, with an algebra of heap operations:

$$\texttt{select}(\texttt{store}(h, o, f, v), o, f) = v$$

$$(o \neq o' \vee f \neq f') \;\rightarrow\; \texttt{select}(\texttt{store}(h, o, f, x), o', f') = \texttt{select}(h, o', f')$$

Example

$\texttt{select}(\texttt{store}(h, \texttt{o}, \texttt{f}, 15), \texttt{o}, \texttt{f}) \rightsquigarrow 15$
$\texttt{select}(\texttt{store}(h, \texttt{o}, \texttt{f}, 15), \texttt{o}, \texttt{g}) \rightsquigarrow \texttt{select}(h, \texttt{o}, \texttt{g})$
$\texttt{select}(\texttt{store}(h, \texttt{o}, \texttt{f}, 15), \texttt{u}, \texttt{f}) \rightsquigarrow$
$\quad \texttt{if } (\texttt{o} = \texttt{u}) \texttt{ then } 15 \texttt{ else } \texttt{select}(h, \texttt{u}, \texttt{f})$

Pretty Printing

Shorthand Notations for Heap Operations

$\texttt{o.f@h}$	is	$\texttt{select}(\texttt{h}, \texttt{o}, \texttt{f})$
$\texttt{h}[\texttt{o.f} := \texttt{v}]$	is	$\texttt{store}(\texttt{h}, \texttt{o}, \texttt{f}, \texttt{v})$

Pretty Printing

Shorthand Notations for Heap Operations

$\mathtt{o.f@h}$	is	$\mathtt{select(h, o, f)}$
$\mathtt{h[o.f := v]}$	is	$\mathtt{store(h, o, f, v)}$

therefore:

$\mathtt{u.f@h[o.f := v]}$	is	$\mathtt{select(store(h, o, f, v), u, f)}$
$\mathtt{h[o.f := v][o'.f' := v']}$	is	$\mathtt{store(store(h, o, f, v), o', f', v')}$

Pretty Printing

Shorthand Notations for Heap Operations

$\texttt{o.f@h}$	is	$\texttt{select(h, o, f)}$
$\texttt{h[o.f := v]}$	is	$\texttt{store(h, o, f, v)}$

therefore:

$\texttt{u.f@h[o.f := v]}$	is	$\texttt{select(store(h, o, f, v), u, f)}$
$\texttt{h[o.f := v][o'.f' := v']}$	is	$\texttt{store(store(h, o, f, v), o', f', v')}$

Very-Shorthand Notations for Current Heap

Current heap always in special variable heap.

$\texttt{o.f}$	is	$\texttt{select(heap, o, f)}$
$\texttt{\{o.f := v\}}$	is update	$\texttt{\{heap := heap[o.f := v]\}}$

Field Update Assignment Rule

Changing the value of fields

How to (symbolically) execute assignment to field, e.g., `p.age=18;` ?

$$\text{assign}\ \frac{\Gamma \Longrightarrow \{\mathtt{o.f} := t\}\langle \mathtt{rest}\rangle\varphi, \Delta}{\Gamma \Longrightarrow \langle \mathtt{o.f = t;\ rest}\rangle\varphi, \Delta}$$

Admit on left-hand side of update JAVA location expressions

Field Update Assignment Rule

Changing the value of fields

How to (symbolically) execute assignment to field, e.g., `p.age=18;` ?

$$\text{assign}\ \frac{\Gamma \Longrightarrow \{\texttt{p.age} := 18\}\langle \textit{rest}\rangle\varphi, \Delta}{\Gamma \Longrightarrow \langle \texttt{p.age = 18; } \textit{rest}\rangle\varphi, \Delta}$$

Admit on left-hand side of update JAVA location expressions

Field Update Assignment Rule

Changing the value of fields

How to (symbolically) execute assignment to field, e.g., `p.age=18;` ?

$$\text{assign}\ \frac{\Gamma \Longrightarrow \{\mathtt{o.f} := t\}\langle \mathit{rest}\rangle\varphi, \Delta}{\Gamma \Longrightarrow \langle \mathtt{o.f = t;}\ \mathit{rest}\rangle\varphi, \Delta}$$

Admit on left-hand side of update JAVA location expressions

But is this rule correct? See below.

Dynamic Logic: KeY input file

```
\javaSource "path to source code referenced in problem";

\programVariables { Person p; }

\problem {
       \<{   p.age = 18;  }\> p.age = 18
}
```

KeY reads in all source files and creates automatically
the necessary signature (types, program variables, field constants)

Dynamic Logic: KeY input file

```
\javaSource "path to source code referenced in problem";

\programVariables { Person p; }

\problem {
       \<{   p.age = 18;  }\> p.age = 18
}
```

KeY reads in all source files and creates automatically
the necessary signature (types, program variables, field constants)

Demo

`updates/firstAttributeExample.key`

Refined Semantics of Program Modalities

Does abrupt termination count as normal termination?
No! Need to distinguish normal and exceptional termination

Refined Semantics of Program Modalities

Does abrupt termination count as normal termination?
No! Need to distinguish normal and exceptional termination

- $\langle \mathrm{p} \rangle \varphi$: p terminates normally and formula φ holds in final state (total correctness)

Refined Semantics of Program Modalities

Does abrupt termination count as normal termination?
No! Need to distinguish normal and exceptional termination

- $\langle \mathrm{p} \rangle \varphi$: p terminates normally and formula φ holds in final state (total correctness)
- $[\mathrm{p}] \varphi$: If p terminates normally then formula φ holds in final state (partial correctness)

Refined Semantics of Program Modalities

Does abrupt termination count as normal termination?
No! Need to distinguish normal and exceptional termination

- $\langle \mathrm{p} \rangle \varphi$: p terminates normally and formula φ holds in final state (total correctness)
- $[\mathrm{p}] \varphi$: If p terminates normally then formula φ holds in final state (partial correctness)

Abrupt termination on top-level counts as non-termination!

Example Reconsidered: Exception Handling

```
\javaSource "path to source code";

\programVariables {
  ...
}

\problem {
       p != null -> \<{   p.age = 18;  }\> p.age = 18
}
```

Only provable when no top-level exception is thrown

Demo

`updates/secondAttributeExample.key`

Which Objects do Exist?

How to model object creation with **new** ?

Which Objects do Exist?

How to model object creation with **new** ?

Constant Domain Assumption

Assume that domain $\mathcal{D}$ is the same in all states $(\mathcal{D}, \delta, \mathcal{I}) \in \mathit{States}$

Consequence:
Quantifiers and modalities commute:

$$\models (\forall\ T\ x;\ [\mathrm{p}]\varphi) \leftrightarrow [\mathrm{p}](\forall\ T\ x;\ \varphi)$$

Object Creation

Realizing Constant Domain Assumption

- Implicitly declared field **boolean** `<created>` in class `Object`
- `<created>` has value **true** iff argument object has been created
- Object creation modeled as $\{\texttt{heap} := \texttt{create(heap, ob)}\}$ for not (yet) created `ob` (essentially sets `<created>` field of `ob` to **true**)

Object Creation

Realizing Constant Domain Assumption

- Implicitly declared field **boolean** `<created>` in class `Object`
- `<created>` has value **true** iff argument object has been created
- Object creation modeled as $\{\texttt{heap} := \texttt{create(heap, ob)}\}$ for not (yet) created `ob` (essentially sets `<created>` field of `ob` to **true**)

$$\frac{\Gamma,\ \texttt{ob.<created>} = \texttt{FALSE} \Longrightarrow \{\texttt{heap} := \texttt{create}(\texttt{heap}, \texttt{ob})\}\{\texttt{o} := \texttt{ob}\}\langle \texttt{o.<init>}(\textit{param})\,;\,\omega\rangle\varphi,\ \Delta}{\Gamma \Longrightarrow \langle \texttt{o = new T}(\textit{param})\,;\ \omega\rangle\varphi, \Delta}$$

`ob` is a fresh program variable

Object Creation

Realizing Constant Domain Assumption

- Implicitly declared field **boolean** `<created>` in class `Object`
- `<created>` has value **true** iff argument object has been created
- Object creation modeled as $\{\texttt{heap} := \texttt{create(heap, ob)}\}$ for not (yet) created `ob` (essentially sets `<created>` field of `ob` to **true**)

$$\frac{\Gamma,\ \texttt{ob.<created>} = \texttt{FALSE} \Longrightarrow \{\texttt{heap} := \texttt{create(heap, ob)}\}\{\texttt{o} := \texttt{ob}\}\langle \texttt{o.<init>(}\mathit{param}\texttt{)}\,;\,\omega\rangle\varphi,\ \Delta}{\Gamma \Longrightarrow \langle \texttt{o = new T(}\mathit{param}\texttt{)}\,;\ \omega\rangle\varphi, \Delta}$$

`ob` is a fresh program variable

Alternatives exisit in the literature. E.g.:
[Ahrendt, de Boer, Grabe, *Abstract Object Creation in Dynamic Logic – To Be or Not To Be Created*, Springer, LNCS 5850]

Dynamic Logic to (almost) full Java

KeY supports full sequential Java, with some limitations:

- No concurrency
- No generics
- No I/O
- No dynamic class loading or reflection
- API method calls: need either JML contract or implementation

Dynamic Logic to (almost) full Java

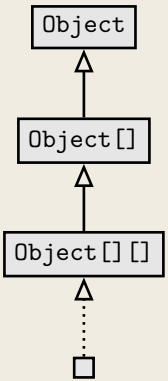
KeY supports full sequential Java, with some limitations:

- No concurrency
- No generics
- No I/O
- No dynamic class loading or reflection
- API method calls: need either JML contract or implementation

- Recently added: support for floating-point numbers/arithemic

Java Features in Dynamic Logic: Arrays

Arrays



- JAVA type hierarchy includes array types
- Types ordered according to JAVA subtyping rules
- Function `arr` : int $\to$ Field turns integer index into type Field (required in `store`).
- Store array elements on heap
- Value of `a[i]` in heap $\texttt{store}(\texttt{heap}, \texttt{a}, \texttt{arr}(\texttt{i}), 8)$ is 8
- Arrays `a` and `b` can refer to same object (aliasing)

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

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

Temporary variables store result of evaluating subexpression

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

Java Features in Dynamic Logic: Abrupt Termination

Abrupt Termination: Exceptions and Jumps

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

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

Java Features in Dynamic Logic: Abrupt Termination

Abrupt Termination: Exceptions and Jumps

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

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

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

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

Java Features in Dynamic Logic: Abrupt Termination

Abrupt Termination: Exceptions and Jumps

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

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

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

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

Demo

`exceptions/try-catch.key` (inspect file, auto-prove, inspect proof)

Java Features in Dynamic Logic: Null

Null pointer exceptions

There are no "exceptions" in FOL: interpretations $\mathcal{I}$ are total

Need to model possibility that $\mathsf{o} = \mathbf{null}$ in $\mathsf{o.a}$

Java Features in Dynamic Logic: Null

Null pointer exceptions

There are no "exceptions" in FOL: interpretations $\mathcal{I}$ are total

Need to model possibility that $o = \mathbf{null}$ in $o.a$

- KeY branches over $o = \mathbf{null}$ and $o \neq \mathbf{null}$ upon each field access *within modalities*[a]

Java Features in Dynamic Logic: Null

Null pointer exceptions

There are no "exceptions" in FOL: interpretations $\mathcal{I}$ are total

Need to model possibility that $\mathtt{o} = \mathbf{null}$ in $\mathtt{o.a}$

- KeY branches over $\mathtt{o} = \mathbf{null}$ and $\mathtt{o} \neq \mathbf{null}$ upon each field access *within modalities*[a]
- Thereby, $\mathtt{o.a}$ appears *outside modalities* mostly under assumption $\mathtt{o} \neq \mathbf{null}$

Java Features in Dynamic Logic: Null

Null pointer exceptions

There are no "exceptions" in FOL: interpretations $\mathcal{I}$ are total

Need to model possibility that o $= \mathbf{null}$ in o.a

- KeY branches over o $= \mathbf{null}$ and o $\neq \mathbf{null}$ upon each field access *within modalities*[a]
- Thereby, o.a appears *outside modalities* mostly under assumption o $\neq \mathbf{null}$
- $\mathbf{null}$.a *outside modalities* has a value, which is unknown

Java Features in Dynamic Logic: Null

Null pointer exceptions

There are no "exceptions" in FOL: interpretations $\mathcal{I}$ are total

Need to model possibility that o $= \mathbf{null}$ in o.a

- KeY branches over o $= \mathbf{null}$ and o $\neq \mathbf{null}$ upon each field access *within modalities*[a]
- Thereby, o.a appears *outside modalities* mostly under assumption o $\neq \mathbf{null}$
- $\mathbf{null}$.a *outside modalities* has a value, which is unknown

[a] Can be changed with Taclet Option runtimeExceptions

Field Update Assignment Rule Revisited (A)

Changing the value of fields

How to (symbolically) execute assignment to field?

$$\frac{\Gamma, \qquad \Longrightarrow \{\mathtt{o.f} := \mathtt{e}\}\langle \quad \omega\rangle\varphi, \Delta}{\Gamma \Longrightarrow \langle \quad \mathtt{o.f} = \mathtt{e}; \omega\rangle\varphi, \Delta}$$

Field Update Assignment Rule Revisited (A)

Changing the value of fields

How to (symbolically) execute assignment to field?

$$\frac{\Gamma, \mathtt{o} \neq \mathbf{null} \Longrightarrow \{\mathtt{o.f} := \mathtt{e}\}\langle \ \ \omega\rangle\varphi, \Delta}{\Gamma \Longrightarrow \langle \ \mathtt{o.f} = \mathtt{e}; \omega\rangle\varphi, \Delta}$$

Field Update Assignment Rule Revisited (A)

Changing the value of fields

How to (symbolically) execute assignment to field?

$$\frac{\Gamma, \mathtt{o} = \mathbf{null} \Longrightarrow \qquad\qquad , \Delta \qquad\qquad \Gamma, \mathtt{o} \neq \mathbf{null} \Longrightarrow \{\mathtt{o.f} := \mathtt{e}\}\langle \quad \omega\rangle\varphi, \Delta}{\Gamma \Longrightarrow \langle \quad \mathtt{o.f} = \mathtt{e}; \omega\rangle\varphi, \Delta}$$

Field Update Assignment Rule Revisited (A)

Changing the value of fields

How to (symbolically) execute assignment to field?

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

Field Update Assignment Rule Revisited (A)

Changing the value of fields

How to (symbolically) execute assignment to field?

$$\frac{\Gamma, \mathsf{o} \neq \mathbf{null} \Longrightarrow \{\mathsf{o.f} := \mathsf{e}\}\langle\pi\ \omega\rangle\varphi, \Delta \qquad \Gamma, \mathsf{o} = \mathbf{null} \Longrightarrow \langle\pi\, \mathbf{throw}\ \texttt{new NullPointerException();}\ \omega\rangle\varphi, \Delta}{\Gamma \Longrightarrow \langle\pi\, \mathsf{o.f} = \mathsf{e};\, \omega\rangle\varphi, \Delta}$$

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

Field Update Assignment Rule Revisited (B)

Changing the value of fields

How to (symbolically) execute assignment to field?

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

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

Revisit: Field Assignment Demo

```
\javaSource "path to source code referenced in problem";

\programVariables { Person p; }

\problem {
       p != null -> \<{   p.age = 18;  }\> p.age = 18
}
```

Demo

updates/secondAttributeExample.key

Java Features in Dynamic Logic: Aliasing

Demo

`aliasing/attributeAlias1.key`

Java Features in Dynamic Logic: Aliasing

Demo

`aliasing/attributeAlias1.key`

Reference Aliasing

Alias resolution causes proof split