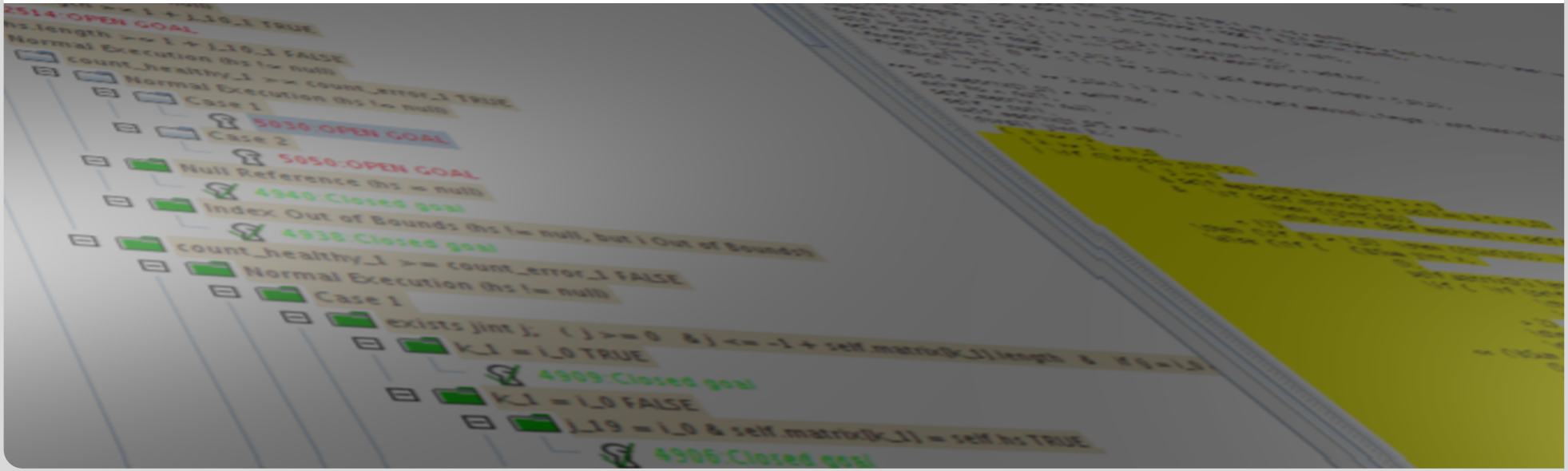
Karlsruher Institut für Technologie

PSE – Einführungsveranstaltung

Prof. Bernhard Beckert, Thorsten Bormer, Daniel Bruns | 27. Oktober 2011

Institut für Theoretische Informatik – Anwendungsorientierte formale Verifikation



Pflichtenheft

Karlsruher Institut für Technologie

Umfang: ca. 20 Seiten

Inhalte:

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

Aufbau des Analysewerkzeugs

Komponenten des Werkzeugs

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

Weiterhin:

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

Zielbestimmung (Interpreter)

Karlsruher Institut für Technologie

Ermöglicht die (schrittweise) Ausführung eine 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)

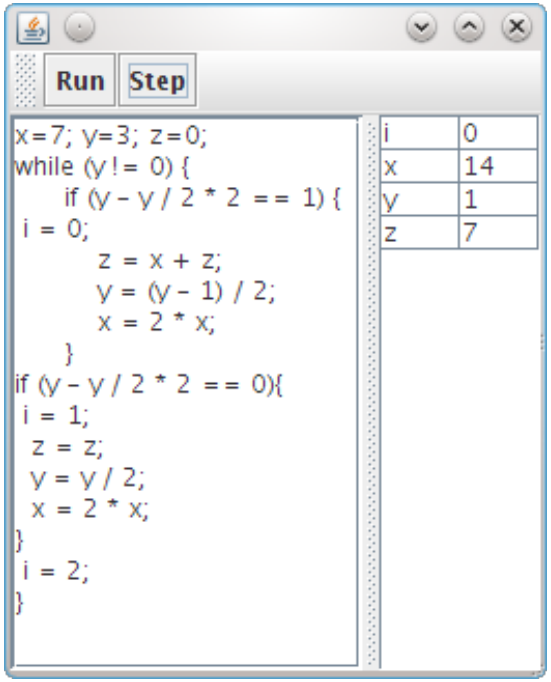
Karlsruher Institut für Technologie

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

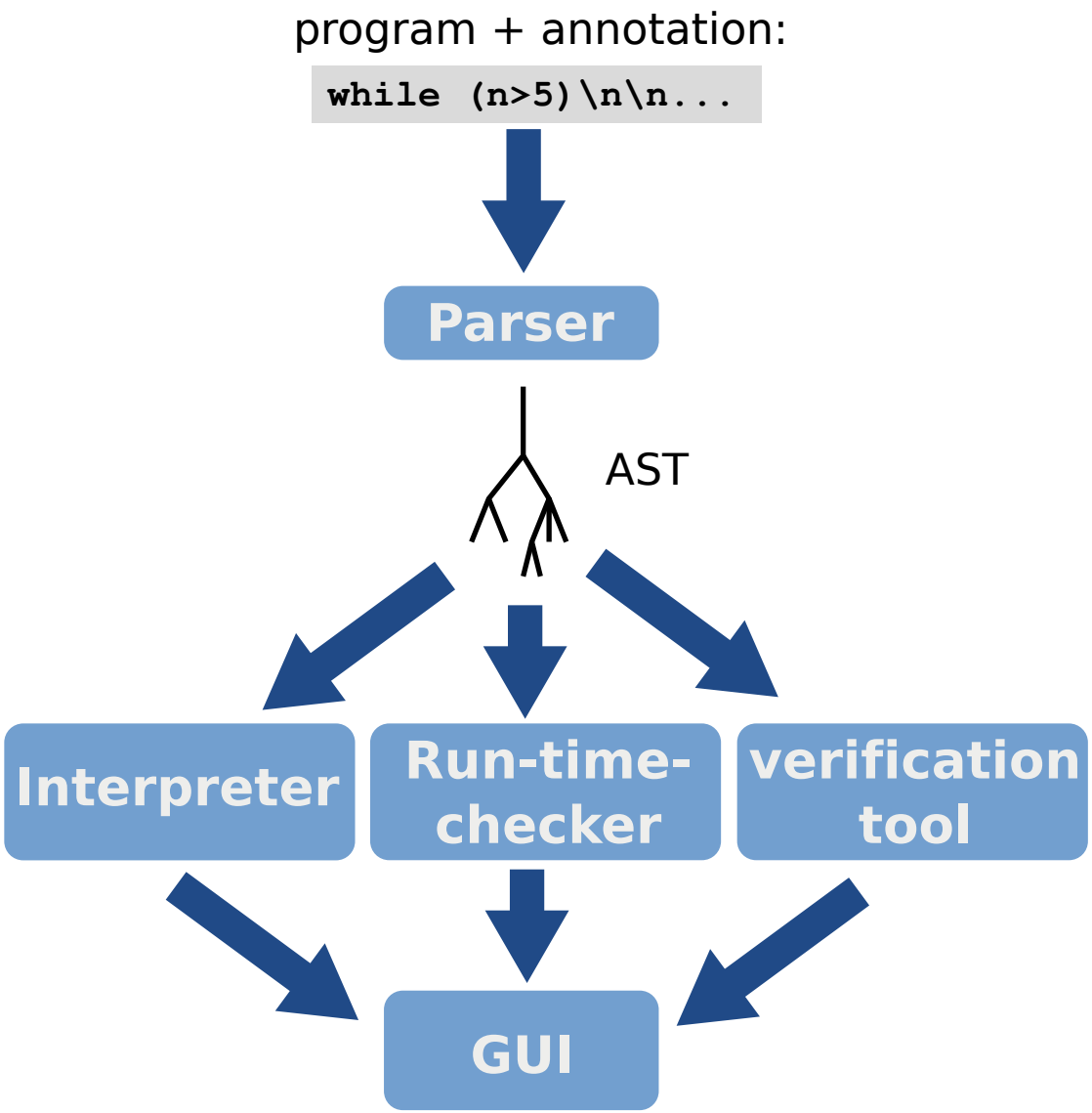
Grafische Benutzeroberfläche

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: Unicode-Symbole für logische Operatoren
- Wunsch: Verwaltung von alternativen Beweisverpflichtungen

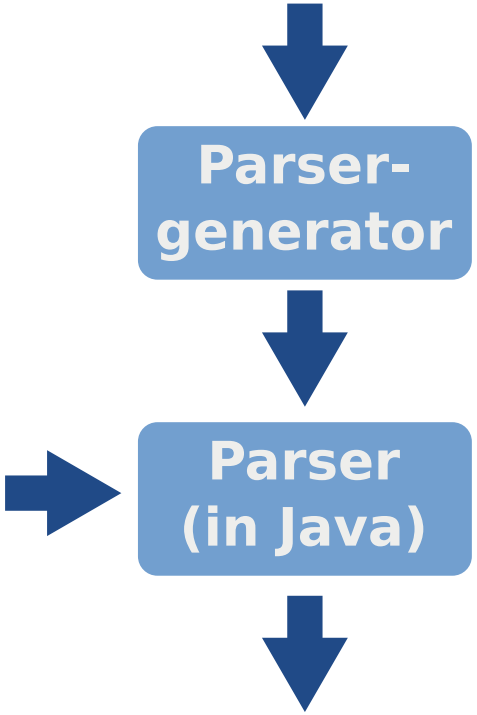
Workflow des Analysewerkzeugs



Parsergeneratoren

Grammatik:

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




```
a = 5;
b = 10;
```

```
public final WhileParser.block_return block() throws RecognitionException {
    WhileParser.block_return retval = new WhileParser.block_return();
    retval.start = input.LT(1);
    Object root_0 = null;
    WhileParser.statement_return statement3 =null;
    RewriteRuleSubtreeStream stream_statement=new
    RewriteRuleSubtreeStream(adaptor,"rule statement");
    try {
        // /home/tbormer/workspace/HoareVCG/src/parser/While.g:69:13: ( ( statement
        )+ -> ^( BLOCK ( statement )+ ) )
        // /home/tbormer/workspace/HoareVCG/src/parser/While.g:69:15: ( statement )+
        {
        // /home/tbormer/workspace/HoareVCG/src/parser/While.g:69:15: ( statement )+
        int cnt1=0;
        loop1:
        do {
            int alt1=2;
            int LA1_0 = input.LA(1);

            if ( (LA1_0==Assert||(LA1_0 >= Identifier && LA1_0 <= If)||LA1_0==While)
            ) {
                alt1=1;
            }
            switch (alt1) {
            case 1 :
                // /home/tbormer/workspace/HoareVCG/src/parser/While.g:69:15:
                statement
                {
                pushFollow(FOLLOW_statement_in_block401);
                statement3=statement();
                state._fsp--;
                stream_statement.add(statement3.getTree());
                }
                break;
            default :
                if ( cnt1 >= 1 ) break loop1;
                    EarlyExitException eee =
                        new EarlyExitException(1, input);
                    throw eee;
            }
            cnt1++;
        } while (true);
        // AST REWRITE
        // elements: statement
        // token labels:
        // rule labels: retval
        // token list labels:
        // rule list labels:
        // wildcard labels:
        retval.tree = root_0;
        RewriteRuleSubtreeStream stream_retval=new
        RewriteRuleSubtreeStream(adaptor,"rule
        retval",retval!=null?retval.tree:null);
        root_0 = (Object)adaptor.nil();
        // 69:26: -> ^( BLOCK ( statement )+ )
        {
            // /home/tbormer/workspace/HoareVCG/src/parser/While.g:69:29: ^( BLOCK (
            statement )+ )
            {
            Object root_1 = (Object)adaptor.nil();
```

Beispiele gängiger Verifikationsumgebungen

Karlsruher Institut für Technologie

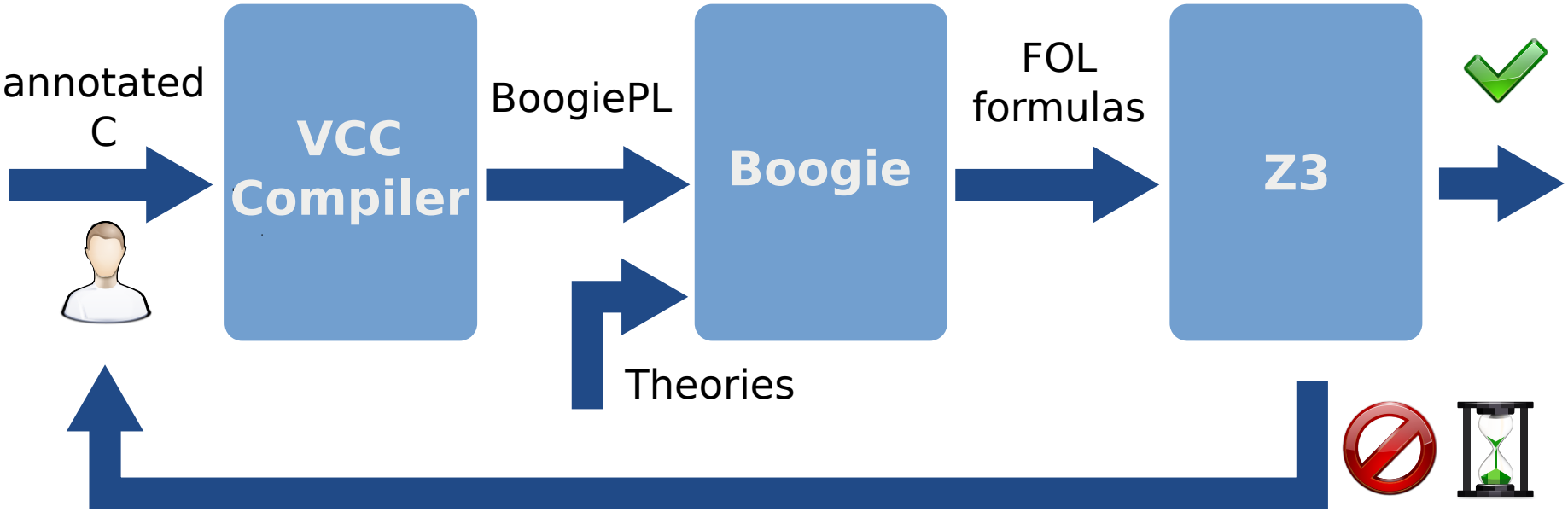
Autoactive

- Code Contracts
- VCC
- Why/Krakatoa

Interactive

- KeY
- Isabelle/HOL

VCC – Ein Verifikationswerkzeug für C



Komponenten der Verifikationstoolchain im PSE

Karlsruher Institut für Technologie

- Parsergeneratoren: antlr, javacc
- Beweiser: Z3

Demo – Verifikation mit VCC

Karlsruher Institut für Technologie

DEMO – Verifikation von `square.c`

Quelltext: square.c
Boogie: square.bpl
Hintergrundtheorie: VccPrelude.bpl
FOL: square.simp

Demo – Z3

```
(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)
```

Z3

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

Einführung Aussagen- und Prädikatenlogik

Propositional Logic: Syntax

Special symbols

$(\quad) \quad \neg \quad \wedge \quad \vee \quad \rightarrow \quad \leftrightarrow$

Signature

Propositional variables $\Sigma = \{p_0, p_1, \ldots\}$

Formulas

- **The propositional variables** $p \in \Sigma$ **are formulas**
- **If** A, B **are formulas, then**

$$\neg A \qquad (A \wedge B) \qquad (A \vee B) \qquad (A \rightarrow B) \qquad (A \leftrightarrow B)$$

are formulas

Propositional Logic: Semantics

Interpretation

Function $I : \Sigma \to \{\underline{\textbf{true}}, \underline{\textbf{false}}\}$

Valuation

Extension of interpretation to formulas as follows:

$$\textbf{val}_I(p) = I(p)$$

$$\textbf{val}_I(\neg p) = \begin{cases} \underline{\textbf{true}} & \textbf{if } I(p) = \underline{\textbf{false}} \\ \underline{\textbf{false}} & \textbf{if } I(p) = \underline{\textbf{true}} \end{cases}$$

Propositional Logic: Semantics

$$\mathbf{val}_I(\alpha) = \begin{cases} \underline{\textbf{true}} & \textbf{if val}_I(\alpha_1) = \underline{\textbf{true}} \\ & \textbf{and val}_I(\alpha_2) = \underline{\textbf{true}} \\ \underline{\textbf{false}} & \textbf{if val}_I(\alpha_1) = \underline{\textbf{false}} \\ & \textbf{or val}_I(\alpha_2) = \underline{\textbf{false}} \end{cases}$$

$$\mathbf{val}_I(\beta) = \begin{cases} \underline{\textbf{true}} & \textbf{if val}_I(\beta_1) = \underline{\textbf{true}} \\ & \textbf{or val}_I(\beta_2) = \underline{\textbf{true}} \\ \underline{\textbf{false}} & \textbf{if val}_I(\beta_1) = \underline{\textbf{false}} \\ & \textbf{and val}_I(\beta_2) = \underline{\textbf{false}} \end{cases}$$

Propositional Logic: Semantics

$$\mathbf{val}_I(A \leftrightarrow B) \quad = \quad \begin{cases} \underline{\textbf{true}} & \textbf{if } \mathbf{val}_I(A) = \mathbf{val}_I(B) \\ \underline{\textbf{false}} & \textbf{if } \mathbf{val}_I(A) \neq \mathbf{val}_I(B) \end{cases}$$

Predicate Logic: Syntax

Additional special symbols

“,” “$\forall$” “$\exists$”

Object variables

Var $= \{x_0, x_1, \ldots\}$

Signature

Triple $\Sigma = \langle F_\Sigma, P_\Sigma, \alpha_\Sigma \rangle$ **consisting of**

- **set** F_Σ **of functions symbols**
- **set** P_Σ **of predicate symbols**
- **function** $\alpha_\Sigma : F_\Sigma \cup P_\Sigma \to \mathbb{N}$
  assigning aritys to function and predicate symbols

Predicate Logic: Syntax

Karlsruher Institut für Technologie

Terms

- **variables** $x \in$ **Var are terms**
- **if** $f \in F_\Sigma$**,** $\alpha_\Sigma(f) = n$**, and** $t_1, \ldots, t_n$ **terms, then** $f(t_1, \ldots, t_n)$ **is a term**

Atoms

If $p \in P_\Sigma$**,** $\alpha_\Sigma(p) = n$**, and** $t_1, \ldots, t_n$ **terms, then** $p(t_1, \ldots, t_n)$ **is an atom**

Predicate Logic: Syntax

Formulas

- **Atoms are formulas**
- **If A, B are formulas, $x \in$ Var, then**

$$\neg A, \quad (A \wedge B), \quad (A \vee B), \quad (A \rightarrow B), \quad (A \leftrightarrow B), \quad \forall x A, \quad \exists x A$$

are formulas

Literals

If A is an atom, then A and $\neg A$ are literals

Example

Signature

$\Sigma_{\leq} = \langle \{0, a, b, f\}, \{in_iv, leq\}, \alpha \rangle$

with

$\alpha(0) = \alpha(a) = \alpha(b) = 0$

$\alpha(f) = 1$

$\alpha(leq) = 2$

$\alpha(in_iv) = 3$ **(in interval)**

Formula

$$\phi = \neg \underbrace{leq(y, x)}_{\textbf{Atom}} \rightarrow \exists z \underbrace{(\neg leq(z, x) \wedge \neg leq(y, z))}_{\textbf{Scope of } \exists z}$$

Predicate Logic: Semantics

Interpretation

A pair $\mathcal{D} = \langle D, I \rangle$ **where**

- D **an arbitrary non-empty set, the *universe***
- I **an interpretation function**

  for $f \in F_\Sigma$**:** $I(f) : D^{\alpha(f)} \to D$

  for $p \in P_\Sigma$**:** $I(p) : D^{\alpha(p)} \to \{\underline{\textbf{true}}, \underline{\textbf{false}}\}$

Variable assignment

A function λ : **Var** $\to D$

Predicate Logic: Semantics

Valuation

Extension of interpretation and variable assignment to formulas

$\mathbf{val}_{\mathcal{D},\lambda}(x) = \lambda(x)$ **for** $x \in$ **Var**

$\mathbf{val}_{\mathcal{D},\lambda}(f(t_1,\ldots,t_n)) = I(f)(\mathbf{val}_{\mathcal{D},\lambda}(t_1),\ldots,\mathbf{val}_{\mathcal{D},\lambda}(t_n))$

$\mathbf{val}_{\mathcal{D},\lambda}(p(t_1,\ldots,t_n)) = I(p)(\mathbf{val}_{\mathcal{D},\lambda}(t_1),\ldots,\mathbf{val}_{\mathcal{D},\lambda}(t_n))$

$$\mathbf{val}_{\mathcal{D},\lambda}(\forall x A) = \begin{cases} \underline{\textbf{true}} & \textbf{if val}_{\mathcal{D},\lambda_x^d}(A) = \underline{\textbf{true}} \quad \textbf{for all } d \in D \\ \underline{\textbf{false}} & \textbf{otherwise} \end{cases}$$

$$\mathbf{val}_{\mathcal{D},\lambda}(\exists x A) = \begin{cases} \underline{\textbf{true}} & \textbf{if val}_{\mathcal{D},\lambda_x^d}(A) = \underline{\textbf{true}} \quad \textbf{for some } d \in D \\ \underline{\textbf{false}} & \textbf{otherwise} \end{cases}$$

$\mathbf{val}_{\mathcal{D},\lambda}$ **defined for propositional operators in the same way as** $\mathbf{val}_I$**.**

Predicate Logic: Semantics

Example

$$
\begin{array}{lll}
D & = & \mathbb{R} \\
I(0) & = & 0 \\
I(a) & = & -1 \\
I(b) & = & 1 \\
I(f) & = & \begin{cases} \mathbb{R} \to \mathbb{R} \\ x \mapsto x^2 \end{cases} \\
I(leq) = \underline{\textbf{true}} & \textbf{iff} & x \leq_{\mathbb{R}} y \\
I(in_iv) = \underline{\textbf{true}} & \textbf{iff} & x \in [a, b]
\end{array}
$$

Predicate Logic: Semantics

Karlsruher Institut für Technologie

Model

An interpretation $\mathcal{D}$ is model of a set Φ of formulas iff

$\text{val}_{\mathcal{D},\lambda}(A) =$ <u>true</u> for all λ and all $A \in \Phi$.

<u>Notation:</u> $\mathcal{D} \models \Phi$

Satisfiable

Φ ist satisfiable iff there are

an interpretation $\mathcal{D}$ and a variable assignment λ s.t.

$\text{val}_{\mathcal{D},\lambda}(A) =$ <u>true</u> for all $A \in \Phi$

Validity

A is valid iff

all interpretations are a model of A

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** $\doteq$, $<$, $\leq$, $>$, $\geq$

<u>NOTE:</u> 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

WHILE: Example – Square of a Number

Compute the square of X (the complicated way)

Making use of: $n^2 = 1 + 3 + 5 + \cdots + (2n - 1)$

$$\left.\begin{array}{l} I := 0; \\ Y := 0; \\ \underline{\textbf{while}}\ I < X\ \underline{\textbf{do}} \\ \quad Y := Y + 2 * I + 1; \\ \quad I := I + 1 \\ \underline{\textbf{od}} \end{array}\right\} \alpha_{\textbf{square}}$$

WHILE: Operational Semantics

Given

A (fixed) first-order structure $\mathcal{A}$ interpreting the function and predicate symbols in the signature

State

$$s \quad = \quad (\mathcal{A}\,, \beta)$$ **where**

β **a variable assignment** **(i.e. function interpreting the variables)**

WHILE: Operational Semantics

State update

$$s[X/e] \quad = \quad (\mathcal{A}\,, \beta[X/e])$$

with

$$\beta[X/e](Y) \quad = \quad \begin{cases} e & \textbf{if } Y = X \\ \beta(Y) & \textbf{otherwise} \end{cases}$$

WHILE: Operational Semantics

Define the relation $s[\![\alpha]\!]s'$ **as follows**

- $s[\![X := t]\!]s'$ **iff** $s' = s[X/s(t)]$
- $s[\![$**<u>if</u>** B **<u>then</u>** α **<u>else</u>** β **<u>fi</u>**$]\!]s'$ **iff**

  $s \models B$ **and** $s[\![\alpha]\!]s'$ **or** $s \models \neg B$ **and** $s[\![\beta]\!]s'$
- $s[\![$**<u>while</u>** B **<u>do</u>** α **<u>od</u>**$]\!]s'$ **iff** **there are states** $s = s_0, \ldots, s_t = s'$ **s.t.**

  $s_i \models B$ **for** $0 \leq i \leq t-1$ **and** $s_t \models \neg B$ **and**

  $s_0[\![\alpha]\!]s_1$, $s_1[\![\alpha]\!]s_2$, **...**, $s_{t-1}[\![\alpha]\!]s_t$
- $s[\![\alpha;\ \beta]\!]s'$ **iff** **there is a state** s'' **such that**

  $s[\![\alpha]\!]s''$ **and** $s''[\![\beta]\!]s'$

$[\![\alpha]\!]$ **is a partial function**

Zielbestimmungen (While-Sprache)

Karlsruher Institut für Technologie

- 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

Expressing Program Properties

Karlsruher Institut für Technologie

Logic for expressing properties

Full first-order logic (usually with arithmetic)

Partial correctness assertion (Hoare formula)

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

Meaning:

If α 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 $s[\![\alpha]\!]s'$, then $s' \models Q$

Expressing Program Properties: Examples

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

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

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

Zielbestimmungen (Annotationssprache)

Karlsruher Institut für Technologie

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`
- Soll: syntactic sugar für Vor-/Nachbedingungen, (In)varianten, globale Annahmen
- Wunsch: Möglichkeit, verschiedene (getrennte) Vor-/Nachbedingungen anzugeben

Weiterführende Materialien

- 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)