

# Pflichtenheft

Simon Bischof

Jan Haag

Adrian Herrmann  
Matthias Schnetz

Lin Jin

Tobias Schlumberger

13. November 2011

# Inhaltsverzeichnis

<b>1</b>	<b>Produktübersicht</b>	<b>3</b>
<b>2</b>	<b>Zielbestimmung</b>	<b>3</b>
2.1	Musskriterien . . . . .	3
2.2	Sollkriterien . . . . .	3
2.3	Kannkriterien . . . . .	3
2.4	Abgrenzungskriterien . . . . .	3
<b>3</b>	<b>Produkteinsatz</b>	<b>3</b>
3.1	Anwendungsbereiche . . . . .	4
3.2	Zielgruppen . . . . .	4
<b>4</b>	<b>Produktumgebung</b>	<b>4</b>
4.1	Software . . . . .	4
4.2	Hardware . . . . .	4
4.3	Schnittstellen . . . . .	4
<b>5</b>	<b>Funktionale Anforderungen</b>	<b>4</b>
5.1	Übersicht . . . . .	4
5.2	Details . . . . .	4
<b>6</b>	<b>Produktdaten</b>	<b>6</b>
<b>7</b>	<b>Produktleistungen</b>	<b>6</b>
<b>8</b>	<b>Weitere nichtfunktionale Anforderungen</b>	<b>6</b>
<b>9</b>	<b>Qualitätsanforderungen</b>	<b>6</b>
<b>10</b>	<b>Globale Testfälle und Testszenarien</b>	<b>6</b>
<b>11</b>	<b>Systemmodelle</b>	<b>8</b>
11.1	Übersicht . . . . .	8
<b>12</b>	<b>Benutzungsoberfläche</b>	<b>8</b>
<b>13</b>	<b>Spezielle Anforderungen an die Entwicklungsumgebung</b>	<b>8</b>
<b>14</b>	<b>Zeit- und Ressourcenplanung</b>	<b>9</b>
14.1	Phasenverantwortliche . . . . .	9
<b>15</b>	<b>Ergänzungen</b>	<b>9</b>
15.1	While-Sprache . . . . .	9
15.1.1	Variablentypen . . . . .	9
15.1.2	Sprachkonstrukte . . . . .	9
15.1.3	Operatoren . . . . .	10
15.1.4	Sonstige Spracheigenschaften . . . . .	10
15.2	Annotationssprache . . . . .	10
<b>16</b>	<b>Glossar</b>	<b>11</b>

# 1 Produktübersicht

Das Produkt ist ein Werkzeug zur Analyse und Korrektheitsprüfung von Programmen. Der Benutzer hat die Möglichkeit, in eine grafische Oberfläche ein Programm mit der im Anhang spezifizierten While-Sprache zu laden und mit Annotationen zu versehen, die als Vor- und Nachbedingungen geprüft werden.

## 2 Zielbestimmung

### 2.1 Musskriterien

- schrittweise Ausführung eines Programms
- Inspektion des aktuellen Programmzustands durch den Benutzer
- Prüfung von Zusicherungen in Programmen zur Laufzeit; Rückmeldung im Fehlerfall über die grafische Benutzeroberfläche
- formale Verifikation eines Programms durch einen automatischen Beweiser
- grafische Benutzeroberfläche in englischer Sprache zur Steuerung der einzelnen Komponenten des Systems
- Programmiersprache mit Zuweisungen, Bedingten Anweisungen und Schleifen
- Ausdrücke in Annotationen ähnlich wie in der Programmiersprache, zusätzlich Quantoren
- Zusicherungen erlauben Aussagen der Prädikatenlogik
- Annotationen assert und assume

### 2.2 Sollkriterien

- Auswertung von Formeln mit Quantoren über eingeschränkten Bereich
- syntactic sugar für Vor- und Nachbedingungen, (In)Varianten und globale Annahmen

### 2.3 Kannkriterien

- Auswertung von benutzerdefinierten Ausdrücken über Programmzustand
- Auswertung von Formeln mit Quantoren mit Hilfe eines Beweisers
- Programmiersprache unterstützt Methodenaufrufe

### 2.4 Abgrenzungskriterien

- nur Optionen in der grafischen Oberfläche, die einen echten Mehrwert bieten
- Programmiersprache unterstützt nur die Typen Ganzzahl, Boolean und Arrays
- Programmiersprache unterstützt keinen Heap
- Programmiersprache unterstützt keine Nebenläufigkeit

# 3 Produkteinsatz

Das Produkt ermöglicht dem Benutzer die Analyse eines Programmes, das in der im Anhang beschriebenen While-Sprache geschrieben ist. Dabei ist es sowohl möglich, die Anweisungen des Programms schrittweise oder komplett zu durchlaufen und dabei zur Laufzeit auf im Programmcode durch Annotationen eingebettete Zusicherungen zu überprüfen, als auch die formale Verifikation des Programms mithilfe des automatischen Theorembeweisers Z3 von Microsoft.

### 3.1 Anwendungsbereiche

- Formale Verifikation von Programmen
- Testen von Programmen der While-Sprache

### 3.2 Zielgruppen

- Informatiker

## 4 Produktumgebung

### 4.1 Software

- Java Runtime Environment SE 6 oder höher
- Windows NT 4.0 oder neuer oder Betriebssystem mit Wine

### 4.2 Hardware

- mindestens 512 MB Arbeitsspeicher

### 4.3 Schnittstellen

- Das Produkt erzeugt Dateien im SMT-LIB 2.0 Format als Schnittstelle zu Z3

## 5 Funktionale Anforderungen

### 5.1 Übersicht

/F10/	Anzeige von Quelltext
/F20/	Syntax Highlighting
/F30/	Veränderung des Quelltextes
/F40/	Speichern des Quelltextes in eine Datei
/F50/	Laden eines Programms aus einer Datei
/F60/	Erkennen und Anzeigen von Syntaxfehlern
/F70/	Erkennen und Anzeigen von Typfehlern
/F80/	Ausführen eines Programms
/F90/	Setzen und Entfernen von Breakpoints
/F100/	Programmabbruch bei Fehler mit Anzeige des Fehlers
/F110/	Anzeige des Programmzustands
/F120/	Auswertung von benutzerdefinierten Ausdrücken
/F130/	Auswertung der Annotationssprache
/F140/	Anzeige der Beweiseinstellungen
/F150/	Laden der Beweiseinstellungen aus einer Datei
/F160/	Speichern der Beweiseinstellungen in eine Datei
/F170/	Beweiser starten und Ergebnis anzeigen

### 5.2 Details

#### /F10/ Anzeige von Quelltext

Der Benutzer sieht den Quelltext des aktuellen Programms in der grafischen Oberfläche.

#### /F20/ Syntax Highlighting

Teile des angezeigten Quelltextes werden abhängig von ihrer Bedeutung in unterschiedlichen Farben angezeigt.

- /F30/** Veränderung des Quelltextes  
Der Benutzer kann den angezeigten Quelltext verändern oder einen neuen Quelltext eingeben.
- /F40/** Speichern des Quelltextes in eine Datei  
Der Benutzer kann den eingegebenen Quelltext ( $\rightarrow$ /F30/) in eine Datei speichern. Dabei gibt der Benutzer den gewünschten Speicherort und den Dateinamen über eine grafische Oberfläche an.
- /F50/** Laden eines Programms aus einer Datei  
Der Benutzer kann ein Quellcode eines Programms aus einer Datei laden. Hierbei gibt der Benutzer die zu ladende Datei über eine grafische Oberfläche an.
- /F60/** Erkennen und Anzeigen von Syntaxfehlern  
Falls der aktuell angezeigte Quelltext Syntaxfehler enthält, wird dies dem Benutzer angezeigt, wenn er die Programmausführung oder die Beweisführung anfordert.
- /F70/** Erkennen und Anzeigen von Typfehlern  
Falls der aktuell angezeigte Quelltext Typfehler enthält, wird dies dem Benutzer angezeigt, wenn er die Programmausführung oder die Beweisführung anfordert.
- /F80/** Ausführen eines Programms  
Die Anweisungen im aktuell angezeigten Quelltext können ausgeführt werden. Dabei sind folgende Ausführungsmodi möglich:
- |                      |   |
|----------------------|---|
| Single Step          | Es wird auf eine Interaktion des Benutzers nur eine Anweisung ausgeführt.   |
| Run until Breakpoint | Es werden sequentiell alle Anweisungen ausgeführt, bis das Programm beendet ist oder ein Breakpoint ( $\rightarrow$ /F90/) erreicht wird. |
- /F90/** Setzen und Entfernen von Breakpoints  
Der Benutzer kann Breakpoints zur Pausierung eines Programmdurchlaufs an Stellen des angezeigten Quellcodes setzen und wieder entfernen.
- /F100/** Programmabbruch bei Fehler mit Anzeige des Fehlers  
Tritt ein Fehler zur Laufzeit des Programms auf, wird die Ausführung abgebrochen und der Fehler dem Benutzer angezeigt.
- /F110/** Anzeige des Programmszustands  
Der Benutzer sieht während der Ausführung eines Programms den aktuellen Inhalt aller globalen und lokalen Variablen. Die Anzeige wird nur aktualisiert, wenn das Programm die Ausführung pausiert.
- /F120/** Auswertung von benutzerdefinierten Ausdrücken  
Der Benutzer kann Ausdrücke über den aktuellen Programmszustand auswerten lassen.
- /F130/** Auswertung der Annotationssprache  
Während der Programmausführung werden die Bedingungen der Annotationssprache ausgewertet. Schlägt eine Überprüfung fehl, so wird die Ausführung des Programms angehalten und dem Benutzer angezeigt, welche Bedingung falsch ist.
- /F140/** Anzeige der Beweisereinstellungen  
Der Benutzer kann die aktuellen Einstellungen ( $\rightarrow$ /P10/) für die Ausführung des Beweisers ( $\rightarrow$ /F170/) anzeigen lassen. Diese werden vor der Anzeige aus einer Datei geladen ( $\rightarrow$ /F150/) und bei einer Veränderung wieder in eine Datei geschrieben ( $\rightarrow$ /F160/).
- /F150/** Laden der Beweisereinstellungen aus einer Datei  
Die Einstellungen für die Ausführung des Beweisers ( $\rightarrow$ /P10/) werden automatisch aus einer Datei gelesen.
- /F160/** Speichern der Beweisereinstellungen in eine Datei  
Die Einstellungen ( $\rightarrow$ /P10/) werden in eine Datei gespeichert.
- /F170/** Beweiser starten und Ergebnis anzeigen  
Der Benutzer kann den Beweiser für das aktuelle Programm starten. Das Ergebnis des Beweisers wird dem Benutzer nach Beendigung des Beweisvorgangs angezeigt.

## 6 Produktdaten

Es sind folgende Daten des Benutzers zu Speichern:

/D10/ Benutzereinstellungen

- Timeout in Sekunden für den Beweiser
- Speicherbegrenzung in Megabyte für den Beweiser

## 7 Produktleistungen

- Verifikation- und Falsifikationsrückmeldungen vom Beweiser müssen korrekt sein, in anderen Fällen soll dem Benutzer mitgeteilt werden, dass der Beweiser das Problem nicht lösen kann

## 8 Weitere nichtfunktionale Anforderungen

Die Lizenz von Z3 erlaubt ausschließlich eine nicht-kommerzielle Benutzung.

## 9 Qualitätsanforderungen

- Programmiersprache muss erweiterbar sein
- mögliche Anbindung anderer Beweiser, die das SMT-LIB 2.0 Format unterstützen
- unter (durch benutzerdefinierte Einstellungen vorgegebene, →/F140/) Normalbedingungen stürzt das Programm nicht ab
- jede Aktion gibt in jedem Fall eine Rückmeldung an den Benutzer

## 10 Globale Testfälle und Testszenarien

Folgende Funktionssequenzen sind zu überprüfen:

/T10/ Eingabe des leeren Programms → keine Syntaxfehler

/T20/ Eingabe des leeren Programms, Beweisvorgang starten → Verifikation durch den Beweiser

/T30/ Eingabe des leeren Programms, Annotation `_(ensure false)`, Programmausführung → Annotationsbedingung schlägt fehl

/T40/ Eingabe des leeren Programms, Annotation `_(ensure true)`, Programmausführung → Annotationsbedingung schlägt nicht fehl

/T50/ Eingabe eines beliebigen Programms, Annotation `_require false`, Programmausführung → Annotationsbedingung schlägt fehl, Programmausführung wird abgebrochen

/T60/ Eingabe eines Programms mit Zuweisung `x = 1`, Annotation `_(require x>0)`, Programmausführung → Annotationsbedingung schlägt nicht fehl

/T70/ Eingabe eines Programms mit Syntaxfehler → Anzeige des Syntaxfehlers

/T80/ Eingabe eines Programms mit while-Schleife oder if-Konstrukt, deren Bedingung kein boolean ist → Anzeige des Typfehlers

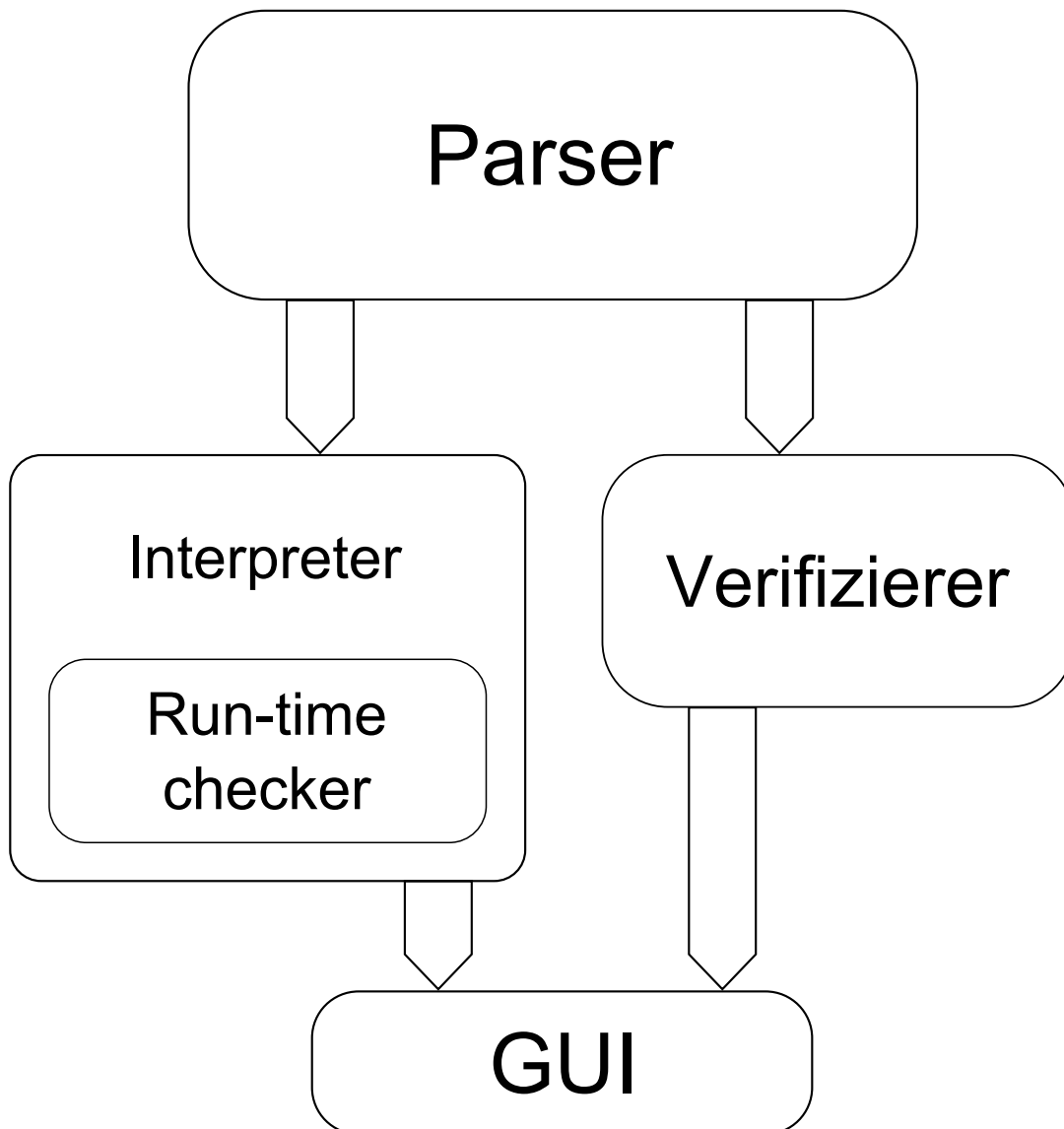
/T90/ Eingabe eines Programms mit einem Operator, der nur auf Ganzzahlen definiert ist und auf einen boolean angewendet wird → Anzeige des Typfehlers

/T100/ Eingabe eines Programms mit einem Operator, der nur auf booleans definiert ist und auf Ganzzahlen angewendet wird → Anzeige des Typfehlers

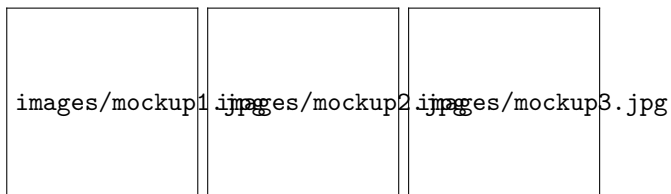
- /T110/ Eingabe eines Programms mit einer Zuweisung eines boolean an eine Ganzzahl → Anzeige des Typfehlers
- /T120/ Eingabe eines Programms mit einer Zuweisung einer Ganzzahl an einen boolean → Anzeige des Typfehlers
- /T130/ Eingabe eines Programms mit einer Zuweisung einer Ganzzahl oder eines boolean an ein Array → Anzeige des Typfehlers
- /T140/ Eingabe eines Programms mit einem Vergleichsoperator, der auf zwei unterschiedliche Typen ausgeführt wird → Anzeige des Typfehlers
- /T150/ Eingabe eines Programms mit Zuweisung  $y = 2x$ , Annotation `_ensure(y==2x)`, Beweisvorgang starten → Verifizierung durch den Beweiser
- /T160/ Eingabe eines Programms mit Zuweisung  $y = 2x$ , Annotation `_ensure(y%2==0)`, Beweisvorgang starten → Verifizierung durch den Beweiser
- /T170/ Eingabe eines längeren Programms ohne Schleifen und ohne if-Konstrukt, Eingabe von Annotationen, die in jedem Fall wahr sind, Beweisvorgang starten → Verifizierung durch den Beweiser
- /T180/ Eingabe eines längeren Programms ohne Schleifen und mit if-Konstrukt, Eingabe von Annotationen, die in jedem Fall wahr sind, Beweisvorgang starten → Verifizierung durch den Beweiser
- /T190/ Eingabe eines längeren Programms mit Schleifen, Eingabe von Annotationen, die in jedem Fall wahr sind, Beweisvorgang starten → Verifizierung durch den Beweiser
- /T200/ Eingabe eines Programms mit Annotation `_(ensure x!=x)`, Beweisvorgang starten → Falsifikation durch den Beweiser
- /T210/ Eingabe eines längeren Programms mit Schleife (z.B. russische Multiplikation, Maximum der Zahlen in einem Array), Eingabe von Annotationen, die in mindestens einem Fall falsch sind, Beweisvorgang starten → Falsifikation durch den Beweiser
- /T220/ Eingabe eines Programms mit Annotation `_(require x%2==0)`, Zuweisung  $y = 2*(x/2)$  und Annotation `_ensure(x==y)`, Beweisvorgang starten → Verifikation durch den Beweiser
- /T230/ Eingabe eines Programms ohne Annotation, Beweisvorgang starten → Beweisvorgang wird nicht gestartet
- /T240/ Eingabe eines Programms mit Division durch 0, Ausführung des Programms → Anzeige des Laufzeitfehlers
- /T250/ Eingabe eines Programms mit Deklaration und Initialisierung eines Arrays, fehlerhafter Arrayzugriff, Ausführung des Programms → Anzeige des Laufzeitfehlers

## 11 Systemmodelle

### 11.1 Übersicht



## 12 Benutzungsoberfläche



## 13 Spezielle Anforderungen an die Entwicklungsumgebung

**Versionsverwaltung** Git

**Kommunikation** E-Mail, Wiki



## 14 Zeit- und Ressourcenplanung

### 14.1 Phasenverantwortliche

Phase	Verantwortliche(r)
Pflichtenheft	Tobias
Entwurf	Adrian
Implementierung	Simon, Jan
Validierung	Lin
Abschlusspräsentation	Matthias

## 15 Ergänzungen

### 15.1 While-Sprache

#### 15.1.1 Variablentypen

**Ganzzahl** ganze Zahl beliebiger Genauigkeit.

**Boolean** Wahrheitswert, entweder wahr oder falsch.

**Array** eine Zusammenfassung mehrerer Variablen eines Variablentyps. Die maximale Anzahl möglicher Variablen in einem Array ist nach erster Festlegung konstant.

#### 15.1.2 Sprachkonstrukte

Konstrukt	Bedeutung
if-else	Ausführung bestimmter Anweisungen, falls eine Bedingung erfüllt ist. Ausführung anderer Anweisungen, falls die Bedingung nicht erfüllt ist.
while	Wiederholte Ausführung von bestimmten Anweisungen, wenn eine Bedingung erfüllt ist.
Methoden	Definition einer Folge von Anweisungen mit Parametern, die zu einem anderen Zeitpunkt im Programmablauf aufgerufen werden können, und einen Wert zurückgeben.
einzeilige Kommentare	Annotation, die keinen Einfluss auf die Quelltextverarbeitung haben.

### 15.1.3 Operatoren

Operator	Eingabetypen	Ergebnistyp	Funktion
==	2 Variablen desselben Typs	Boolean	Prüft T1 und T2 auf Gleichheit, falls sie vom selben Variablentyp sind. Arrays sind gleich, wenn alle Elemente der tiefsten Ebene des Arrays gleich sind.
!=	2 Variablen desselben Typs	Boolean	Prüft T1 und T2 auf Ungleichheit, falls sie vom selben Variablentyp sind. Liefert denselben Wert wie die Negation von ==.
=	1 Variable und ein Ausdruck	?	Weist der Variable den Wert des Ausdrucks zu, wenn dieser Wert vom selben Variablentyp ist, wie die Variable.
<	2 Ganzzahlen, Ganzzahl1 und Ganzzahl2	Boolean	Prüft, ob Ganzzahl1 kleiner als Ganzzahl2 ist.
>	2 Ganzzahlen, Ganzzahl1 und Ganzzahl2	Boolean	Prüft, ob Ganzzahl1 größer als Ganzzahl2 ist.
<=	2 Ganzzahlen, Ganzzahl1 und Ganzzahl2	Boolean	Prüft, ob Ganzzahl1 kleiner oder gleich Ganzzahl2 ist.
>=	2 Ganzzahlen, Ganzzahl1 und Ganzzahl2	Boolean	Prüft, ob Ganzzahl1 größer oder gleich Ganzzahl2 ist.
+	1 Ganzzahl, Ganzzahl	Ganzzahl	Gibt Ganzzahl zurück.
-	1 Ganzzahl, Ganzzahl	Ganzzahl	Gibt -Ganzzahl zurück.
+	2 Ganzzahlen, Ganzzahl1 und Ganzzahl2	Ganzzahl	Addiert Ganzzahl1 und Ganzzahl2.
-	2 Ganzzahlen, Ganzzahl1 und Ganzzahl2	Ganzzahl	Subtrahiert Ganzzahl2 von Ganzzahl1.
*	2 Ganzzahlen, Ganzzahl1 und Ganzzahl2	Ganzzahl	Multipliziert Ganzzahl1 mit Ganzzahl2.
/	2 Ganzzahlen, Ganzzahl1 und Ganzzahl2	Ganzzahl	Dividiert Ganzzahl1 durch Ganzzahl2. Das Ergebnis wird zur Null hin gerundet.
%	2 Ganzzahlen, Ganzzahl1 und Ganzzahl2	Ganzzahl	Liefert den Rest der Ganzzahldivision von Ganzzahl1 durch Ganzzahl2.
!	1 Boolean, Boolean	Boolean	Negiert den Wert von Boolean.
&	2 Boolean, Boolean1 und Boolean2	Boolean	Liefert die Konjunktion von Boolean1 und Boolean2.
	2 Boolean, Boolean1 und Boolean2	Boolean	Liefert die Disjunktion von Boolean1 und Boolean2.

### 15.1.4 Sonstige Spracheigenschaften

- Lexical Scoping
- sequentielle Komposition
- Kurzauswertung von logischen Operatoren

## 15.2 Annotationssprache

Die Annotationssprache besteht aus Ausdrücken der While-Sprache mit Quantoren und zusätzlich den folgenden Konstrukten:

<b>require</b>	Stellt eine Vorbedingung vor Schleifen oder Methoden dar.
<b>ensure</b>	Stellt eine Nachbedingung nach Schleifen oder Methoden dar.
<b>assert</b>	Stellt eine Bedingung an der aktuellen Stelle im Programmablauf dar.

## 16 Glossar

**Anweisung (Programm)** Eine syntaktisch korrekte Vorschrift, die bei der Abarbeitung des Programms auszuführen ist.

**Beweiser** Ein Programm, das versucht die Gültigkeit eines Theorems zu überprüfen. Ein Beispiel für ein solches Programm ist  $\rightarrow Z3$ .

**Kurzauswertung** Die Auswertung eines Ausdrucks wird abgebrochen, sobald der Wert des Ausdrucks bekannt ist.

**Lexical Scoping** Auf eine lokale Variable kann nur in ihrem lexikalischen Sichtbarkeitsbereich Bezug genommen werden.

**sequentielle Komposition** Hintereinanderausführung von mehreren Anweisungen.

**SMT-LIB 2.0** Ein Standardformat zur Beschreibung von Satisfiability Modulo Theories Problemen, d.h. Formeln der Prädikatenlogik erster Stufe.

**Z3** Ein  $\rightarrow$ Beweiser von Microsoft.