



Deductive Verification of Object-Oriented Software

Part B

Bernhard Beckert | VTSA, 24.-28.08.2015

KIT – INSTITUTE FOR THEORETICAL COMPUTER SCIENCE



KIT – University of the State of Baden-Wuerttemberg and National Laboratory of the Helmholtz Association

www.kit.edu

<ロ> < 置> < 置> < 置> < 置 > のへの

Part III

Program Verification with Dynamic Logic



Sequent Calculus

Rules for Programs: Symbolic Execution



A Calculus for 100% JAVA CARD



Taclets – KeY's Rule Description Language

JAVA CARD DL

Sequent Calculus Bernhard Beckert - Deductive Verification of Object-Oriented Software

Symbolic Execution

イロト イヨト イヨト A Calculus for 100% JAVA CARD VTSA, 24.-28.08.2015

Taclets 35/102

nac 1

Part III

Program Verification with Dynamic Logic

🕖 JAVA CARD DL

Sequent Calculus

9 Rules for Programs: Symbolic Execution

0 A Calculus for 100% JAVA CARD

Taclets – KeY's Rule Description Language

JAVA CARD DL

Sequent Calculus

Symbolic Execution

4 □ > 4 (권 > 4 분 > 4 분 > 1 분)
A Calculus for 100% JAVA CARD
0000

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015

Taclets 00000 36/102

nac



Syntax

- Basis: Typed first-order predicate logic
- Modal operators $\langle p \rangle$ and [p] for each (JAVA CARD) program p
- Class definitions in background (not shown in formulas)

• F: p terminates and F holds in the final state

JAVA CARD DL 0000000

Sequent Calculus Bernhard Beckert - Deductive Verification of Object-Oriented Software

Symbolic Execution

A

イロト 人間ト イヨト イヨト -

nan

VTSA, 2428.08.2015	37/10
000	00000
Calculus for 100% JAVA CARD	laciets



Syntax

- Basis: Typed first-order predicate logic
- Modal operators $\langle p
 angle$ and [p] for each (JAVA CARD) program p
- Class definitions in background (not shown in formulas)

Semantics (Kripke)

Modal operators allow referring to the final state of *p*:

• [p] F: If p terminates, then F holds in the final state

(partial correctness)

F: p terminates and F holds in the final state

(total correctness)

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD 0000 VTSA, 24,-28,08,2015

シ ۹ (~ Taclets 00000 37/102

=

Bernhard Beckert - Deductive Verification of Object-Oriented Software



Syntax

- Basis: Typed first-order predicate logic
- Modal operators $\langle p \rangle$ and [p] for each (JAVA CARD) program p
- Class definitions in background (not shown in formulas)

Semantics (Kripke)

Modal operators allow referring to the final state of p:

• [p] F: If p terminates, then F holds in the final state

(partial correctness)

JAVA CARD DL 0000000

Sequent Calculus Bernhard Beckert - Deductive Verification of Object-Oriented Software

Symbolic Execution

A Calculus for 100% JAVA CARD Taclets VTSA, 24.-28.08.2015 37/102



Syntax

- Basis: Typed first-order predicate logic
- Modal operators $\langle p \rangle$ and [p] for each (JAVA CARD) program p
- Class definitions in background (not shown in formulas)

Semantics (Kripke)

Modal operators allow referring to the final state of *p*:

• [p] F: If p terminates, then F holds in the final state

(partial correctness)

• F: p terminates and F holds in the final state

(total correctness)

nac

<ロ> <同> < 同> < 三> < 三> < 三</p>

 JAVA CARD DL
 Sequent Calculus
 Symbolic Execution
 A Calculus for 100% JAVA CARD
 Taclets

 0000000
 00000
 00000
 0000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000
 00000



Transparency wrt target programming language

- Encompasses Hoare Logic
- More expressive and flexible than Hoare logic
- Symbolic execution is a natural interactive proof paradigm
- Programs are "first-class citizens"
- Real Java syntax

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD

ク Q (~ Taclets 00000 38/102

 \equiv

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015

イロト イポト イヨト



Transparency wrt target programming language

- Encompasses Hoare Logic
- More expressive and flexible than Hoare logic
- Symbolic execution is a natural interactive proof paradigm

Hoare triple $\{\psi\} \alpha \{\phi\}$ equiv. to DL formula $\psi \rightarrow [\alpha] \phi$

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD

Taclets 00000 38/102

nac

 \equiv

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015

< □ > < □ > < □ > < □ > < □ > <



- Transparency wrt target programming language
- Encompasses Hoare Logic
- More expressive and flexible than Hoare logic

Not merely partial/total correctness:

- can employ programs for specification (e.g., verifying program transformations)
- can express security properties (two runs are indistinguishable)
- extension-friendly (e.g., temporal modalities)

JAVA CARD DI Sequent Calculus Symbolic Execution 0000000

A Calculus for 100% JAVA CARD Taclets 38/102

nac

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015



- Transparency wrt target programming language
- Encompasses Hoare Logic
- More expressive and flexible than Hoare logic
- Symbolic execution is a natural interactive proof paradigm

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD

Taclets 00000 38/102

= nar

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015



 $(balance >= c \& amount > 0) \rightarrow \\ <charge(amount); > balance > c$

<x = 1;>([while (true) {}] false)

Program formulas can appear nested

 $\texttt{forall } int \ val; \left((\ (<q \times \pm val) \right)$

p, q equivalent relative to computation state restricted to x

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD

うく(~ Taclets 00000 39/102

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015

(日)



 $(balance >= c \& amount > 0) \rightarrow \\ <charge(amount); > balance > c$

<x = 1;>([while (true) {}] false)

Program formulas can appear nested

$$forall int val; (((""$$

p, q equivalent relative to computation state restricted to x

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD

: クへで Taclets 00000 39/102

 \equiv

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015

イロト イポト イヨト イヨト



 $(balance >= c \& amount > 0) \rightarrow \\ <charge(amount); > balance > c$

Program formulas can appear nested

$$\texttt{forall } int \ val; \left((\ (""$$

p, q equivalent relative to computation state restricted to x

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD

Taclets 00000 39/102

nan

 \equiv

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015

イロト 人間ト イヨト イヨト



 $(balance >= c \& amount > 0) \rightarrow \\ <charge(amount); > balance > c$

Program formulas can appear nested

$$forall int val; (((""$$

p, q equivalent relative to computation state restricted to x

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD 0000 VTSA, 24.–28.08.2015

う Q (~ Taclets 00000 39/102

 \equiv

Bernhard Beckert - Deductive Verification of Object-Oriented Software



 $(balance >= c \& amount > 0) \rightarrow \\ <charge(amount); > balance > c$

Program formulas can appear nested

$$\texttt{forall } int \ val; \left(($$

p, q equivalent relative to computation state restricted to x

JAVA CARD DL

Sequent Calculus

Symbolic Execution

on A Ca

A Calculus for 100% JAVA CARD 0000 VTSA, 24.–28.08.2015

< □ > < □ > < □ > < □ > < □ > <

ク Q (~ Taclets 00000 39/102

 \equiv

Bernhard Beckert - Deductive Verification of Object-Oriented Software



```
a != null
  ->
     <
       int max = 0;
       if (a.length > 0) max = a[0];
       int i = 1;
       while ( i < a.length ) {</pre>
          if ( a[i] > max ) max = a[i];
          ++i:
     >
          \forall int j; (j >= 0 & j < a.length -> max >= a[i])
          &
          (a.length > 0 ->
            \exists int j; (j >= 0 & j < a.length & max = a[j]))</pre>
        )
                                                         イロト 人間ト イヨト イヨト
                                                                                3
                                                                                   nac
JAVA CARD DL
                 Sequent Calculus
                                   Symbolic Execution
                                                       A Calculus for 100% JAVA CARD
                                                                                   Taclets
00000000
                                                               VTSA, 24.-28.08.2015
                                                                                    40/102
Bernhard Beckert - Deductive Verification of Object-Oriented Software
```

Variables



Logical variables disjoint from program variables

- No quantification over program variables
- Programs do not contain logical variables
- "Program variables" actually non-rigid functions

JAVA CARD DL

Sequent Calculus

Symbolic Execution

▲ □ ▷ < () ▷ ▷ < () ▷ ▷ < () ▷ ▷ ○ ○
 A Calculus for 100% JAVA CARD
 0000

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015

Taclets 00000 41/102

nac



Example

 $(i + 1 \doteq x) = (i + 1 \doteq x)$

Interpretation of i depends on computation state \Rightarrow flexible

Locations are always flexible Logical variables, standard functions are always rigid

A Calculus for 100% JAVA CARD DL Sequent Calculus Symbolic Execution Ococo O



Example

 $(i + 1 \doteq x) = (i + 1 \doteq x)$

Interpretation of i depends on computation state ⇒ flexible
 Interpretation of x and + do not depend on state ⇒ rigid
 Locations are always flexible
 Logical variables, standard functions are always rigid



Example

$$(i + 1 \doteq x) = (i + 1 \doteq x)$$

- Interpretation of i depends on computation state \Rightarrow flexible
- Interpretation of x and + do not depend on state

 $\Rightarrow flexible \\\Rightarrow rigid$

Locations are always flexible Logical variables, standard functions are always rigid

A Calculus for 100% JAVA CARD DL Sequent Calculus Symbolic Execution A Calculus for 100% JAVA CARD Taclets Sococo Coco Sernhard Beckert – Deductive Verification of Object-Oriented Software VTSA, 24.-28.08.2015 42/102



Example

$$\operatorname{i}; \operatorname{i}, x; (\mathbf{i} + 1 \doteq x \rightarrow \operatorname{i}, (\mathbf{i} + \mathbf{i}))$$

- Interpretation of i depends on computation state \Rightarrow flexible
- Interpretation of x and + do not depend on state

 \Rightarrow rigid

Locations are always flexible Logical variables, standard functions are always rigid



Example

 $\langle \text{int } \mathbf{i}; \rangle \langle \text{forall } int \; x; (\mathbf{i} + 1 \doteq x \rightarrow \langle \mathbf{i} + \mathbf{i}; \rangle (\mathbf{i} \doteq x))$

- Interpretation of i depends on computation state ⇒ flexible
- Interpretation of x and + do not depend on state

\Rightarrow rigid

Locations are always flexible Logical variables, standard functions are always rigid



Validity



A JAVA CARD DL formula is valid iff it is true in all states.

We need a calculus for checking validity of formulas

JAVA CARD DL

Sequent Calculus

Symbolic Execution

< □ ▷ < □ ▷ < Ξ ▷ < Ξ ▷ < Ξ ▷ Ξ
 A Calculus for 100% JAVA CARD
 0000

Taclets 00000 43/102

= nar

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015





A JAVA CARD DL formula is valid iff it is true in all states.

We need a calculus for checking validity of formulas

JAVA CARD DL

Sequent Calculus

Symbolic Execution

4 □ ▷ 4 큔 ▷ 4 분 ▷ 4 분 ▷ 4
A Calculus for 100% JAVA CARD
0000

Taclets 00000 43/102

= nar

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015

Part III

Program Verification with Dynamic Logic

🕖 JAVA CARD DL

Sequent Calculus

9 Rules for Programs: Symbolic Execution

0 A Calculus for 100% JAVA CARD

Taclets – KeY's Rule Description Language

JAVA CARD DL

Sequent Calculus

Symbolic Execution

ecution /

A Calculus for 100% JAVA CARD 0000 VTSA, 24,-28,08,2015

イロト イポト イヨト イヨト

Bernhard Beckert - Deductive Verification of Object-Oriented Software

Part III

Program Verification with Dynamic Logic



Sequent Calculus

9 Rules for Programs: Symbolic Execution

0 A Calculus for 100% JAVA CARD

1 Taclets – KeY's Rule Description Language

JAVA CARD DL

Sequent Calculus

Symbolic Execution

Ition A Ca

A Calculus for 100% JAVA CARD 0000 VTSA, 24-28,08,2015

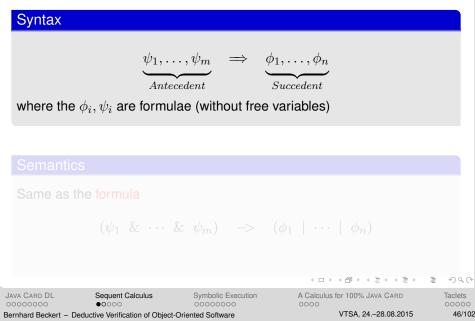
イロト イポト イヨト イヨト

ク へ (~ Taclets 00000 45/102

Bernhard Beckert - Deductive Verification of Object-Oriented Software

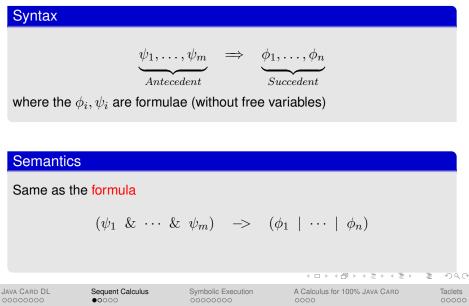
Sequents and their Semantics





Sequents and their Semantics



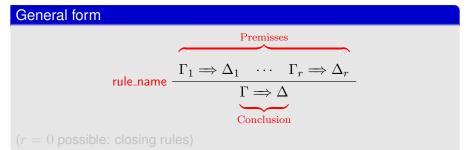


Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015

46/102



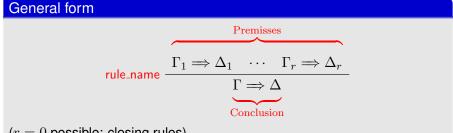


Soundness

If all premisses are valid, then the conclusion is valid

Use in practice Goal is matched to conclusion JAVA CARD DL Sequent Calculus Symbolic Execution occose oc





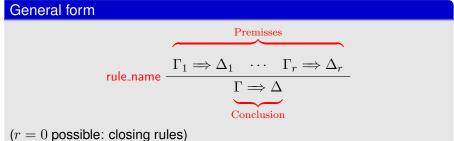
(r = 0 possible: closing rules)

Soundness

If all premisses are valid, then the conclusion is valid

Use in practice Goal is matched to conclusion JAVA CARD DL Sequent Calculus Symbolic Execution October Calculus for 100% JAVA CARD Taclets October Calculus Verification of Object-Oriented Software VTSA, 24.–28.08.2015 47/102



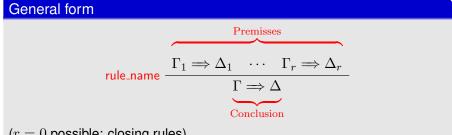


Soundness

If all premisses are valid, then the conclusion is valid

JAVA CARD DL 00000000	Sequent Calculus ○●○○○ Deductive Verification of Object	Symbolic Execution	A Calculus for 100% JAVA CARD 0000 VTSA, 24.–28.08.2015	Taclets 00000 47/102
Goal is ma	atched to conclus			10





(r = 0 possible: closing rules)

Soundness

If all premisses are valid, then the conclusion is valid

Use in pr	actice			
Goal is ma	atched to conclu	sion		= .,
Java Card DL 00000000 Bernhard Beckert – E	Sequent Calculus OOOO Deductive Verification of Object	Symbolic Execution 0000000 -Oriented Software	A Calculus for 100% JAVA CARD 0000 VTSA, 24.–28.08.2015	Taclets 00000 47/102

Some Simple Sequent Rules



$$\begin{array}{c} \operatorname{not} \operatorname{left} & \frac{\Gamma \Longrightarrow A, \Delta}{\Gamma, ! A \Longrightarrow \Delta} \\ \\ \operatorname{imp.left} & \frac{\Gamma \Longrightarrow A, \Delta}{\Gamma, A \Longrightarrow \Delta} \\ \\ \operatorname{imp.left} & \frac{\Gamma \Rightarrow A, \Delta}{\Gamma, A \to B \Longrightarrow \Delta} \\ \\ \operatorname{close_goal} & \frac{\Gamma, A \Longrightarrow A, \Delta}{\Gamma, A \to B \Longrightarrow \Delta} \\ \\ \operatorname{close_goal} & \frac{\Gamma, A \Longrightarrow A, \Delta}{\Gamma, A \to B \Longrightarrow \Delta} \\ \\ \operatorname{close_by_true} & \frac{\Gamma \Longrightarrow \operatorname{true}, \Delta}{\Gamma \Rightarrow \operatorname{true}, \Delta} \\ \\ \operatorname{all_left} & \frac{\Gamma, \backslash \operatorname{forall} t x; \phi, \{x/e\}\phi \Rightarrow \Delta}{\Gamma, \backslash \operatorname{forall} t x; \phi \Rightarrow \Delta} \\ \\ \operatorname{where} e \text{ var-free term of type } t' \prec t \\ \\ \operatorname{coecococc} \\ \\ \\ \operatorname{Sequent Calculus} \\ \underset{O \subseteq O \subset O}{\operatorname{Symbolic} \operatorname{Execution}} \\ \\ \operatorname{coecoccc} \\ \\ \end{array} \\ \begin{array}{c} \operatorname{ACalculus for 100\% Java CARD} \\ \operatorname{Calculus for 100\% Java CARD} \\ \\ \operatorname{Sequent Calculus} \\ \operatorname{Symbolic} \operatorname{Execution} \\ \\ \operatorname{coecocccc} \\ \end{array} \\ \begin{array}{c} \operatorname{ACalculus for 100\% Java CARD} \\ \operatorname{Sector} \\ \\ \operatorname{Coecoccccc} \\ \end{array} \\ \begin{array}{c} \operatorname{ACalculus for 100\% Java CARD} \\ \operatorname{Sector} \\ \end{array} \\ \begin{array}{c} \operatorname{ACalculus for 100\% Java CARD} \\ \operatorname{Calculus for 100\% Java CARD} \\ \end{array} \\ \begin{array}{c} \operatorname{ACalculus for 100\% Java CARD} \\ \operatorname{Calculus for 100\% Java CARD} \\ \end{array} \\ \begin{array}{c} \operatorname{ACalculus for 100\% Java CARD} \\ \operatorname{Calculus for 100\% Java CARD} \\ \end{array} \\ \begin{array}{c} \operatorname{ACalculus for 100\% Java CARD} \\ \operatorname{ACalculus for 100\% Java CARD} \\ \end{array} \\ \begin{array}{c} \operatorname{ACalculus for 100\% Java CARD} \\ \operatorname{ACalculus for 100\% Java CARD} \\ \end{array} \\ \begin{array}{c} \operatorname{ACalculus for 100\% Java CARD} \\ \operatorname{ACalculus for 100\% Java CARD} \\ \end{array} \\ \begin{array}{c} \operatorname{ACalculus for 100\% Java CARD} \\ \operatorname{ACalculus for 100\% Java CARD} \\ \end{array} \\ \begin{array}{c} \operatorname{ACalculus for 100\% Java CARD} \\ \operatorname{ACalculus for 100\% Java CARD} \\ \end{array} \\ \begin{array}{c} \operatorname{ACalculus for 100\% Java CARD} \\ \operatorname{ACalculus for 100\% Java CARD} \\ \operatorname{ACalculus for 100\% Java CARD} \\ \end{array} \\ \begin{array}{c} \operatorname{ACalculus for 100\% Java CARD} \\ \operatorname{ACalculus for 100\% Java CARD} \\ \operatorname{ACalculus for 100\% Java CARD} \\ \end{array} \\ \begin{array}{c} \operatorname{ACalculus for 100\% Java CARD} \\ \operatorname{ACalculus for 100\% Java CARD} \\ \end{array} \\ \end{array} \\ \begin{array}{c} \operatorname{ACalculus for 100\% Java CARD} \\ \operatorname{ACalculus for 100\% Java CARD} \\ \end{array} \\ \end{array}$$

Some Simple Sequent Rules



$$\operatorname{not_left} \quad \frac{\Gamma \Rightarrow A, \Delta}{\Gamma, ! A \Rightarrow \Delta}$$

$$\operatorname{imp_left} \quad \frac{\Gamma \Rightarrow A, \Delta}{\Gamma, A \Rightarrow A} \quad \Gamma, B \Rightarrow \Delta$$

$$\operatorname{imp_left} \quad \frac{\Gamma \Rightarrow A, \Delta}{\Gamma, A \Rightarrow B \Rightarrow \Delta}$$

$$\operatorname{close_goal} \quad \overline{\Gamma, A \Rightarrow A, \Delta} \qquad \operatorname{close_by_true} \quad \overline{\Gamma \Rightarrow \operatorname{true}, \Delta}$$

$$\operatorname{all_left} \quad \frac{\Gamma, \backslash \operatorname{forall} t x; \phi, \{x/e\}\phi \Rightarrow \Delta}{\Gamma, \backslash \operatorname{forall} t x; \phi \Rightarrow \Delta}$$

$$\operatorname{where } e \text{ var-free term of type } t' \prec t \qquad \operatorname{coso}$$

$$\operatorname{coso} \qquad \operatorname{Sequent Calculus} \qquad \operatorname{Symbolic Execution} \qquad \operatorname{coso} \qquad \operatorname{ACalculus for 100\% Java CARD} \qquad \operatorname{Taclets} \qquad \operatorname{coso} \qquad \operatorname{Symbolic Execution} \qquad \operatorname{coso} \qquad \operatorname{Calculus for 100\% Java CARD} \qquad \operatorname{Taclets} \qquad \operatorname{coso} \qquad \operatorname{Coso} \qquad \operatorname{Symbolic Execution} \qquad \operatorname{coso} \qquad \operatorname{Calculus for 100\% Java CARD} \qquad \operatorname{Taclets} \qquad \operatorname{coso} \qquad \operatorname{Calculus for 100\% Java CARD} \qquad \operatorname{Taclets} \qquad \operatorname{coso} \qquad \operatorname{Calculus for 100\% Java CARD} \qquad \operatorname{Taclets} \qquad \operatorname{coso} \qquad \operatorname{Calculus for 100\% Java CARD} \qquad \operatorname{Taclets} \qquad \operatorname{coso} \qquad \operatorname{Calculus for 100\% Java CARD} \qquad \operatorname{Taclets} \qquad \operatorname{coso} \qquad \operatorname{Calculus for 100\% Java CARD} \qquad \operatorname{Calculus for 100\% Java CARD} \qquad \operatorname{Taclets} \qquad \operatorname{coso} \qquad \operatorname{Calculus for 100\% Java CARD} \qquad \operatorname{Taclets} \qquad \operatorname{Calculus for 100\% Java CARD} \qquad \operatorname{Taclets} \qquad \operatorname{Calculus for 100\% Java CARD} \qquad \operatorname{Taclets} \qquad \operatorname{Calculus for 100\% Java CARD} \qquad \operatorname{Calculuus for 100\% Java CARD} \qquad \operatorname{Calculus for$$

Bernhard Beckert - Deductive Verification of Object-Oriented Software

JAVA

VTSA, 24.–28.08.2015

Some Simple Sequent Rules



$$\mathsf{not_left} \ \frac{\Gamma \Longrightarrow A, \Delta}{\Gamma, !A \Longrightarrow \Delta}$$

$$\begin{array}{c} \text{imp_left} \ \ \hline \Gamma \Longrightarrow A, \Delta \quad \Gamma, B \Longrightarrow \Delta \\ \hline \Gamma, A \longrightarrow B \Longrightarrow \Delta \end{array}$$

close_goal
$$\overline{\Gamma, A \Longrightarrow A, \Delta}$$

$$\begin{array}{l} \text{IIlleft} & \frac{\Gamma, \forall \texttt{forall} \ t \ x; \phi, \ \{x/e\}\phi \Longrightarrow \Delta}{\Gamma, \forall \texttt{forall} \ t \ x; \phi \Longrightarrow \Delta} \end{array}$$

where e var-free term of type $t' \prec t$

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD

Taclets 00000 48/102

Bernhard Beckert – Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015

Some Simple Sequent Rules



not_left
$$\frac{\Gamma \Longrightarrow A, \Delta}{\Gamma, !A \Longrightarrow \Delta}$$

$$\text{imp_left} \ \ \frac{\Gamma \Longrightarrow A, \Delta \qquad \Gamma, B \Longrightarrow \Delta}{\Gamma, A \twoheadrightarrow B \Longrightarrow \Delta}$$

$$\begin{array}{c} \mathsf{close_goal} & \hline \\ \hline \Gamma, A \Longrightarrow A, \Delta \end{array} \qquad \qquad \\ \begin{array}{c} \mathsf{close_by_true} & \hline \\ \hline \\ \Gamma \Longrightarrow \mathrm{true}, \Delta \end{array}$$

all_left
$$\frac{\Gamma, \langle \text{forall } t \; x; \phi, \; \{x/e\}\phi \Longrightarrow \Delta}{\Gamma, \langle \text{forall } t \; x; \phi \Longrightarrow \Delta}$$

where e var-free term of type $t' \prec t$

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD

VTSA, 24.-28.08.2015

Taclets 00000 48/102

Some Simple Sequent Rules



not_left
$$\frac{\Gamma \Longrightarrow A, \Delta}{\Gamma, !A \Longrightarrow \Delta}$$

$$\text{imp_left} \ \ \frac{\Gamma \Longrightarrow A, \Delta \qquad \Gamma, B \Longrightarrow \Delta}{\Gamma, A \twoheadrightarrow B \Longrightarrow \Delta}$$

$$\label{eq:close_goal} \begin{tabular}{close_goal}{close_by_true} \begin{tabular}{close_by_true}{close_by_true} \$$

$$\mathsf{all_left} \ \frac{\Gamma, \backslash \mathsf{forall} \ t \ x; \phi, \ \{x/e\}\phi \Longrightarrow \Delta}{\Gamma, \backslash \mathsf{forall} \ t \ x; \phi \Longrightarrow \Delta}$$

where e var-free term of type $t' \prec t$

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD

Taclets 00000 48/102

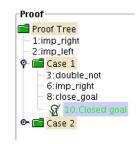
20

Bernhard Beckert - Deductive Verification of Object-Oriented Software



Proof tree

- Proof is tree structure with goal sequent as root
- Rules are applied from conclusion (old goal) to premisses (new goals)
- Rule with no premiss closes proof branch
- Proof is finished when all goals are closed



JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD

Taclets 00000 49/102

nar

1

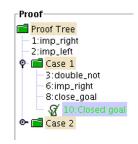
Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015



Proof tree

- Proof is tree structure with goal sequent as root
- Rules are applied from conclusion (old goal) to premisses (new goals)



JAVA CARD DL

Sequent Calculus 00000

Symbolic Execution

<ロト < 同ト < ヨト < ヨト -A Calculus for 100% JAVA CARD

Taclets 49/102

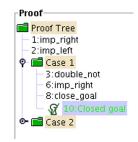
nac

Bernhard Beckert - Deductive Verification of Object-Oriented Software



Proof tree

- Proof is tree structure with goal sequent as root
- Rules are applied from conclusion (old goal) to premisses (new goals)
- Rule with no premiss closes proof branch
- Proof is finished when all goals are closed



JAVA CARD DL

Sequent Calculus

Symbolic Execution

Taclets 00000 49/102

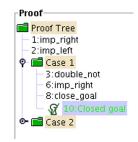
nac

Bernhard Beckert - Deductive Verification of Object-Oriented Software



Proof tree

- Proof is tree structure with goal sequent as root
- Rules are applied from conclusion (old goal) to premisses (new goals)
- Rule with no premiss closes proof branch
- Proof is finished when all goals are closed



JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD

シ ۹ (~ Taclets 00000 49/102

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015

イロト イポト イヨト イヨト

Part III

Program Verification with Dynamic Logic



Sequent Calculus

9 Rules for Programs: Symbolic Execution

0 A Calculus for 100% JAVA CARD

1 Taclets – KeY's Rule Description Language

JAVA CARD DL

Sequent Calculus

Symbolic Execution

ecution

A Calculus for 100% JAVA CARD 0000 VTSA, 24.–28.08.2015

イロト イポト イヨト イヨト

シ ۹ (~ Taclets 00000 50/102

Part III

Program Verification with Dynamic Logic



Rules for Programs: Symbolic Execution

Taclets – KeY's Rule Description Language

JAVA CARD DL

Sequent Calculus Bernhard Beckert - Deductive Verification of Object-Oriented Software

Symbolic Execution

イロト イポト イヨト イヨト A Calculus for 100% JAVA CARD VTSA, 24.-28.08.2015

nac Taclets 51/102



- Sequent rules for program formulas?
- What corresponds to top-level connective in a program?

The Active Statement in a Program

Sequent rules execute symbolically the active statement

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD 0000

Taclets 00000 52/102

Bernhard Beckert - Deductive Verification of Object-Oriented Software



- Sequent rules for program formulas?
- What corresponds to top-level connective in a program?

The Active Statement in a Program

l:{try{ i=0; j=0; } finally{ k=0; }}

Sequent rules execute symbolically the active statement

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD 0000

Taclets 00000 52/102

0

Bernhard Beckert - Deductive Verification of Object-Oriented Software



- Sequent rules for program formulas?
- What corresponds to top-level connective in a program?

The Active Statement in a Program

l:{try{ i=0; j=0; } finally{ k=0; }}

Sequent rules execute symbolically the active statement

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD 0000

Taclets 00000 52/102

0

Bernhard Beckert - Deductive Verification of Object-Oriented Software



- Sequent rules for program formulas?
- What corresponds to top-level connective in a program?

The Active Statement in a Program 1:{try{ i=0; j=0; } finally{ k=0; }} passive prefix π active statement i=0; rest ω 0 JAVA CARD DL Sequent Calculus Symbolic Execution A Calculus for 100% JAVA CARD Taclets 0000000

Bernhard Beckert – Deductive Verification of Object-Oriented Software

VTSA, 24.–28.08.2015

52/102



- Sequent rules for program formulas?
- What corresponds to top-level connective in a program?

The Active Statement in a Program 1:{try{ i=0; j=0; } finally{ k=0; }} passive prefix π active statement i=0; rest ω Sequent rules execute symbolically the active statement Sequent Calculus A Calculus for 100% JAVA CARD JAVA CARD DL Symbolic Execution Taclets 0000000 52/102

Bernhard Beckert - Deductive Verification of Object-Oriented Software

Rules for Symbolic Program Execution



If-then-else rule

$$\begin{array}{ll} \Gamma,B=\textit{true} \Longrightarrow \phi,\Delta & \Gamma,B=\textit{false} \Longrightarrow " \phi,\Delta \\ \hline \Gamma \Longrightarrow <\textit{if} \ (B) \ \{ \ p \ \} \ \textit{else} \ \{ \ q \ \} \ \omega > \phi,\Delta \end{array}"$$

$$\Gamma \Longrightarrow \langle v=y; y=y+1; x=v; \omega \rangle \phi, \Delta$$

$$\Gamma \Longrightarrow \{loc := val\} {<} \omega {>} \phi, \Delta$$

JAVA CARD DL

Sequent Calculus

Symbolic Execution 0000000

A Calculus for 100% JAVA CARD VTSA, 24.-28.08.2015

Bernhard Beckert - Deductive Verification of Object-Oriented Software

Taclets 53/102

DQ C

 \equiv

Rules for Symbolic Program Execution



If-then-else rule

$$\begin{array}{ccc} \Gamma,B=\textit{true} \Longrightarrow \phi,\Delta & \Gamma,B=\textit{false} \Longrightarrow " \phi,\Delta \\ \hline \Gamma \Longrightarrow < \texttt{if} \ (B) \ \{ \ p \ \} \ \texttt{else} \ \{ \ q \ \} \ \omega > \phi,\Delta \end{array}"$$

Complicated statements/expressions are simplified first, e.g.

$$\frac{\Gamma \Longrightarrow \langle v=y; y=y+1; x=v; \omega \rangle \phi, \Delta}{\Gamma \Longrightarrow \langle x=y++; \omega \rangle \phi, \Delta}$$

$$\Gamma \Longrightarrow \{loc := val\} {<} \omega {>} \phi, \Delta$$

$$\Gamma \Longrightarrow < loc=val; \ \omega > \phi, \Delta$$

JAVA CARD DL

Sequent Calculus

Symbolic Execution 0000000

A Calculus for 100% JAVA CARD VTSA, 24.-28.08.2015

Taclets 53/102

DQ C

 \equiv

Rules for Symbolic Program Execution



If-then-else rule

$$\begin{array}{ccc} \Gamma,B=\textit{true} \Longrightarrow \phi,\Delta & \Gamma,B=\textit{false} \Longrightarrow " \phi,\Delta \\ \hline \Gamma \Longrightarrow < \texttt{if} \ (B) \ \{ \ p \ \} \ \texttt{else} \ \{ \ q \ \} \ \omega > \phi,\Delta \end{array}"$$

Complicated statements/expressions are simplified first, e.g.

$$\frac{\Gamma \Longrightarrow \langle \mathbf{v}=\mathbf{y}; \ \mathbf{y}=\mathbf{y}+\mathbf{1}; \ \mathbf{x}=\mathbf{v}; \ \omega \rangle \phi, \Delta}{\Gamma \Longrightarrow \langle \mathbf{x}=\mathbf{y}+\mathbf{y}; \ \omega \rangle \phi, \Delta}$$

Simple assignment rule

$$\Gamma \Longrightarrow \{loc := val\} {<} \omega {>} \phi, \Delta$$

$$\Gamma \Longrightarrow {<} loc = val; \ \omega {>} \phi, \Delta$$

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD 0000 VTSA, 24.–28.08.2015

・ロト ・ 四 ト ・ 注 ト ・ 注 ト

シ ۹ (~ Taclets 00000 53/102

 \equiv

Extending DL by Explicit State Updates



Updates

explicit syntactic elements in the logic

Elementary Updates

$$\{ loc := val \} \phi$$

where (roughly)

lacksquare loc a program variable x, an attribute access o.attr, or an array access a[i]

val is same as loc, or a literal, or a logical variable

Parallel Updates

$$\{loc_1 := t_1 \mid \mid \cdots \mid \mid loc_n := t_n\}\phi$$

no dependency between the n components (but 'right wins' semantics)

20

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD 0000 VTSA, 24.–28.08.2015 Taclets 00000 54/102

Extending DL by Explicit State Updates



Updates

explicit syntactic elements in the logic

Elementary Updates

$$\{ loc := val \} \phi$$

where (roughly)

- *loc* a program variable x, an attribute access o.attr, or an array access a[i]
- *val* is same as *loc*, or a literal, or a logical variable

$$\{loc_1 := t_1 \mid | \cdots \mid | loc_n := t_n\}\phi$$

JAVA CARD DL

Sequent Calculus Bernhard Beckert - Deductive Verification of Object-Oriented Software

Symbolic Execution 0000000

A Calculus for 100% JAVA CARD VTSA, 24.-28.08.2015 Taclets 54/102

Extending DL by Explicit State Updates



Updates

explicit syntactic elements in the logic

Elementary Updates

$$\{ \underline{loc} := \underline{val} \} \phi$$

where (roughly)

- *loc* a program variable x, an attribute access *o.attr*, or an array access a[i]
- val is same as loc, or a literal, or a logical variable

Parallel Updates

$$\{loc_1 := t_1 \mid | \cdots | | loc_n := t_n\}\phi$$

no dependency between the n components (but 'right wins' semantics)

20

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD

Taclets 00000 54/102

Bernhard Beckert - Deductive Verification of Object-Oriented Software

Why Updates?



Updates are:

- lazily applied (i.e. substituted into postcondition)
- eagerly parallelised + simplified

Advantages

- no renaming required
- delayed/minimized proof branching (efficient aliasing treatment)

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD

ク Q (~ Taclets 00000 55/102

 \equiv

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015

Why Updates?



Updates are:

- *lazily applied* (i.e. substituted into postcondition)
- eagerly parallelised + simplified

Advantages

- no renaming required
- delayed/minimized proof branching (efficient aliasing treatment)

JAVA CARD DL

Sequent Calculus

Symbolic Execution 00000000

イロト 人間ト イヨト イヨト A Calculus for 100% JAVA CARD VTSA, 24.-28.08.2015

nac Taclets 55/102

 \equiv



= nar

56/102

x < y -> <int t=x; x=y; y=t;> y < x

JAVA CARD DL

Sequent Calculus

Symbolic Execution 00000000

< ロ > < 同 > < 臣 > < 臣 > -A Calculus for 100% JAVA CARD Taclets VTSA, 24.-28.08.2015



 $x < y \implies {t:=x} < x=y; y=t; y < x$ x < y -> <int t=x; x=y; y=t;> y < x

JAVA CARD DL

Sequent Calculus

Symbolic Execution

Taclets 00000 56/102

= nar



 $x < y \implies \{t:=x\} \{x:=y\} < y=t; > y < x$ $x < y \implies {t:=x} < x=y; y=t; y < x$ x < y -> <int t=x; x=y; y=t;> y < x

JAVA CARD DL

Sequent Calculus

Symbolic Execution

4 □ ▷ 4 ⓓ ▷ 4 ≧ ▷ 4 ≧ ▷ 4 ≧ ▷ ≧
A Calculus for 100% JAVA CARD
0000
VTSA, 24-28.08.2015

Taclets 00000 56/102

= nar



 $x < y \implies \{t:=x \mid | x:=y\} \{y:=t\} > y < x$ $x < y \implies \{t:=x\} \{x:=y\} < y=t; > y < x$ $x < y \implies \{t:=x\} < x=y; y=t; > y < x$ x < y -> <int t=x; x=y; y=t;> y < x

JAVA CARD DL

Sequent Calculus

Symbolic Execution

Taclets 00000 56/102

= nar



 $x < y \implies \{t:=x \mid | x:=y \mid | y:=x\} <> y < x$ $x < y \implies \{t:=x \mid | x:=y\} \{y:=t\} > y < x$ $x < y \implies \{t:=x\} \{x:=y\} < y=t; > y < x$ $x < y \implies {t:=x} < x=y; y=t; y < x$ x < y -> <int t=x; x=y; y=t;> y < x

JAVA CARD DL

Sequent Calculus

Symbolic Execution 00000000

イロト イポト イヨト イヨト A Calculus for 100% JAVA CARD Taclets VTSA, 24.-28.08.2015

Bernhard Beckert - Deductive Verification of Object-Oriented Software

56/102

= nar



$$\begin{array}{rcl} x < y & \implies & x < y \\ \vdots \\ x < y & \implies & \{x:=y \mid\mid y:=x\} <> \ y < x \\ \vdots \\ x < y & \implies & \{t:=x \mid\mid x:=y \mid\mid y:=x\} <> \ y < x \\ \vdots \\ x < y & \implies & \{t:=x \mid\mid x:=y\} \{y:=t\} <> \ y < x \\ \vdots \\ x < y & \implies & \{t:=x\} \{x:=y\} < y=t; > \ y < x \\ \vdots \\ x < y & \implies & \{t:=x\} < x=y; \ y=t; > \ y < x \\ \vdots \\ \implies & x < y \implies & (t:=x\} < x=y; \ y=t; > \ y < x \end{array}$$

JAVA CARD DL

Sequent Calculus

Symbolic Execution

Taclets 00000 56/102

= nar



$$\begin{array}{rcl} x < y & \Longrightarrow & x < y \\ & \vdots \\ x < y & \Longrightarrow & \{x:=y \mid\mid y:=x\} <> \ y < x \\ & \vdots \\ x < y & \Longrightarrow & \{t:=x \mid\mid x:=y \mid\mid y:=x\} <> \ y < x \\ & \vdots \\ x < y & \Longrightarrow & \{t:=x \mid\mid x:=y\} \{y:=t\} <> \ y < x \\ & \vdots \\ x < y & \Longrightarrow & \{t:=x\} \{x:=y\} < y=t; > \ y < x \\ & \vdots \\ x < y & \Longrightarrow & \{t:=x\} < x=y; \ y=t; > \ y < x \\ & \vdots \\ & \Rightarrow & x < y -> < int \ t=x; \ x=y; \ y=t; > \ y < x \end{array}$$

JAVA CARD DL

Sequent Calculus

Symbolic Execution 00000000

A Calculus for 100% JAVA CARD VTSA, 24.-28.08.2015

nac Taclets 56/102

Ξ

Program State Representation



Local program variables

Modeled as non-rigid constants

Heap

Modeled with theory of arrays:

	ightarrow Heap (the heap in the current state)
select:	$Heap \times Object \times Field \rightarrow Any$
	Hean & Object & Field & Any Hear

Heap axioms (excerpt)

 $\begin{aligned} select(store(h, o, f, x), o, f) &= x\\ select(store(h, o, f, x), u, f) &= select(h, u, f) \text{ if } o \neq u \end{aligned}$

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD 0000 VTSA, 24.–28.08.2015

< ロ > < 回 > < 回 > < 回 > < 回 > <

う ۹ (~ Taclets 00000 57/102

 \equiv

Program State Representation



Local program variables

Modeled as non-rigid constants

Heap

Modeled with theory of arrays:

$neap: \rightarrow neap$ (the neap in the current state	heap:	\rightarrow Heap (the heap in the current state)
---	-------	--

- select: $Heap \times Object \times Field \rightarrow Any$
- store: $Heap \times Object \times Field \times Any \rightarrow Heap$

Heap axioms (excerpt)

 $\begin{aligned} select(store(h, o, f, x), o, f) &= x\\ select(store(h, o, f, x), u, f) &= select(h, u, f) \text{ if } o \neq u \end{aligned}$

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD 0000 VTSA, 24,-28,08,2015

イロト イポト イヨト イヨト

Taclets 00000 57/102

= 900

Program State Representation



Local program variables

Modeled as non-rigid constants

Heap

Modeled with theory of arrays:

$meap$. $\rightarrow meap$ (the neap in the current state)	heap:	\rightarrow Heap	(the heap in the current state)
---	-------	--------------------	---------------------------------

- select: $Heap \times Object \times Field \rightarrow Any$
- $Heap \times Object \times Field \times Any \rightarrow Heap$ store:

Heap axioms (excerpt)

select(store(h, o, f, x), o, f) = xselect(store(h, o, f, x), u, f) = select(h, u, f) if $o \neq u$

JAVA CARD DL

Sequent Calculus

Symbolic Execution 00000000

A Calculus for 100% JAVA CARD VTSA, 24.-28.08.2015

nac Taclets 57/102

Bernhard Beckert - Deductive Verification of Object-Oriented Software

イロト イポト イヨト イヨト 二日

Handling Abrupt Termination



Abrupt termination handled by program transformations

Changing control flow = rearranging program parts

Example

TRY-THROW

$\Rightarrow \left\langle \begin{array}{c} \text{if (exc instanceof T)} \\ \text{{try {e=exc; r} finally {s}}} \\ \text{{else {s throw exc;}}} \\ \omega \end{array} \right\rangle \phi, \ \Delta$

$\Gamma \Longrightarrow \langle try\{throw exc; q\} catch(T e)\{r\} finally\{s\} \omega \rangle \phi, \Delta$

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD 0000 VTSA, 24–28,08,2015

Taclets 00000 58/102

Handling Abrupt Termination



• Abrupt termination handled by program transformations

Changing control flow = rearranging program parts

Example

TRY-THROW

$$\Gamma \Longrightarrow \left\langle \begin{array}{c} \text{if (exc instance of T)} \\ \text{{try {e=exc; r} finally {s}}} \\ \text{{else {s throw exc;}}} \\ \omega \end{array} \right\rangle \phi, \ \Delta$$

 $\Gamma \Longrightarrow \langle \texttt{try}\{\texttt{throw exc; q} \} \texttt{ catch}(\texttt{T e})\{\texttt{r}\} \texttt{ finally}\{\texttt{s}\} \ \omega \rangle \phi, \ \Delta$

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus f

A Calculus for 100% JAVA CARD 0000 VTSA, 24.–28.08.2015

・ロト ・ 同ト ・ ヨト ・ ヨト

シ ۹ (~ Taclets 00000 58/102

Handling Abrupt Termination



• Abrupt termination handled by program transformations

Changing control flow = rearranging program parts

Example

TRY-THROW

$$\Gamma \Longrightarrow \left\langle \begin{matrix} \pi \text{ if (exc instance of T)} \\ \{ \text{try {e=exc; r} finally {s}} \\ \text{else {s throw exc;}} \end{matrix} \right. \omega \end{matrix} \right\rangle \phi, \ \Delta$$

 $\Gamma \Longrightarrow \langle \pi \text{ try} \{ \text{throw exc; } q \} \text{ catch}(T e) \{ r \} \text{ finally} \{ s \} \omega \rangle \phi, \Delta$

JAVA CARD DL

Sequent Calculus

Symbolic Execution

ion A Calo

A Calculus for 100% JAVA CARD 0000 VTSA, 24.–28.08.2015

・ロト ・ 同 ト ・ ヨ ト ・ ヨ ト

シ ۹ (~ Taclets 00000 58/102

Part III

Program Verification with Dynamic Logic



Rules for Programs: Symbolic Execution

Taclets – KeY's Rule Description Language

JAVA CARD DL

Sequent Calculus

Symbolic Execution 0000000

イロト イポト イヨト イヨト A Calculus for 100% JAVA CARD

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015

Taclets 59/102

nac

Part III

Program Verification with Dynamic Logic

- A Calculus for 100% JAVA CARD
- Taclets KeY's Rule Description Language

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD Taclets VTSA, 24.-28.08.2015

Bernhard Beckert - Deductive Verification of Object-Oriented Software

イロト イポト イヨト イヨト

60/102

nac

Supported Java Features



method invocation with polymorphism/dynamic binding

- object creation and initialisation
- arrays
- abrupt termination
- throwing of NullPointerExceptions, etc.
- bounded integer data types
- transactions

All JAVA CARD language features are fully addressed in KeY

JAVA CARD DL

Sequent Calculus

Symbolic Execution

ecution

VTSA, 24.–28.08.2015

A Calculus for 100% JAVA CARD

0000

イロト 人間ト イヨト イヨト

Taclets 00000 61/102

= nar

Supported Java Features



method invocation with polymorphism/dynamic binding

- object creation and initialisation
- arrays
- abrupt termination
- throwing of NullPointerExceptions, etc.
- bounded integer data types
- transactions

All JAVA CARD language features are fully addressed in KeY

JAVA CARD DL

Sequent Calculus Bernhard Beckert - Deductive Verification of Object-Oriented Software

Symbolic Execution

A Calculus for 100% JAVA CARD 0000 VTSA, 24.-28.08.2015

nac Taclets 61/102

3



Ways to deal with Java features

- Program transformation, up-front
- Local program transformation, done by a rule on-the-fly
- Modeling with first-order formulas
- Special-purpose extensions of program logic

Pro: Feature needs not be handled in calculus Contra: Modified source code Example in KeY: Very rare: treating inner classes

JAVA CARD DL

Sequent Calculus

Symbolic Execution

c Execution

A Calculus for 100% JAVA CARD Taclets 0000 VTSA, 24.-28.08.2015 62/102

nac

3

< □ > < □ > < □ > < □ > < □ > <



Ways to deal with Java features

- Program transformation, up-front
- Local program transformation, done by a rule on-the-fly
- Modeling with first-order formulas
- Special-purpose extensions of program logic

Pro: Flexible, easy to implement, usable Contra: Not expressive enough for all features Example in KeY: Complex expression eval, method inlining, etc., etc.

JAVA CARD DL 00000000 Sequent Calculus

Symbolic Execution

cution A C

A Calculus for 100% JAVA CARD T 0000 VTSA, 24.–28.08.2015

Taclets 00000 62/102

nac



Ways to deal with Java features

- Program transformation, up-front
- Local program transformation, done by a rule on-the-fly
- Modeling with first-order formulas

Pro: No logic extensions required, enough to express most features Contra: Creates difficult first-order POs. unreadable antecedents Example in KeY: Dynamic types and branch predicates

JAVA CARD DI

Sequent Calculus

Symbolic Execution

0000

<ロト < 同ト < ヨト < ヨト -A Calculus for 100% JAVA CARD VTSA, 24.-28.08.2015

Taclets 62/102

nac



Ways to deal with Java features

- Program transformation, up-front
- Local program transformation, done by a rule on-the-fly
- Modeling with first-order formulas
- Special-purpose extensions of program logic

Pro: Arbitrarily expressive extensions possible Contra: Increases complexity of all rules Example in KeY: Method frames, updates

JAVA CARD DL

Sequent Calculus

Symbolic Execution

ecution A Ca

A Calculus for 100% JAVA CARD •••• VTSA, 24.–28.08.2015

<ロト < 同ト < ヨト < ヨト -

Taclets 00000 62/102

nac



Non-program rules

- first-order rules
- rules for data-types
- first-order modal rules
- induction rules

JAVA CARD DL

Sequent Calculus Bernhard Beckert - Deductive Verification of Object-Oriented Software

Symbolic Execution

A Calculus for 100% JAVA CARD 0000 VTSA, 24.-28.08.2015

STREES FR

Taclets 63/102

-, 2 C



Non-program rules

- first-order rules
- rules for data-types
- first-order modal rules
- induction rules
- Pules for reducing/simplifying the program (symbolic execution) Replace the program by
 - case distinctions (proof branches) and
 - sequences of updates
- 3 Rules for handling loops
 - using loop invariants
 - using induction
- In the second second
- Update simplification

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD 0000 VTSA, 24.–28.08.2015



Non-program rules

- first-order rules
- rules for data-types
- first-order modal rules
- induction rules
- 2 Rules for reducing/simplifying the program (symbolic execution) Replace the program by
 - case distinctions (proof branches) and
 - sequences of updates
- 3 Rules for handling loops
 - using loop invariants
 - using induction

JAVA CARD DL

Sequent Calculus Bernhard Beckert - Deductive Verification of Object-Oriented Software

Symbolic Execution

A Calculus for 100% JAVA CARD 0000 VTSA, 24.-28.08.2015

-, 2 P Taclets 63/102



Non-program rules

- first-order rules
- rules for data-types
- first-order modal rules
- induction rules
- 2 Rules for reducing/simplifying the program (symbolic execution) Replace the program by
 - case distinctions (proof branches) and
 - sequences of updates
- 3 Rules for handling loops
 - using loop invariants
 - using induction

Rules for replacing a method's invocation by the method's contract

JAVA CARD DL

Sequent Calculus Bernhard Beckert - Deductive Verification of Object-Oriented Software

Symbolic Execution

A Calculus for 100% JAVA CARD 0000 VTSA, 24.-28.08.2015

+,20 Taclets 63/102



Non-program rules

- first-order rules
- rules for data-types
- first-order modal rules
- induction rules
- Pules for reducing/simplifying the program (symbolic execution) Replace the program by
 - case distinctions (proof branches) and
 - sequences of updates
- 3 Rules for handling loops
 - using loop invariants
 - using induction

In the second second

Opdate simplification

JAVA CARD DL

Symbolic Execution

A Calculus for 100% JAVA CARD

Part III

Program Verification with Dynamic Logic

- A Calculus for 100% JAVA CARD
- Taclets KeY's Rule Description Language

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD Taclets 0000 VTSA, 24.-28.08.2015

nac 1

64/102

イロト イポト イヨト イヨト

Part III

Program Verification with Dynamic Logic

- 7 JAVA CARD DL
- 8 Sequent Calculus
- 9 Rules for Programs: Symbolic Execution
 - A Calculus for 100% JAVA CARD
- 1 Taclets KeY's Rule Description Language

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD

Taclets 00000 65/102

= nar

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015

イロト イポト イヨト イヨト

Taclets: KeY's Rule Description Language



Taclets

- represent sequent calculus rules in KeY
- use a simple text-based format
- are descriptive, but with operational flavor
- are not a tactic metalanguage

Proof Goals Proof Search Strategy Rules
Proof Tree
🗉 💼 Use Axiom
Imal Execution (logArray != null)
🗆 🖬 Invariant Initially Valid
CUT: self.logArray[0]. <created> = TRUE TR</created>
CUT: self.logArray[0]. <created> = TRUE FA</created>
🗉 💼 Body Preserves Invariant
Normal Execution (logArray != null)
□ 💼 i_0 <= 2 TRUE
Normal Execution (logArray != null)
Normal Execution (I_arr != null)
🗉 💼 Post
🗉 💼 Post
$\Box \equiv \operatorname{result}_1 <= -1 + \operatorname{result}_0$
유 790 OPEN GOAL
□ mesult_1 <= -1 + result_0
Exceptional Post
🖽 💼 Pre
Image: Second
Exceptional Post
🖽 📰 Pre
Image: Null reference (Ir = null)
Null Reference (I_arr = null)
Index Out of Bounds (I_arr != null,
Image: Second
i_0 <= 2 FALSE
Image: Second
🗉 💼 Use Case
ନ 179:OPEN GOAL
Image: Second Seco
Index Out of Bounds (logArray != null, but 0 Out
🗉 💼 Show Axiom Satisfiability

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD Taclets VTSA, 24.-28.08.2015

Bernhard Beckert - Deductive Verification of Object-Oriented Software

・ロト ・ 同ト ・ ヨト

00000 66/102

nac 1



andLeft
$$\frac{\Gamma, A, B \Longrightarrow \Delta}{\Gamma, A \& B \Longrightarrow \Delta}$$

JAVA CARD DL

Sequent Calculus Bernhard Beckert - Deductive Verification of Object-Oriented Software

Symbolic Execution

A Calculus for 100% JAVA CARD

Taclets 00000 67/102

VTSA, 24.-28.08.2015



andLeft
$$\frac{\Gamma, A, B \Longrightarrow \Delta}{\Gamma, A \& B \Longrightarrow \Delta}$$

Taclet

```
andLeft {
    find (A \& B ==>)
    \replacewith ( A, B ==>)
};
```

Unique name

JAVA CARD DL

Sequent Calculus Bernhard Beckert - Deductive Verification of Object-Oriented Software

Symbolic Execution

A Calculus for 100% JAVA CARD

Taclets 00000 67/102

VTSA, 24.-28.08.2015



andLeft
$$\frac{\Gamma, A, B \Longrightarrow \Delta}{\Gamma, A \& B \Longrightarrow \Delta}$$

Taclet

```
andLeft {
    \find ( A & B ==> )
    \replacewith ( A, B ==>)
};
```

- Unique name
- Find expression:
 - Formula (Term) to be modified
 - Sequent arrow ==> formula must occur top level and on the corresponding side of the sequent.
 - Goal Description: describes new sequent

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD

VTSA, 24.-28.08.2015

Taclets ○●○○○ 67/102



andLeft
$$\frac{\Gamma, A, B \Longrightarrow \Delta}{\Gamma, A \& B \Longrightarrow \Delta}$$

Taclet

```
andLeft {
    \find ( A & B ==> )
    \replacewith ( A, B ==>)
};
```

- Unique name
- Find expression:
 - Formula (Term) to be modified
 - Sequent arrow ==> formula must occur top level and on the corresponding side of the sequent.

Goal Description: describes new sequent

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD 0000

Taclets ○●○○○ 67/102

20

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015



andLeft
$$\frac{\Gamma, A, B \Longrightarrow \Delta}{\Gamma, A \& B \Longrightarrow \Delta}$$

Taclet

```
andLeft {
    \find ( A & B ==> )
    \replacewith ( A, B ==>)
};
```

- Unique name
- Find expression:
 - Formula (Term) to be modified
 - Sequent arrow ==> formula must occur top level and on the corresponding side of the sequent.

Goal Description: describes new sequent

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD 0000

Taclets ○●○○○ 67/102

20

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015



Some rules are only sound in a certain context

Taclet

```
modusPonens {
  assumes (A ==>)
  find (A \rightarrow B ==>)
  \replacewith( B ==> )
};
```

JAVA CARD DL

Symbolic Execution

A Calculus for 100% JAVA CARD Taclets VTSA, 24.-28.08.2015

イロト 人間ト イヨト イヨト

Bernhard Beckert - Deductive Verification of Object-Oriented Software

00000 68/102

3 nac



Some rules are only sound in a certain context

Taclet

```
modusPonens {
  \assumes ( A ==> )
  \find ( A -> B ==> )
  \replacewith( B ==> )
};
```

JAVA CARD DL

Symbolic Execution

ition A C

A Calculus for 100% JAVA CARD Taclets 0000 VTSA, 24.–28.08.2015 68/102

= nar

イロト 人間ト イヨト イヨト



Some rules are only sound in a certain context

Taclet

```
modusPonens {
  assumes (A ==>)
  find (A \rightarrow B ==>)
  \replacewith( B ==> )
};
```

JAVA CARD DL

Sequent Calculus

Symbolic Execution

A Calculus for 100% JAVA CARD VTSA, 24.-28.08.2015

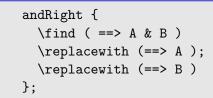
イロト 人間ト イヨト イヨト

Taclets 00000 68/102

3 nac



Proof Splitting: and Right



Variable Conditions: allRight

 $\Gamma \Longrightarrow A, \Delta \quad \Gamma \Longrightarrow B, \Delta$

 $\Gamma \Longrightarrow A \& B, \Delta$

$\frac{\Gamma \Longrightarrow \{x/c}{\Gamma \Longrightarrow \forall T x}$	Φ, Δ ; Φ, Δ , c new	\varcond(\ ne	<pre>\forall x;phi) w(c,\dependingOn(ph n (==> {\subst x;c}</pre>	phi)
			《曰》《卽》《臣》《臣》 臣	500
JAVA CARD DL 00000000 Bernhard Beckert – Deduc	Sequent Calculus 00000 ctive Verification of Object-Orie	Symbolic Execution	A Calculus for 100% JAVA CARD 0000 VTSA, 24.–28.08.2015	Taclets 000●0 69/102

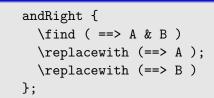


69/102

Proof Splitting: and Right

 $\Gamma \Longrightarrow A, \Delta \quad \Gamma \Longrightarrow B, \Delta$

 $\Gamma \Longrightarrow A \& B, \Delta$



Variable Conditions: allRight allRight { \find (==> \forall x;phi) $\frac{\Gamma \Longrightarrow \{x/c\}\Phi, \Delta}{\Gamma \Longrightarrow \forall T x; \Phi, \Delta}, \mathsf{c new}$ \varcond(\new(c,\dependingOn(phi))) \replacewith (==> {\subst x;c}phi) }; 1 nac JAVA CARD DL Sequent Calculus Symbolic Execution A Calculus for 100% JAVA CARD Taclets 00000

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.–28.08.2015

Taclets for Program Transformations π if (exc == null) { $\Gamma \Longrightarrow \left\langle \begin{array}{c} \text{ try{ throw new NPE(); catch(T e) {r};} \\ \text{ } \text{ else if (exc instance of T) {e=exc; r}} \right\rangle \phi$ else throw exc; ω $\Gamma \Longrightarrow \langle \pi \text{ try} \{ \text{throw exc; } q \} \text{ catch}(T e) \{ r \}; \omega \rangle \phi$ \find (<.. try { throw #se; #slist } catch (#t #v0) { #slist1 } ...> post) \replacewith (<... if (#se == null) { try { throw new NullPointerException(); } catch (#t #v0) { #slist1 } } else if (#se instanceof #t) { #t #v0 = (#t) #se;#slist1 } else throw #se; ...> post) JAVA CARD DL Sequent Calculus Symbolic Execution A Calculus for 100% JAVA CARD Taclets 00000

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015

Part IV

Verifying Information-Flow Properties

Information Flow

Is Formalisation in DL



Objects and Information Flow

Information Flow

Formalisation in DL

Objects and Information Flow

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015

イロト イポト イヨト イヨト

71/102

nac

Э

Part IV

Verifying Information-Flow Properties

Information Flow

3 Formalisation in DL

14

Objects and Information Flow

Information Flow •0000000 Formalisation in DL

Objects and Information Flow

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015

イロト イヨト イヨト

72/102

nac

1

Language-based Information Security



Secret and public information

Partitioning of the set of program variables into

- variables which contain confidential information ("high variables")
 - NOT observable by the attacker -
- variables which contain non-confidential information ("low variables")
 observable by the attacker –

Informal definition of non-interference

A program is secure, if the initial values of the high variables do not interfere with the final values of the low variables.

 Information Flow
 Formalisation in DL

 ○●○○○○○○○
 ○○○○○

 Bernhard Beckert – Deductive Verification of Object-Oriented Software

Objects and Information Flow

VTSA, 24.-28.08.2015

Language-based Information Security



Secret and public information

Partitioning of the set of program variables into

- variables which contain confidential information ("high variables")
 - NOT observable by the attacker -
- variables which contain non-confidential information ("low variables")
 observable by the attacker –

Informal definition of non-interference

A program is secure, if the initial values of the high variables do not interfere with the final values of the low variables.

Information Flow Formalisation in DL 00000000 Bernhard Beckert – Deductive Verification of Object-Oriented Software - ▲日 > ▲ 国 > ▲ 国 > ▲ 国 > ▲ 国 > ▲ 国 > ▲ 国 > ▲ 国 >

Objects and Information Flow

VTSA, 24.-28.08.2015



Note

Sequential Java programs Termination not considered

Which methods are secure?

Information Flow

Formalisation in DL

・ロト・日・・日・・日・・日・ シック

Objects and Information Flow

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015



Note

Sequential Java programs Termination not considered

Which methods are secure?

```
void m_1() {
 low = high;
}
 if (high > 0) {low = 1;}
 else {low = 2:}:
                           Formalisation in DL
```

Information Flow 00000000 Bernhard Beckert - Deductive Verification of Object-Oriented Software

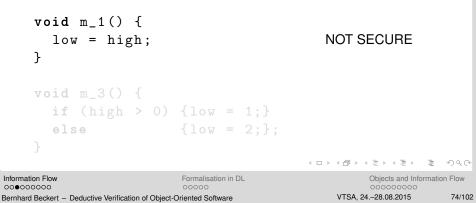
Objects and Information Flow



Note

Sequential Java programs Termination not considered

Which methods are secure?

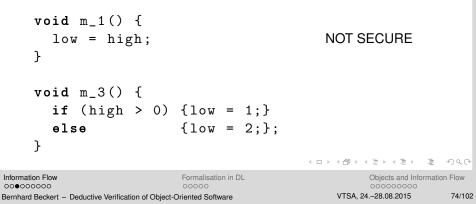




Note

Sequential Java programs Termination not considered

Which methods are secure?

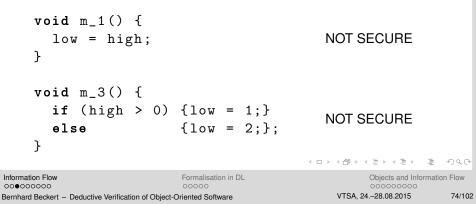




Note

Sequential Java programs Termination not considered

Which methods are secure?





Which methods are secure?

```
void m_4() {
    high = 0;
    low = high;
}
void m_5() {
    low = high;
    low = low-high;
}
```

void m_6() {
 if (false) low = high;

Information Flow

Formalisation in DL

・・

VTSA, 24.–28.08.2015

Objects and Information Flow

75/102

Which methods are secure?

```
void m_4() {
    high = 0;
    low = high;
}
```

```
void m_5() {
   low = high;
   low = low-high;
}
```

void m_6() {
 if (false) low = high;

Information Flow

Formalisation in DL



SECURE

Bernhard Beckert – Deductive Verification of Object-Oriented Software

VTSA, 24.–28.08.2015

Examples

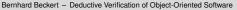
Which methods are secure?

```
void m_4() {
    high = 0;
    low = high;
}
void m_5() {
    low = high;
    low = low-high;
}
```

Information Flow

Formalisation in DL

Objects and Information Flow



0000000 VTSA, 24.–28.08.2015

◆□▶ ◆□▶ ◆ □▶ ◆ □▶ ● ● ● ● ● ●

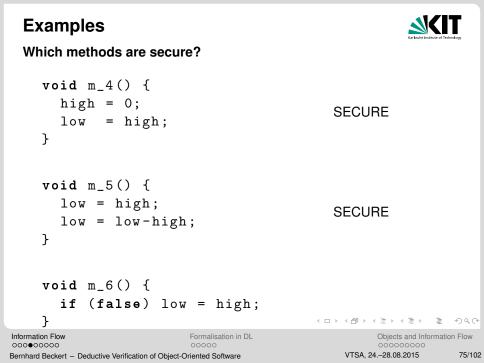
SECURE

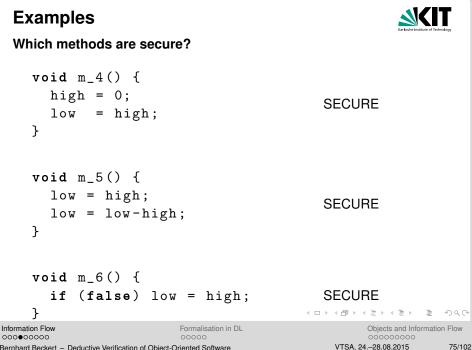


Examples Which methods are secure? void $m_4()$ { high = 0;SECURE low = high; } **void** m_5() { low = high; SECURE low = low - high;} if (false) low = high; ◆□▶ ◆□▶ ◆ □▶ ◆ □▶ ● ● ● ● ● ● Information Flow Formalisation in DL Objects and Information Flow 000000000

Bernhard Beckert – Deductive Verification of Object-Oriented Software

VTSA, 24.–28.08.2015





Information-flow Analysis Approaches



5900

76/102

performance

Informa 0000

Bernhar

A				
security type systems				
graph-theoretical rea on dependence grap				
	dependency g by ghost code			
	approximative c for Hoare style I			
	Hoar	alization in e style logics elf-composition		
		formalization in higher order logic		
		precision / ex	► kpressiveness	
I		< □ > < ♂ >		~ ? ~
tion Flow	Formalisation in DL		Objects and Informa	tion Flow
d Beckert – Deductive Verification of Object	-Oriented Software	VTSA, 2	428.08.2015	76/1

Information-flow Analysis Approaches



performance





Definition (Low-equivalence on states)

Two states are low-equivalent if they assign the same values to low variables.

Definition (Non-interference)

Starting P in two arbitrary low-equivalent states results in two final states that are also low-equivalent.

Information Flow

Formalisation in DL

・ロト・日本・ モー・ モー うくの

Objects and Information Flow

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015



Definition (Low-equivalence on states)

Two states are low-equivalent if they assign the same values to low variables.

Definition (Non-interference)

Starting P in two arbitrary low-equivalent states results in two final states that are also low-equivalent.

Information Flow

Formalisation in DL

・ロト・白ト・ヨト・ヨー うへつ

Objects and Information Flow

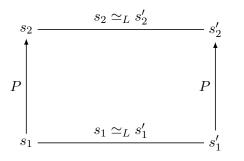
Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015

Non-interference



- P a program
- L the set of low variables



where

$$s_i \simeq_L s'_i \quad \Leftrightarrow \quad \forall v \in L \ (v^{s_i} = v^{s'_i})$$

Information Flow 000000000

Formalisation in DL

Э Objects and Information Flow

3

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015

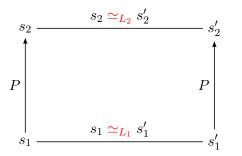
79/102

nar

Non-interference



- P a program
- L_1, L_2 sets of low variables



where

$$s_i \simeq_{L_i} s'_i \quad \Leftrightarrow \quad \forall v \in L_i \ (v^{s_i} = v^{s'_i})$$

Information Flow

Formalisation in DL

Objects and Information Flow

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015

イロト イポト イヨト イヨト

80/102

nac

Э



Encoding with alternating guantifiers

For all low input values in_l , there exist low output values r such that for all high input values in_h , if we assign the values in_l to the program variables low and in_h to the program variables high, then after execution of P the values of low are r.

$$\forall in_l \exists r \forall in_h (\{low := in_l \mid | high := in_h\}[P] low = r)$$

Information Flow

Formalisation in DI 00000 Bernhard Beckert - Deductive Verification of Object-Oriented Software

- P Objects and Information Flow

VTSA, 24.-28.08.2015

81/102

+120



Encoding with alternating guantifiers

For all low input values in_l , there exist low output values r such that for all high input values in_h , if we assign the values in_l to the program variables low and in_h to the program variables high, then after execution of P the values of low are r.

$$\forall in_l \exists r \forall in_h (\{low := in_l \mid | high := in_h\}[P] low = r)$$

Problem

Not suitable for automatic verification → instantiation of existential quantifier difficult.

Information Flow

Formalisation in DI 00000 Bernhard Beckert - Deductive Verification of Object-Oriented Software

Objects and Information Flow VTSA, 24.-28.08.2015 81/102

Non-Interference in JavaDL (Version 2)



Encoding with self-composition

Running two instances of P on the same low values but on arbitrary high values results in low variables which have the same values.

$$\begin{split} \forall in_l \forall in_h^1 \forall in_h^2 \forall out_l^1 \forall out_l^2 \ \{low := in_l\}(\\ \{high := in_h^1\}[P]out_l^1 = low \\ \land \ \{high := in_h^2\}[P]out_l^2 = low \\ \rightarrow out_l^1 = out_l^2 \end{split}$$

Information Flow

00000 Bernhard Beckert - Deductive Verification of Object-Oriented Software

Formalisation in DI

Objects and Information Flow

VTSA, 24.-28.08.2015

・ロト ・ 同 ト ・ ヨ ト ・ ヨ ト

82/102

DQ (P

Declassification



Intuition

Let T(high, low) be a term.

The only thing the attacker is allowed to learn about the secret inputs is the value of T in the initial state.

Information Flow

Formalisation in DI 00000 Bernhard Beckert - Deductive Verification of Object-Oriented Software

 < □ > < □ > < □ > < □ > < □ > < □ > MQ P

Objects and Information Flow

VTSA, 24.-28.08.2015

Declassification



Intuition

Let T(high, low) be a term.

The only thing the attacker is allowed to learn about the secret inputs is the value of T in the initial state.

Definition (Non-interference with declassification)

Starting P in two arbitrary low-equivalent states coinciding in the value of T results in two final states that are also low-equivalent.

Information Flow

Formalisation in DI 00000 Bernhard Beckert - Deductive Verification of Object-Oriented Software

イロト イポト イヨト イヨト 3 DQ C

Objects and Information Flow

VTSA, 24.-28.08.2015

Declassification in JavaDL



Encoding non-interference with declassification

Running two instances of P on the same low values and arbitrary high values coinciding on T results in low variables which have the same values.

$$\begin{split} \forall in_l \forall in_h^1 \forall in_h^2 \forall out_l^1 \forall out_l^2 \ \{low := in_l\} (\\ \{high := in_h^1\}T = \{high := in_h^2\}T \\ \land \{high := in_h^1\}[P]out_l^1 = low \\ \land \{high := in_h^2\}[P]out_l^2 = low \\ \rightarrow out_l^1 = out_l^2 \end{split}$$

Information Flow

Formalisation in DI 00000 Bernhard Beckert - Deductive Verification of Object-Oriented Software

Objects and Information Flow 84/102

VTSA, 24.-28.08.2015

DEMO

Verifying Information-flow Properties with the KeY Tool

Information Flow

Formalisation in DL

Objects and Information Flow

VTSA, 24.-28.08.2015

イロト イポト イヨト イヨト

85/102

nac

Э



Leakage by aliasing

```
void m() {
    C c1 = new C(); // new obj
    C c2 = c1; // alias
    c2.x = high;
    low = c1.x;
}
```

Information Flow

Formalisation in DL

Objects and Information Flow •0000000

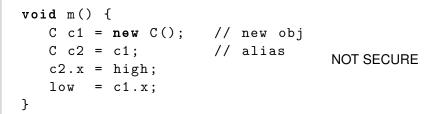
◆□▶ ◆□▶ ◆三▶ ◆三▶ → □ → へへや

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.–28.08.2015 8



Leakage by aliasing



Information Flow	
000000000	

Objects and Information Flow 00000000 VTSA, 24.-28.08.2015

イロト イポト イヨト イヨト

Bernhard Beckert - Deductive Verification of Object-Oriented Software

86/102

nac 3



Object creation and object identity

```
if (high>0) {
    low1 = new C();
    low2 = new C();
} else {
    low2 = new C();
    low1 = new C();
}
```

Assumption

- References are opaque
- Only comparison of objects by == is observable

Information Flow

Formalisation in DL

Objects and Information Flow 00000000 VTSA, 24.–28.08.2015 87/102

,20



Object creation and object identity

```
(high>0) {
if
    low1 = new C();
    low2 = new C();
} else {
    low2 = new C():
    low1 = new C();
}
```

SECURE

Bernhard Beckert - Deductive Verification of Object-Oriented Software

Information Flow

Formalisation in DL

Objects and Information Flow 00000000 VTSA, 24.-28.08.2015

87/102

,20



Object creation and object identity

```
(high>0) {
if
    low1 = new C();
    low2 = new C();
} else {
    low2 = new C():
    low1 = new C();
}
```

SECURE

Assumption

- References are opaque
- Only comparison of objects by == is observable

Information Flow

Formalisation in DL

Objects and Information Flow 00000000 VTSA, 24.-28.08.2015

87/102



Object creation and object identity

```
low1 = new C();
low2 = new C();
if (high>0) { low1 = low2; }
```

if (high>0) { low = new C() }

Information Flow

Formalisation in DL

Objects and Information Flow

◆□▶ ◆□▶ ◆ □▶ ◆ □▶ ● ● ● ● ● ●

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.-28.08.2015 88/102



Object creation and object identity

```
low1 = new C();
low2 = new C();
if (high>0) { low1 = low2; }
```

```
if (high>0) { low = new C() }
```

Information Flow

Formalisation in DI

Objects and Information Flow 000000000 VTSA, 24.-28.08.2015 88/102

◆□▶ ◆□▶ ◆ □▶ ◆ □▶ ● ● ● ● ● ●

NOT SECURE



Object creation and object identity

```
low1 = new C();
low2 = new C();
                                    NOT SECURE
if (high>0) { low1 = low2; }
```

```
if (high>0) { low = new C() }
```

Information Flow

Formalisation in DI

Objects and Information Flow 000000000 VTSA, 24.-28.08.2015

◆□▶ ◆□▶ ◆ □▶ ◆ □▶ ● ● ● ● ● ●

Bernhard Beckert - Deductive Verification of Object-Oriented Software



Object creation and object identity

```
low1 = new C();
low2 = new C();
                                    NOT SECURE
if (high>0) { low1 = low2; }
```

```
(high>0) { low = new C() }
                                 NOT SECURE
if
```

Information Flow

Formalisation in DI

Objects and Information Flow 000000000 VTSA, 24.-28.08.2015 88/102

◆□▶ ◆□▶ ◆ □▶ ◆ □▶ ● ● ● ● ● ●



Idea

ISOMORPHIC object structures in low variables **IDENTITY NOT required**

Instead of

$$s_i \simeq_{L_i} s'_i \quad \Leftrightarrow \quad \forall v \in L_i \ (v^{s_i} = v^{s'_i})$$

use

$$s_i \simeq_{L_i}^{\pi_i} s'_i \quad \Leftrightarrow \quad \forall v \in L_i \ (\pi_i(v^{s_i}) = v^{s'_i})$$

where

π_1, π_2 are compatible

i.e.

 $\pi_1(o) = \pi_2(o)$ if *o* observable in both s_1 and s_2

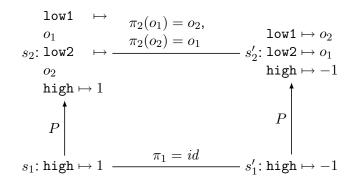
Information Flow

Formalisation in DL Bernhard Beckert - Deductive Verification of Object-Oriented Software

イロト イポト イヨト イヨト nac Э

> Objects and Information Flow 000000000 VTSA, 24.-28.08.2015 89/102





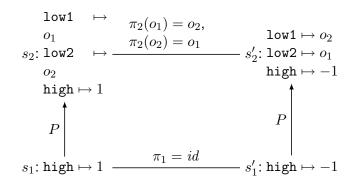
Information Flow

Formalisation in DL

・ ロ ト ・ 同 ト ・ 三 ト ・ nar 1

> Objects and Information Flow 000000000 VTSA, 24.-28.08.2015 90/102





Secure because o_1, o_2 not observable in s_1

Information Flow

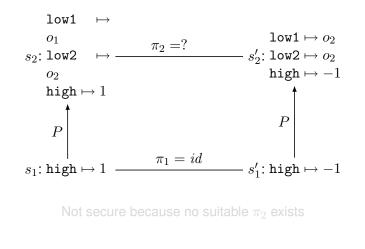
Formalisation in DL

<ロト < □ ト < □ ト < 三 ト < 三 ト < 三 ト 三 少 < で Objects and Information Flow 0000●0000

VTSA, 24.-28.08.2015

90/102





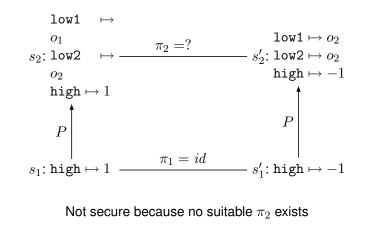
Information Flow

Formalisation in DL

・ ロ ト 4 配 ト 4 重 ト 4 画 ト 4 の 4 の ト

Objects and Information Flow 000000000 VTSA, 24.–28.08.2015 91/102





Information Flow

Formalisation in DL Bernhard Beckert - Deductive Verification of Object-Oriented Software

Objects and Information Flow 0000000000

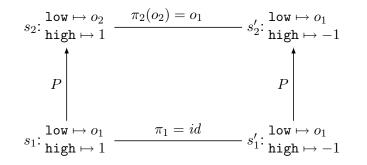
VTSA, 24.-28.08.2015

< D > < P > < E > < E</p>

91/102

nar





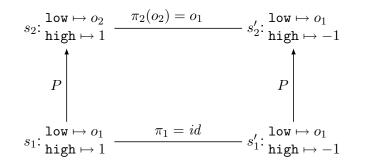
Information Flow

Formalisation in DL

nar

> Objects and Information Flow 0000000000 VTSA, 24.-28.08.2015 92/102





Not secure because o_1 observable in s_1 and $\pi_1(o_1) \neq \pi_2(o_1)$

Information Flow

Formalisation in DL

Objects and Information Flow

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.–28.08.2015

92/102

nar



Optimisations

- L_1, L_2 sequences of low terms (instead of sets of variables)
- π_1 can be fixed to be id

Information Flow

Formalisation in DL

Objects and Information Flow 0000000●0

Bernhard Beckert - Deductive Verification of Object-Oriented Software

VTSA, 24.–28.08.2015

イロト イポト イヨト イヨト

93/102

= nar

DEMO

Objects and Information-flow with the KeY Tool

Information Flow

Formalisation in DL

Objects and Information Flow 00000000 VTSA, 24.–28.08.2015 94/102

nac

Э

イロト イポト イヨト イヨト

Part V Wrap Up



15 Further Usage of Verification Technology



Directions of Current Research in KeY

Further Usage of Verification Technology

Bernhard Beckert - Deductive Verification of Object-Oriented Software

イロト イポト イヨト イヨト nac Э

> Directions of Current Research in KeY VTSA, 24.-28.08.2015 95/102

Part V Wrap Up

15 Further Usage of Verification Technology

Directions of Current Research in KeY

Further Usage of Verification Technology ooo Bernhard Beckert – Deductive Verification of Object-Oriented Software <ロ><日><日><日><日><日><日><日><日><日><日><日><日><日<</p>

Further Usage of Verification Technology



- Verification performs deep Program Analysis
- Information in (partial) proofs usable for other purposes

Further Usage of Verification Technology ●○○ Bernhard Beckert – Deductive Verification of Object-Oriented Software



Specification- and code-based approach

- Achieve strong hybrid coverage criteria
- Exploit strong correspondence: proof branches ↔ program execution paths
- Each leaf of (partial) proof branch contains constraint on inputs resulting in

corresponding path condition

Further Usage of Verification Technology ••• Bernhard Beckert – Deductive Verification of Object-Oriented Software ▲ロ▶▲圖▶▲≣▶▲≣▶ ≣ のQの



- Specification- and code-based approach
- Achieve strong hybrid coverage criteria
- Exploit strong correspondence: proof branches ↔ program execution paths
- Each leaf of (partial) proof branch contains constraint on inputs resulting in
 - corresponding path condition

Further Usage of Verification Technology ● ○ Bernhard Beckert – Deductive Verification of Object-Oriented Software



- Specification- and code-based approach
- Achieve strong hybrid coverage criteria
- Exploit strong correspondence: proof branches ↔ program execution paths

 Each leaf of (partial) proof branch contains constraint on inputs resulting in corresponding path condition

Further Usage of Verification Technology ●●○ Bernhard Beckert – Deductive Verification of Object-Oriented Software 

- Specification- and code-based approach
- Achieve strong hybrid coverage criteria
- Exploit strong correspondence: proof branches ↔ program execution paths
- Each leaf of (partial) proof branch contains constraint on inputs resulting in corresponding path condition

Further Usage of Verification Technology ●●○ Bernhard Beckert – Deductive Verification of Object-Oriented Software - イロト (四) (日) (日) (日) (日) (日)

Part V

Wrap Up



15 Further Usage of Verification Technology

Directions of Current Research in KeY

Further Usage of Verification Technology 000 Bernhard Beckert - Deductive Verification of Object-Oriented Software \equiv $\rightarrow \equiv \rightarrow$ nac

Part V Wrap Up



Further Usage of Verification Technology



Directions of Current Research in KeY

Further Usage of Verification Technology

Bernhard Beckert - Deductive Verification of Object-Oriented Software

< ロ > < 団 > < 臣 > < 臣 > < 臣 > < 臣 < つ < ぐ</p>



Topics

Scalability (combine with light-weight techniques)

- Usability (support user in understanding proof state)
- Concurrency and distribution
- Information-flow / security properties
- Application: eVoting

Further Usage of Verification Technology
000
Bernhard Beckert – Deductive Verification of Object-Oriented Software

くりっか 加 マルボマス 日マ きょうしょ



Topics

- Scalability (combine with light-weight techniques)
- Usability (support user in understanding proof state)
- Concurrency and distribution
- Information-flow / security properties
- Application: eVoting

F	urther Usage of Verification Technology
0	000
В	ernhard Beckert - Deductive Verification of Object-Oriented Software

▲ロ ▶ ▲ 圕 ▶ ▲ ≣ ▶ ▲ ≣ ▶ ■ の Q @ Directions of Current Research in KeY

> •O VTSA, 24.–28.08.2015 101/102



Topics

- Scalability (combine with light-weight techniques)
- Usability (support user in understanding proof state)
- Concurrency and distribution
- Information-flow / security properties
- Application: eVoting

Further Usage of Verification Technology

Bernhard Beckert - Deductive Verification of Object-Oriented Software



Topics

- Scalability (combine with light-weight techniques)
- Usability (support user in understanding proof state)
- Concurrency and distribution
- Information-flow / security properties

Further Usage of Verification Technology

Bernhard Beckert - Deductive Verification of Object-Oriented Software

< ∃ > Directions of Current Research in KeY . VTSA, 24.-28.08.2015 101/102

DQ C



Topics

- Scalability (combine with light-weight techniques)
- Usability (support user in understanding proof state)
- Concurrency and distribution
- Information-flow / security properties
- Application: eVoting

Further Usage of Verification Technology

Bernhard Beckert - Deductive Verification of Object-Oriented Software

 $\leftarrow \equiv \rightarrow$ Directions of Current Research in KeY . VTSA, 24.-28.08.2015 101/102

DQ C

 </

THE END

(for now)

Further Usage of Verification Technology

Bernhard Beckert - Deductive Verification of Object-Oriented Software

 $\exists \rightarrow$ Directions of Current Research in KeY 0. VTSA, 24.-28.08.2015 102/102

SQR

Image: Image:

- E