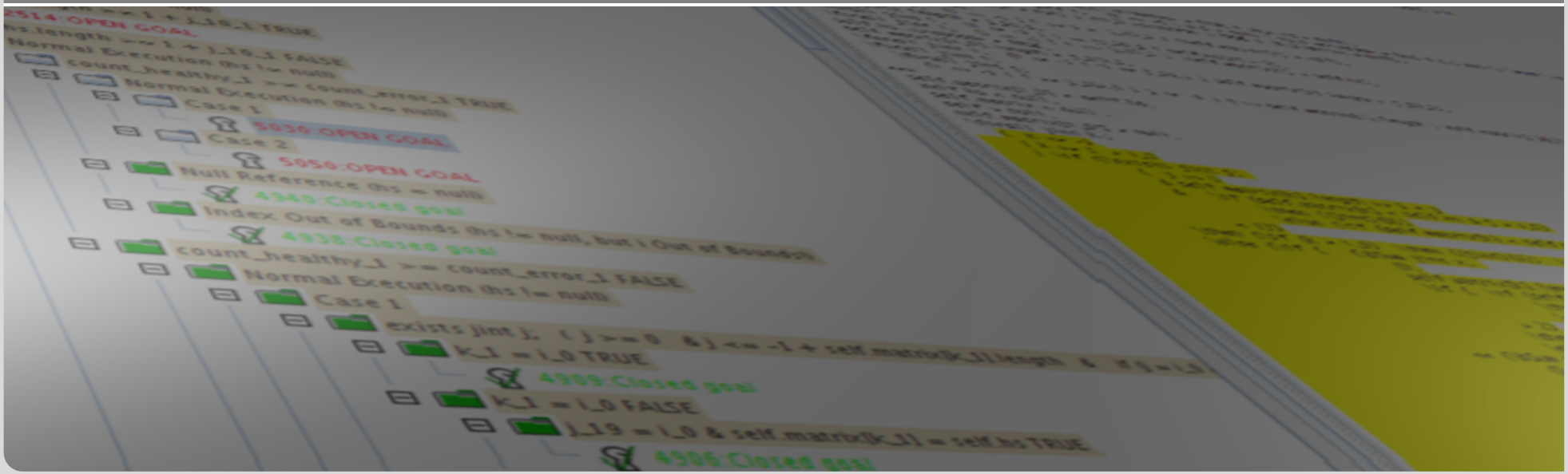
Karlsruher Institut für Technologie

PSE – Überblick

Prof. Bernhard Beckert, Thorsten Bormer, Daniel Bruns | 29. Okt 2012

Institut für Theoretische Informatik – Anwendungsorientierte formale Verifikation



Pflichtenheft

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

First-order Logic

Propositional logic

Assumes that the world contains facts

First-order Logic

Karlsruher Institut für Technologie

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, ...

First-order Logic

Karlsruher Institut für Technologie

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, ...

First-order Logic

Karlsruher Institut für Technologie

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, ...

Syntax of First-order Logic: Basic Elements

Karlsruher Institut für Technologie

Symbols

Constants	$KingJohn, 2, Karlsruhe, C, \ldots$
Predicates	$Brother, >, =, \ldots$
Functions	$Sqrt, LeftLegOf, \ldots$
Variables	$x, y, a, b, \ldots$
Connectives	$\wedge \quad \vee \quad \neg \quad \Rightarrow \quad \Leftrightarrow$
Quantifiers	$\forall \quad \exists$

Syntax of First-order Logic: Basic Elements

Karlsruher Institut für Technologie

Symbols

Constants	$KingJohn, 2, Karlsruhe, C, \ldots$
Predicates	$Brother, >, =, \ldots$
Functions	$Sqrt, LeftLegOf, \ldots$
Variables	$x, y, a, b, \ldots$
Connectives	$\wedge \quad \vee \quad \neg \quad \Rightarrow \quad \Leftrightarrow$
Quantifiers	$\forall \quad \exists$

Note

The equality predicate is always in the vocabulary

It is written in infix notation

Syntax of First-order Logic: Atomic Sentences

Karlsruher Institut für Technologie

Atomic sentence

$$predicate\,(\,term_1,\,\ldots,\,term_n\,)$$

or

$$term_1 = term_2$$

Syntax of First-order Logic: Atomic Sentences

Karlsruher Institut für Technologie

Atomic sentence

$$predicate\,(\,term_1,\,\ldots,\,term_n\,)$$

or

$$term_1 = term_2$$

Term

$$function\,(\,term_1,\,\ldots,\,term_n\,)$$

or

$$constant$$

or

$$variable$$

Syntax of First-order Logic: Atomic Sentences

Karlsruher Institut für Technologie

Example

$$\mathit{Brother}\ (\ \mathit{KingJohn},\ \mathit{RichardTheLionheart}\ )$$

Syntax of First-order Logic: Atomic Sentences

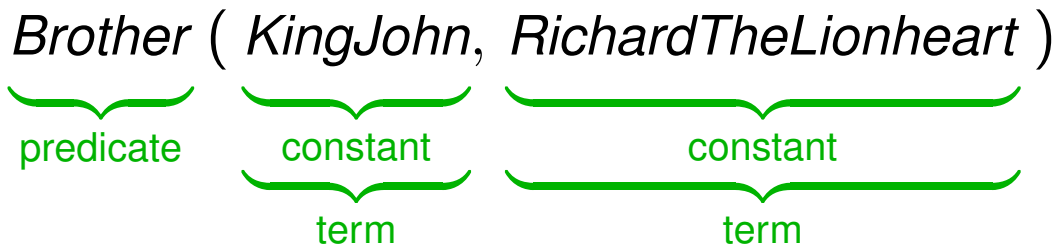
Karlsruher Institut für Technologie

Example

$$\underbrace{\mathit{Brother}}_{\text{predicate}}\,(\,\underbrace{\mathit{KingJohn}}_{\text{constant}},\,\underbrace{\mathit{RichardTheLionheart}}_{\text{constant}}\,)$$

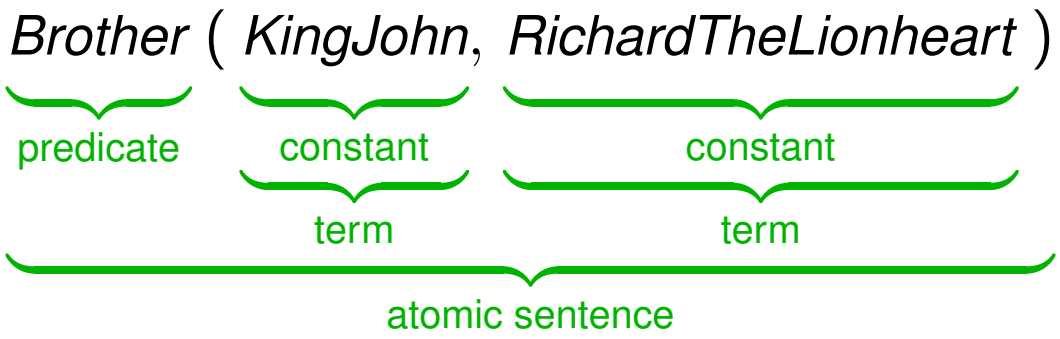
Syntax of First-order Logic: Atomic Sentences

Example



Syntax of First-order Logic: Atomic Sentences

Example



Syntax of First-order Logic: Complex Sentences

Karlsruher Institut für Technologie

Built from atomic sentences using connectives

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

(as in propositional logic)

Syntax of First-order Logic: Complex Sentences

Karlsruher Institut für Technologie

Built from atomic sentences using connectives

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

(as in propositional logic)

Example

$$Sibling(\ KingJohn, Richard\ ) \quad \Rightarrow \quad Sibling(\ Richard, KingJohn\ )$$

Syntax of First-order Logic: Complex Sentences

Built from atomic sentences using connectives

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

(as in propositional logic)

Example

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

Syntax of First-order Logic: Complex Sentences

Karlsruher Institut für Technologie

Built from atomic sentences using connectives

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

(as in propositional logic)

Example

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

Syntax of First-order Logic: Complex Sentences

Karlsruher Institut für Technologie

Built from atomic sentences using connectives

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

(as in propositional logic)

Example

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

Semantics in First-order Logic

Karlsruher Institut für Technologie

Models of first-order logic

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

- a domain (also called universe)
- an interpretation

Semantics in First-order Logic

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

Semantics in First-order Logic

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)

Semantics in First-order Logic

Karlsruher Institut für Technologie

Definition

An atomic sentence

$$predicate\,(\,term_1,\,\ldots,\,term_n\,)$$

is true in a certain model (that consists of a domain and an interpretation)
iff
the domain elements that are the interpretations of $term_1, \ldots, term_n$ are in the relation that is the interpretation of *predicate*

Semantics in First-order Logic

Karlsruher Institut für Technologie

Definition

An atomic sentence

$$predicate\,(\,term_1\,,\,\ldots\,,\,term_n\,)$$

is true in a certain model (that consists of a domain and an interpretation)
iff
the domain elements that are the interpretations of $term_1, \ldots, term_n$ 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

Universal Quantification: Syntax

Karlsruher Institut für Technologie

Syntax

$\forall$ *variables sentence*

Universal Quantification: Syntax

Syntax

$\forall$ *variables sentence*

Example

"Everyone studying in Karlsruhe is smart:

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

Universal Quantification: Semantics

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

Universal Quantification: Semantics

Karlsruher Institut für Technologie

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

Universal Quantification: Semantics

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 $\forall\, x\ StudiesAt(x, Karlsruhe) \Rightarrow Smart(x)$ equivalent to:

$$
\begin{array}{ll}
 & StudiesAt(KingJohn, Karlsruhe) \Rightarrow Smart(KingJohn) \\
\wedge & StudiesAt(Richard, Karlsruhe) \Rightarrow Smart(Richard) \\
\wedge & StudiesAt(Karlsruhe, Karlsruhe) \Rightarrow Smart(Karlsruhe) \\
\wedge & \ldots
\end{array}
$$

Existential Quantification: Syntax

Karlsruher Institut für Technologie

Syntax

$\exists$ *variables sentence*

Existential Quantification: Syntax

Syntax

$\exists$ *variables sentence*

Example

"Someone studying in Karlsruhe is smart:

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

Existential Quantification: Semantics

Karlsruher Institut für Technologie

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

Existential Quantification: Semantics

Karlsruher Institut für Technologie

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*

Intuition

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

Existential Quantification: Semantics

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 $\exists\, x\ StudiesAt(x, Karlsruhe) \wedge Smart(x)$ equivalent to:

$$
\begin{array}{ll}
 & StudiesAt(KingJohn, Karlsruhe) \wedge Smart(KingJohn) \\
\vee & StudiesAt(Richard, Karlsruhe) \wedge Smart(Richard) \\
\vee & StudiesAt(Karlsruhe, Karlsruhe) \wedge Smart(Karlsruhe) \\
\vee & \ldots
\end{array}
$$

Equality

Semantics

$term_1 = term_2$ is true under a given interpretation

if and only if

$term_1$ and $term_2$ have the same interpretation

Equality

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

Properties of First-order Logic

Important notions

- validity
- satisfiability
- unsatisfiablity
- entailment

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

Properties of First-order Logic

Important notions

- validity
- satisfiability
- unsatisfiablity
- 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

Komponenten des Werkzeugs

- Parser (generiert aus Grammatik)
- Interpreter / Runtime-Checker
- Formelgenerator
- Anbindung Verifikationswerkzeug
- GUI

Weiterhin:

- Dokumentation zur Programmier- und Annotationssprache
- Beispielsammlung

Zielbestimmung (Interpreter)

Ermöglicht die (schrittweise) Ausführung eine Programms und die Prüfung von Zusicherungen während der Ausführung (Runtime-Checking).

- Muss: Auswertung der im Programm eingebetteten (quantorenfreien) Annotationen; Rückmeldung im Fehlerfall über die GUI
- Soll: iterative Auswertung von Formeln mit Quantoren über eingeschränkten Bereich
- Wunsch: Auswertung von Formeln mit Quantoren mit Hilfe eines Beweisers (s.u.)
- Wunsch: Inspektion des aktuellen Programmzustands (und Ausdrücken) durch den Benutzer
- Abgrenzung: kein Step-out, kein Hot-Code-Replacement

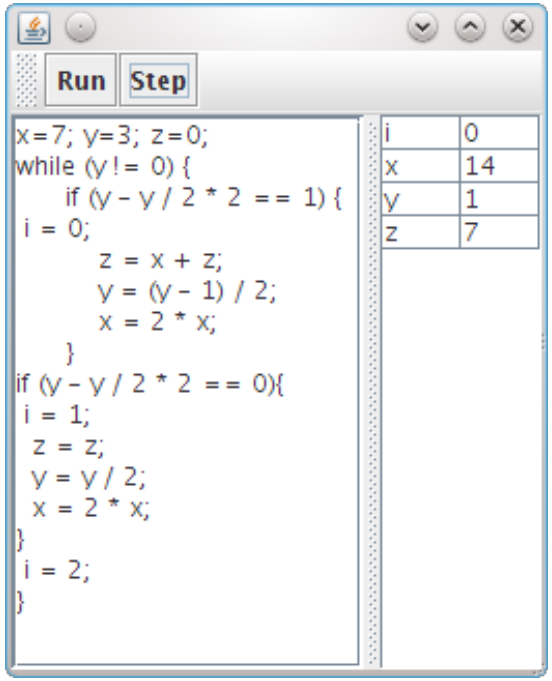
Zielbestimmung (Interpreter)

Ermöglicht die (schrittweise) Ausführung eine Programms und die Prüfung von Zusicherungen während der Ausführung (Runtime-Checking).

- Muss: Auswertung der im Programm eingebetteten (quantorenfreien) Annotationen; Rückmeldung im Fehlerfall über die GUI
- Soll: iterative Auswertung von Formeln mit Quantoren über eingeschränkten Bereich
- Wunsch: Auswertung von Formeln mit Quantoren mit Hilfe eines Beweisers (s.u.)
- Wunsch: Inspektion des aktuellen Programmzustands (und Ausdrücken) durch den Benutzer
- Abgrenzung: kein Step-out, kein Hot-Code-Replacement

Grafische Benutzeroberfläche

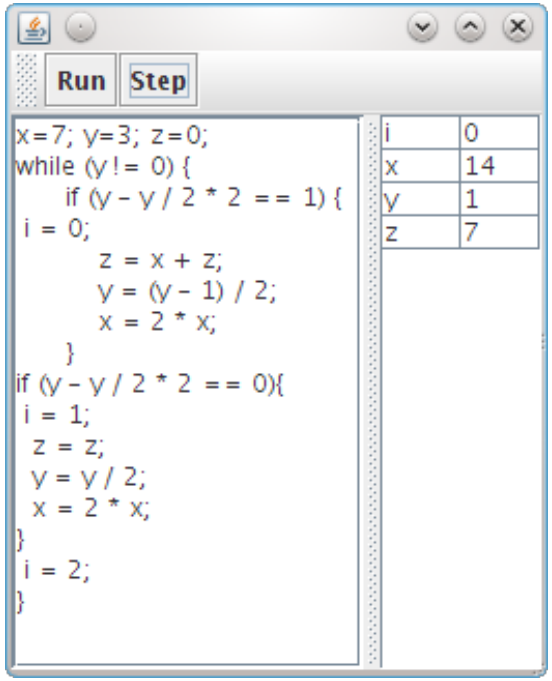
Dient zur Steuerung der einzelnen Komponenten des Systems, sowie der Anzeige von Rückmeldungen der (externen) Module des Werkzeugs.



- Muss: Sprache des Benutzerinterfaces: Englisch
- Abgrenzung: nur Optionen, die einen echten Mehrwehrt bieten
- Wunsch: Syntaxhervorhebung
- Wunsch: Verwaltung von Beweisverpflichtungen

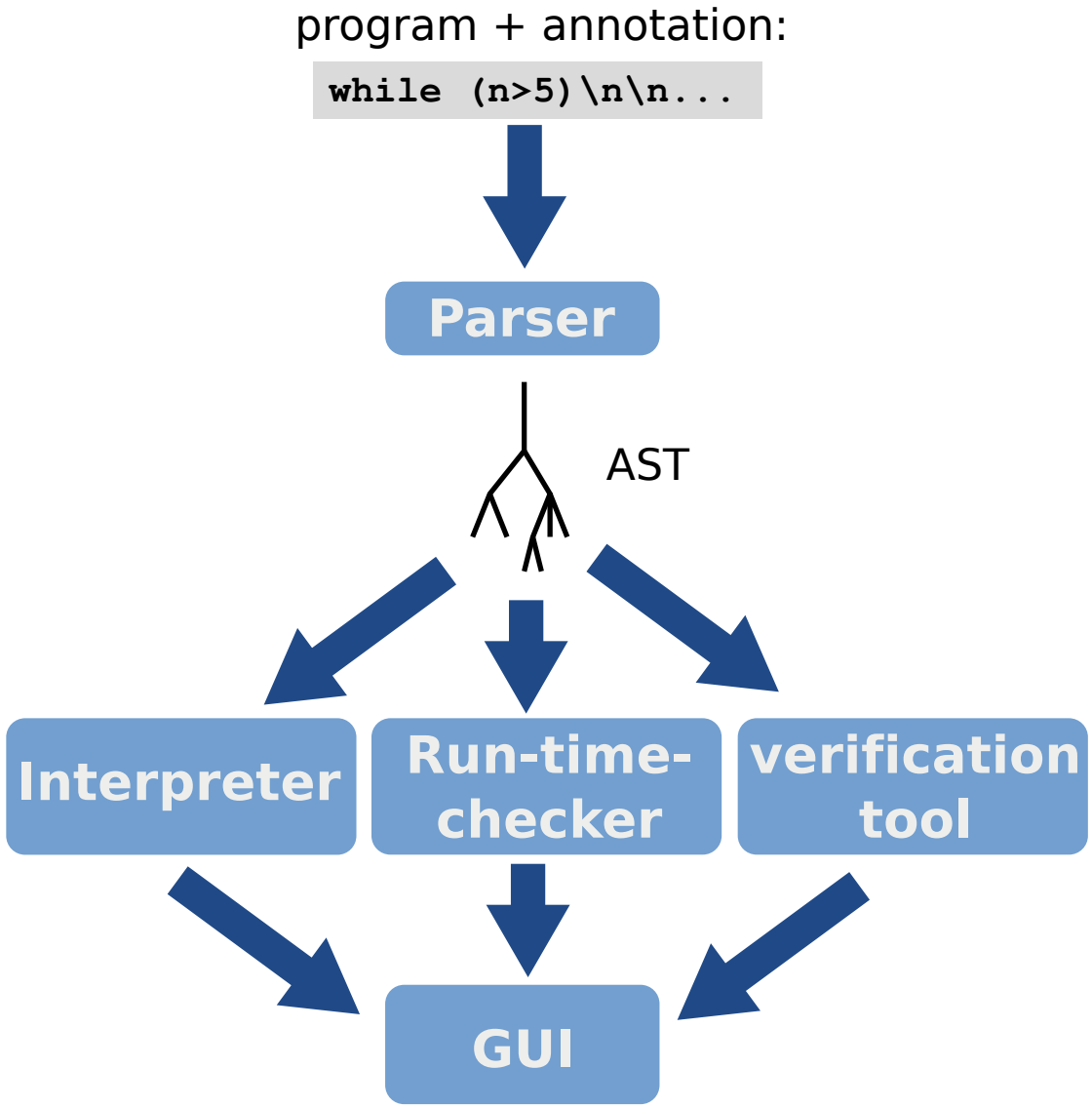
Grafische Benutzeroberfläche

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: Syntaxhervorhebung
- Wunsch: Verwaltung von Beweisverpflichtungen

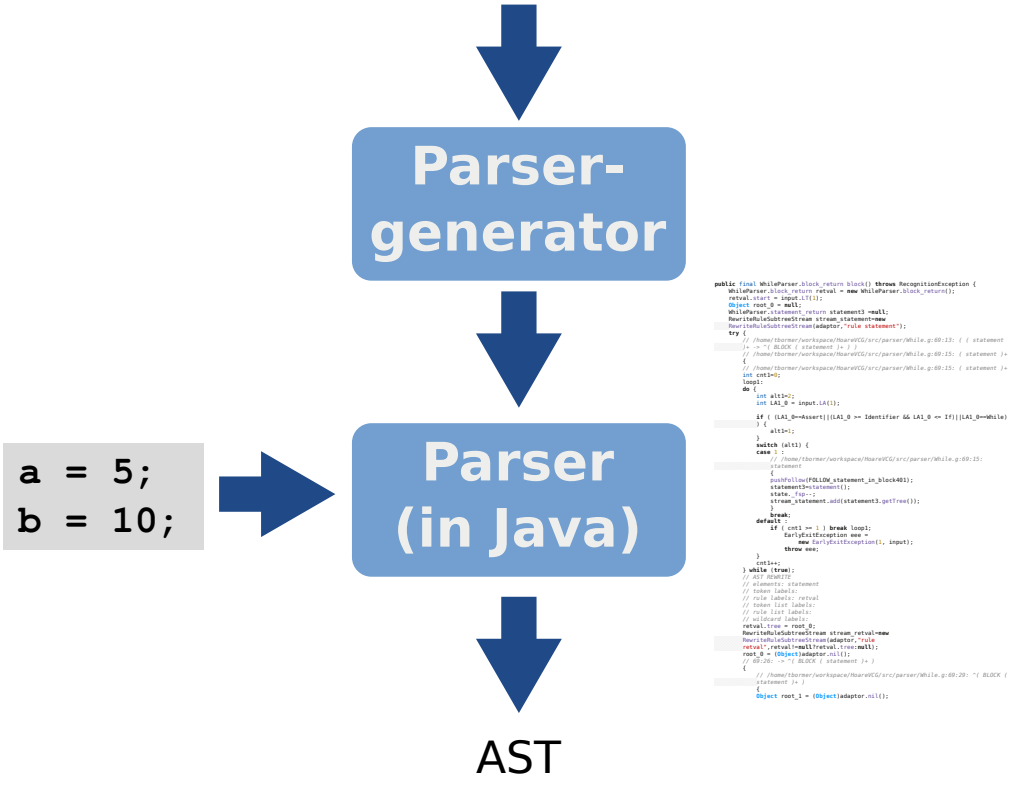
Workflow des Analysewerkzeugs



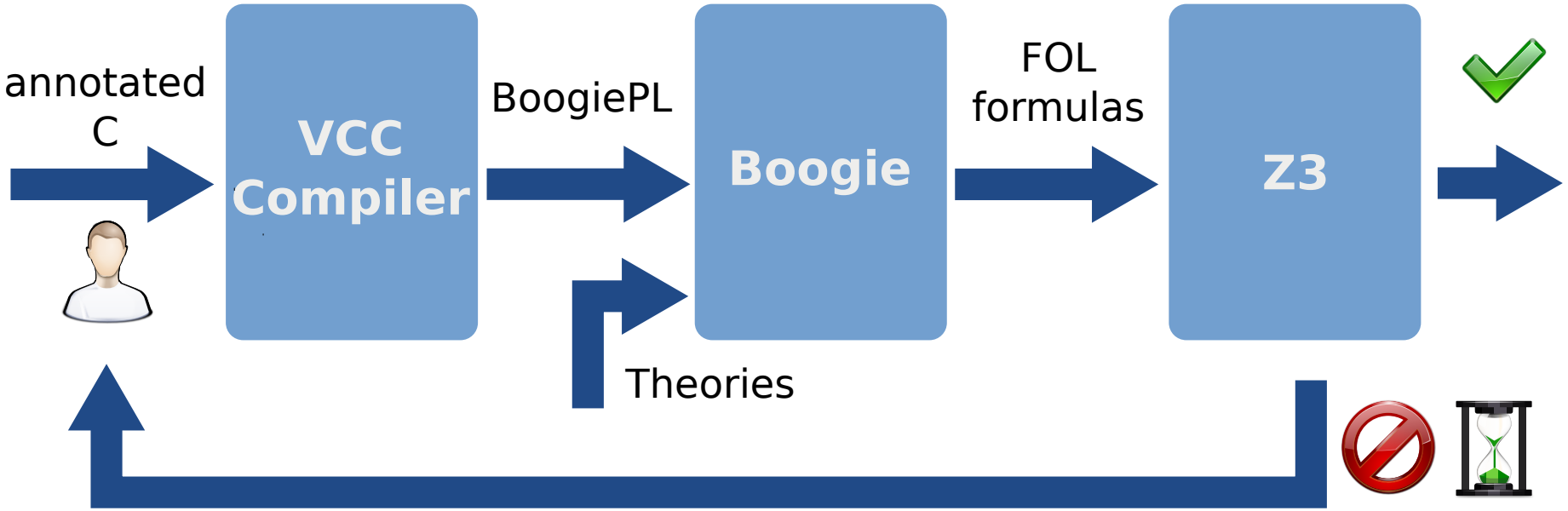
Parsergeneratoren

Grammatik:

```
block: statement+
-> ^(BLOCK statement+);
```



VCC – Ein Verifikationswerkzeug für C



Z3

- automatischer Beweiser für Prädikatenlogik (+ Theorien)
- eingebaute Theorien beinhalten: Arithmetik, Arrays, Bitvektoren
- Anbindung an Z3 durch standardisierte Einfgabeformate (SMT-LIB 2.0, Simplify)
- Quellcode verfügbar (Microsoft Non-Commercial Lizenz); Windows-Binary

Zielbestimmungen (Verifikationskomponente)

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

Eine einfache While-Sprache

WHILE: A Simple Programming Language

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 $s(0), s(s(0)), \ldots$ written as $1, 2, \ldots$)**
- **predicate symbols $\doteq$, $<$, $\leq$, $>$, $\geq$**

NOTE: This is a "convenient assumption" not a definition

WHILE: A Simple Programming Language

Programs

- **Assignments:** $X := t$ — X: variable, t: term
- **Test:** **if** B **then** α **else** β **fi** — B: quantifier-free formula, α, β: programs
- **Loop:** **while** B **do** α **od** — B: quantifier-free formula, α: program
- **Composition:** $\alpha;\ \beta$ — α, β programs

WHILE is computationally complete

Zielbestimmungen (While-Sprache)

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

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*

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 ≡ if (high>0) {low = 1;} else {low = 2;};
```

Example – Checking Non-interference

```
int low, high;
Prog ≡ 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)
}
```

Example – Declassification

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

```
void checkFlow() {
  int lowIn = rand();
  low = lowIn; high = rand();
  _(assume high > 0)
  Prog
  int res1 = low;
  low = lowIn; high = rand();
  _(assume high > 0)
  Prog
  int res2 = low;

  _(assert res1 == res2)
}
```

Zielbestimmungen (Annotationssprache)

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: Deklassifikation von Geheimnissen
- Wunsch: Möglichkeit, verschiedene (getrennte) Vor-/Nachbedingungen anzugeben

Zielbestimmungen (Annotationssprache)

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: Deklassifikation von Geheimnissen
- Wunsch: Möglichkeit, verschiedene (getrennte) Vor-/Nachbedingungen anzugeben

Weiterführende Materialien

- Dokumentation zu ANTLR : `http://antlr.org`
- 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, Zusicherungskalkül)