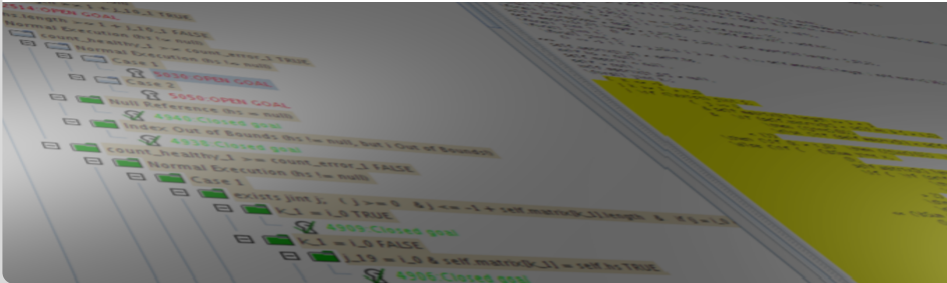


# PSE – Einführungsveranstaltung

Prof. Bernhard Beckert, Thorsten Borner, Daniel Bruns | 30. April 2012

Institut für Theoretische Informatik – Anwendungsorientierte formale Verifikation



Umfang: ca. 20 Seiten

Inhalte:

- Systemmodell und -umgebung
- Zielbestimmungen (Muss-, Wunsch- und Abgrenzungskriterien)
- vollständige funktionale Anforderungen; Qualitätsanforderungen
- GUI-Entwürfe
- ausführliche Testfallszenarien
- Phasenverantwortliche

# Einführung Aussagen- und Prädikatenlogik

## Propositional logic

Assumes that the world contains **facts**

## Propositional logic

Assumes that the world contains **facts**

## First-order logic

Assumes that the world contains

- ▶ **Objects**  
people, houses, numbers, theories, Donald Duck, colors,  
centuries, . . .

## Propositional logic

Assumes that the world contains **facts**

## First-order logic

Assumes that the world contains

- ▶ **Objects**

people, houses, numbers, theories, Donald Duck, colors, centuries, ...

- ▶ **Relations**

red, round, prime, multistoried, ...

brother of, bigger than, part of, has color, occurred after, owns, ...

## Propositional logic

Assumes that the world contains **facts**

## First-order logic

Assumes that the world contains

- ▶ **Objects**

people, houses, numbers, theories, Donald Duck, colors, centuries, ...

- ▶ **Relations**

red, round, prime, multistoried, ...  
brother of, bigger than, part of, has color, occurred after, owns, ...

- ▶ **Functions**

+, middle of, father of, one more than, beginning of, ...

## Symbols

**Constants**     *KingJohn, 2, Karlsruhe, C, ...*

**Predicates**     *Brother, >, =, ...*

**Functions**     *Sqrt, LeftLegOf, ...*

**Variables**     *x, y, a, b, ...*

**Connectives**      $\wedge \vee \neg \Rightarrow \Leftrightarrow$

**Quantifiers**      $\forall \exists$

## Symbols

**Constants**      *KingJohn, 2, Karlsruhe, C, ...*

**Predicates**     *Brother, >, =, ...*

**Functions**      *Sqrt, LeftLegOf, ...*

**Variables**      *x, y, a, b, ...*

**Connectives**    $\wedge \vee \neg \Rightarrow \Leftrightarrow$

**Quantifiers**     $\forall \exists$

## Note

The **equality predicate** is always in the vocabulary  
It is written in infix notation

## Atomic sentence

$predicate ( term_1, \dots, term_n )$

or

$term_1 = term_2$

## Atomic sentence

$predicate ( term_1, \dots, term_n )$

or

$term_1 = term_2$

## Term

$function ( term_1, \dots, term_n )$

or

$constant$

or

$variable$

## Example

*Brother ( KingJohn, RichardTheLionheart )*

## Example

*Brother* ( *KingJohn*, *RichardTheLionheart* )

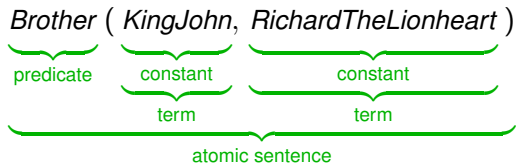
predicate      constant      constant

## Example

*Brother* ( *KingJohn*, *RichardTheLionheart* )

*Brother* is a predicate.  
*KingJohn* and *RichardTheLionheart* are constants.  
Constants are terms.

## Example



Built from atomic sentences using connectives

$$\neg S \quad S_1 \wedge S_2 \quad S_1 \vee S_2 \quad S_1 \Rightarrow S_2 \quad S_1 \Leftrightarrow S_2$$

(as in propositional logic)

Built from atomic sentences using connectives

$$\neg S \quad S_1 \wedge S_2 \quad S_1 \vee S_2 \quad S_1 \Rightarrow S_2 \quad S_1 \Leftrightarrow S_2$$

(as in propositional logic)

Example

$$\textit{Sibling}( \textit{KingJohn}, \textit{Richard} ) \Rightarrow \textit{Sibling}( \textit{Richard}, \textit{KingJohn} )$$

Built from atomic sentences using connectives

$$\neg S \quad S_1 \wedge S_2 \quad S_1 \vee S_2 \quad S_1 \Rightarrow S_2 \quad S_1 \Leftrightarrow S_2$$

(as in propositional logic)

Example

$$\underbrace{Sibling}_{\text{predicate}}(\underbrace{KingJohn}_{\text{term}}, \underbrace{Richard}_{\text{term}}) \Rightarrow \underbrace{Sibling}_{\text{predicate}}(\underbrace{Richard}_{\text{term}}, \underbrace{KingJohn}_{\text{term}})$$

Built from atomic sentences using connectives

$$\neg S \quad S_1 \wedge S_2 \quad S_1 \vee S_2 \quad S_1 \Rightarrow S_2 \quad S_1 \Leftrightarrow S_2$$

(as in propositional logic)

Example

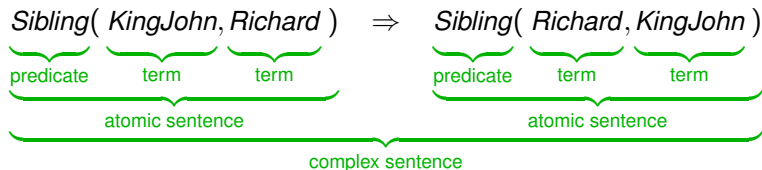
$$\underbrace{\underbrace{\text{Sibling}}_{\text{predicate}} \left( \underbrace{\text{KingJohn}}_{\text{term}}, \underbrace{\text{Richard}}_{\text{term}} \right)}_{\text{atomic sentence}} \Rightarrow \underbrace{\underbrace{\text{Sibling}}_{\text{predicate}} \left( \underbrace{\text{Richard}}_{\text{term}}, \underbrace{\text{KingJohn}}_{\text{term}} \right)}_{\text{atomic sentence}}$$

Built from atomic sentences using connectives

$$\neg S \quad S_1 \wedge S_2 \quad S_1 \vee S_2 \quad S_1 \Rightarrow S_2 \quad S_1 \Leftrightarrow S_2$$

(as in propositional logic)

Example



## Models of first-order logic

Sentences are true or false with respect to models, which consist of

- ▶ a **domain** (also called universe)
- ▶ an **interpretation**

## Models of first-order logic

Sentences are true or false with respect to models, which consist of

- ▶ a **domain** (also called universe)
- ▶ an **interpretation**

## Domain

A non-empty (finite or infinite) set of arbitrary elements

## Models of first-order logic

Sentences are true or false with respect to models, which consist of

- ▶ a **domain** (also called universe)
- ▶ an **interpretation**

## Domain

A non-empty (finite or infinite) set of arbitrary elements

## Interpretation

Assigns to each

- constant symbol: a domain element
- predicate symbol: a relation on the domain (of appropriate arity)
- function symbol: a function on the domain (of appropriate arity)

## Definition

An **atomic sentence**

$$\textit{predicate} ( \textit{term}_1, \dots, \textit{term}_n )$$

is true in a certain model (that consists of a domain and an interpretation)

iff

the domain elements that are the interpretations of  $\textit{term}_1, \dots, \textit{term}_n$  are in the relation that is the interpretation of *predicate*

## Definition

An **atomic sentence**

$$\textit{predicate} ( \textit{term}_1, \dots, \textit{term}_n )$$

is true in a certain model (that consists of a domain and an interpretation)

iff

the domain elements that are the interpretations of  $\textit{term}_1, \dots, \textit{term}_n$  are in the relation that is the interpretation of *predicate*

The truth value of a **complex sentence** in a model is computed from the truth-values of its atomic sub-sentences in the same way as in propositional logic

## Syntax

$\forall$  *variables sentence*

## Syntax

$\forall$  *variables sentence*

## Example

“Everyone studying in Karlsruhe is smart:

$$\forall \underbrace{x}_{\text{variables}} \underbrace{(StudiesAt(x, Karlsruhe) \Rightarrow Smart(x))}_{\text{sentence}}$$

## Semantics

$\forall xP$  is true in a model

iff

for all domain elements  $d$  in the model:

$P$  is true in the model when  $x$  is interpreted by  $d$

## Semantics

$\forall x P$  is true in a model

iff

for all domain elements  $d$  in the model:

$P$  is true in the model when  $x$  is interpreted by  $d$

## Intuition

$\forall x P$  is roughly equivalent to the conjunction of all instances of  $P$

## Semantics

$\forall x P$  is true in a model

iff

for all domain elements  $d$  in the model:

$P$  is true in the model when  $x$  is interpreted by  $d$

## Intuition

$\forall x P$  is roughly equivalent to the conjunction of all instances of  $P$

**Example**  $\forall x \text{StudiesAt}(x, \text{Karlsruhe}) \Rightarrow \text{Smart}(x)$  equivalent to:

$\text{StudiesAt}(\text{KingJohn}, \text{Karlsruhe}) \Rightarrow \text{Smart}(\text{KingJohn})$   
 $\wedge \text{StudiesAt}(\text{Richard}, \text{Karlsruhe}) \Rightarrow \text{Smart}(\text{Richard})$   
 $\wedge \text{StudiesAt}(\text{Karlsruhe}, \text{Karlsruhe}) \Rightarrow \text{Smart}(\text{Karlsruhe})$   
 $\wedge \dots$

## Syntax

$\exists$  *variables sentence*

## Syntax

$\exists$  *variables sentence*

## Example

“Someone studying in Karlsruhe is smart:

$$\exists \underbrace{x}_{\text{variables}} \underbrace{(\text{StudiesAt}(x, \text{Karlsruhe}) \wedge \text{Smart}(x))}_{\text{sentence}}$$

## Semantics

$\exists xP$  is true in a model

iff

there is a domain element  $d$  in the model such that:

$P$  is true in the model when  $x$  is interpreted by  $d$

## Semantics

$\exists x P$  is true in a model

iff

there is a domain element  $d$  in the model such that:

$P$  is true in the model when  $x$  is interpreted by  $d$

## Intuition

$\exists x P$  is roughly equivalent to the disjunction of all instances of  $P$

## Semantics

$\exists x P$  is true in a model

iff

there is a domain element  $d$  in the model such that:

$P$  is true in the model when  $x$  is interpreted by  $d$

## Intuition

$\exists x P$  is roughly equivalent to the disjunction of all instances of  $P$

**Example**  $\exists x \text{StudiesAt}(x, \text{Karlsruhe}) \wedge \text{Smart}(x)$  equivalent to:

- $\text{StudiesAt}(\text{KingJohn}, \text{Karlsruhe}) \wedge \text{Smart}(\text{KingJohn})$
- $\vee \text{StudiesAt}(\text{Richard}, \text{Karlsruhe}) \wedge \text{Smart}(\text{Richard})$
- $\vee \text{StudiesAt}(\text{Karlsruhe}, \text{Karlsruhe}) \wedge \text{Smart}(\text{Karlsruhe})$
- $\vee \dots$

## Semantics

$term_1 = term_2$  is true under a given interpretation

if and only if

$term_1$  and  $term_2$  have the same interpretation

## Semantics

$term_1 = term_2$  is true under a given interpretation

if and only if

$term_1$  and  $term_2$  have the same interpretation

## Examples

$1 = 2$  and  $\forall x \times (Sqrt(x), Sqrt(x)) = x$  are satisfiable

$2 = 2$  is valid

## Important notions

- ▶ validity
- ▶ satisfiability
- ▶ unsatisfiability
- ▶ entailment

are defined for first-order logic in the same way as for propositional logic

## Important notions

- ▶ validity
- ▶ satisfiability
- ▶ unsatisfiability
- ▶ entailment

are defined for first-order logic in the same way as for propositional logic

## Calculi

There are sound and complete calculi for first-order logic (e.g. resolution)

- ▶ Whenever  $KB \vdash \alpha$ , it is also true that  $KB \models \alpha$
- ▶ Whenever  $KB \models \alpha$ , it is also true that  $KB \vdash \alpha$

But these calculi **CANNOT decide** validity, entailment, etc.

# Aufbau des Analysewerkzeugs

- Interpreter
- Run-time-checker
- Verifikationswerkzeug
- GUI

Weiterhin:

- Dokumentation zur Programmier- und Annotationsprache
- Beispielsammlung (verification contest?)

# Zielbestimmung (Interpreter)

Ermöglicht die (schrittweise) Ausführung eines Programms und die Inspektion des aktuellen Programmzustands durch den Benutzer.

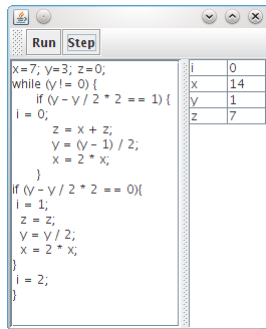
- Wunsch: Benutzer kann den Programmzustand während der Ausführung ändern
- Wunsch: Auswertung von benutzerdefinierten Ausdrücken über Programmzustand

# Zielbestimmung (Run-time-checker)

Erlaubt die Prüfung von Zusicherungen in Programmen *eines konkreten Programmdurchlaufs* zur Laufzeit.

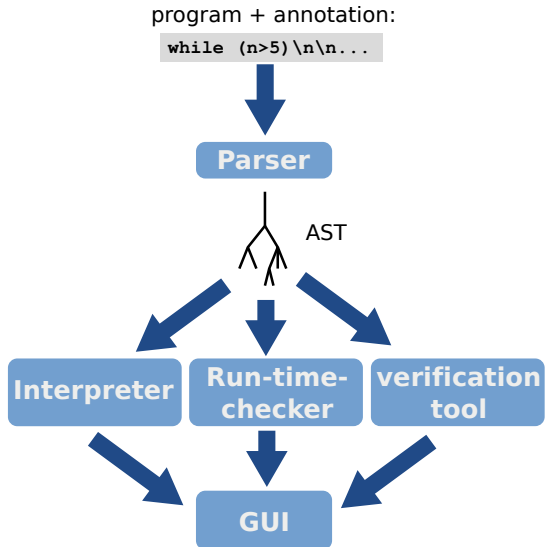
- **Muss:** Auswertung der im Programm eingebetteten (quantorenfreien) Annotationen; Rückmeldung im Fehlerfall über die GUI
- **Soll:** Auswertung von Formeln mit Quantoren über eingeschränkten Bereich
- **Wunsch:** Auswertung von Formeln mit Quantoren mit Hilfe eines Beweisers

Dient zur Steuerung der einzelnen Komponenten des Systems, sowie der Anzeige von Rückmeldungen der (externen) Module des Werkzeugs.



- Muss: Sprache des Benutzerinterfaces: Englisch
- Abgrenzung: nur Optionen, die einen echten Mehrwehrt bieten
- Wunsch: Unicode-Symbole für logische Operatoren
- Wunsch: Verwaltung von alternativen Beweisverpflichtungen

# Workflow des Analysewerkzeugs





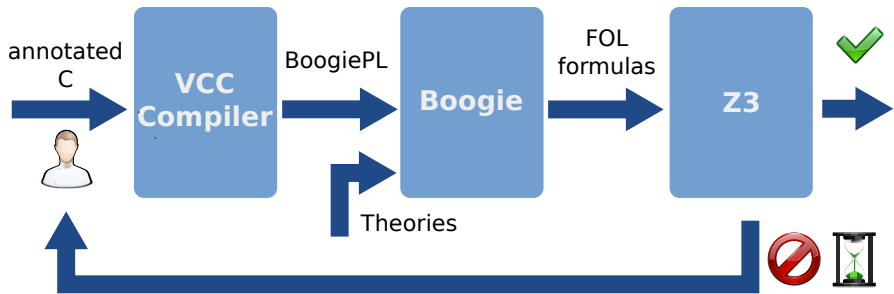
## Autoactive

- Code Contracts
- VCC
- Why/Krakatoa

## Interactive

- KeY
- Isabelle/HOL

# VCC – Ein Verifikationswerkzeug für C



- Parsergeneratoren: antlr, javacc
- Beweiser: Z3

## DEMO – Verifikation von `square.c`

Quelltext: `square.c`

Boogie: `square.bpl`

Hintergrundtheorie: `VccPrelude.bpl`

FOL: `square.simp`

```
(declare-datatypes () ((Nat zero (succ (pred Nat))))))
(declare-fun p (Nat) Bool)
(assert (p zero))
(assert (forall ((x Nat)) (
    implies (p (pred x)) (p x))))
(assert (not (forall ((x Nat)) (p x))))
(check-sat)
```

- automatischer Beweiser für Prädikatenlogik (+ Theorien)
- eingebaute Theorien beinhalten: Arithmetik, Arrays, Bitvektoren
- Anbindung an Z3 durch standardisierte Eingabeformate (SMT-LIB 2.0, Simplify)
- closed source, windows binary (unter Linux: wine)

# Eine einfache While-Sprache

## Logical basis

Typed first-order predicate logic

(Types, variables, terms, formulas, ...)

## Assumption for examples

The signature contains a type *Nat* and appropriate symbols:

- function symbols  $0, s, +, *$  (terms  $s(0), s(s(0)), \dots$  written as  $1, 2, \dots$ )
- predicate symbols  $\dot{=}, <, \leq, >, \geq$

**NOTE:** This is a “convenient assumption” not a definition

## Programs

• Assignments:

$X := t$

$X$ : variable,  $t$ : term

• Test:

if  $B$  then  $\alpha$  else  $\beta$  fi

$B$ : quantifier-free formula,  
 $\alpha, \beta$ : programs

• Loop:

while  $B$  do  $\alpha$  od

$B$ : quantifier-free formula,  
 $\alpha$ : program

• Composition:

$\alpha; \beta$

$\alpha, \beta$  programs

**WHILE is computationally complete**

- Muss: Umfang der while-Sprache wie vorgestellt, zusätzlich: Arrays
- Wunsch: Methodenaufrufe
- Abgr.: nur  $\mathbb{Z}$ , Boolean und Arrays als Datentypen.  
Keine strings, floats, pointer etc.
- Abgr.: kein Heap
- Abgr.: keine Nebenläufigkeit

# Spezifikation von Programmen

## Logic for expressing properties

Full first-order logic (usually with arithmetic)

Partial correctness assertion (Hoare formula)

$$\{P\} \alpha \{Q\}$$

### Meaning:

If  $\alpha$  is started in a state satisfying  $P$  and terminates, then its final state satisfies  $Q$

### Formally:

$\{P\} \alpha \{Q\}$  is valid iff  
for all states  $s, s'$ , if  $s \models P$  and  $s \llbracket \alpha \rrbracket s'$ , then  $s' \models Q$

$\{\text{true}\} X := X + 1 \{X > 1\}$

$\{\text{even}(X)\} X := X + 2 \{\text{even}(X)\}$      **where**  $\text{even}(X) \equiv \exists Z (X \doteq 2 * Z)$

$\{\text{true}\} \alpha_{\text{square}} \{Y = X * X\}$

Prominent information flow property: **non-interference**

Simple case:

- program  $P$
- partion of the program variables of  $P$  in
  - low security variables *low* and
  - high security variables *high*

# Non-Interference

Prominent information flow property: **non-interference**

Simple case:

- program  $P$
- partion of the program variables of  $P$  in
  - low security variables *low* and
  - high security variables *high*

Definition (Non-interference)

For program  $P$  the high variables *high* do not interfere with the low variables *low*

iff

running two instances of  $P$ , with equal values of the low variables, and arbitrary values for the high variables result in the low variables having equal values.

## Example – Illegal Information Flow

```
int low, high;  
Prog  $\equiv$  if (high>0) {low = 1;} else {low = 2;};
```

## Example – Checking Non-interference

```
int low, high;  
Prog  $\equiv$  if (high>0) {low = 1;} else {low = 2;;};
```

---

```
void checkFlow() {  
    int lowIn = rand();  
  
    low = lowIn; high = rand();  
    Prog  
    int res1 = low;  
  
    low = lowIn; high = rand();  
    Prog  
    int res2 = low;  
  
    _(assert res1 == res2)  
}
```

# Zielbestimmungen (Annotationsprache)

Spezifikationen sind im Quelltext des Programms eingebettet, aber durch spezielle Syntax klar vom Programm getrennt:

```
a = 5;  
_(assert true)  
b = 7;
```

- Muss: Syntax und Semantik von Ausdrücken werden aus der Programmiersprache übernommen (soweit möglich)
- Muss: Zusicherungen erlauben Aussagen der Prädikatenlogik
- Muss: Grundlegende Annotationen: `assert`, `assume`, `low/high`
- Soll: syntactic sugar für Vor-/Nachbedingungen, globale Annahmen
- Wunsch: Möglichkeit, verschiedene (getrennte) Vor-/Nachbedingungen anzugeben

- **Muss: Generierung der Beweisverpflichtungen**
  - zur funktionalen Korrektheit (bei festgelegter Anzahl von Schleifendurchläufen)
  - für den Nachweis von Non-interference zwischen high- und low- Variablen im SMT-LIB-Format.
- **Muss: erfolgreiche Verifikation einfacher Programme möglich (russian multiplication, . . .)**

- Web-interface zu Z3, VCC, etc: <http://rise4fun.com>
- Goos, Zimmermann: Vorlesungen über Informatik  
(Band 1: Kapitel 4.2, Prädikatenlogik  
Band 2: Kapitel 8.2, Zusagekalkül)