

PVS

PVS – “Prototype Verification System”

- System für die interaktive Entwicklung von Spezifikationen und Beweisen
- Entwickelt bei SRI International (Kalifornien) seit 1990
- Wurde und wird weltweit verwendet für viele ernsthafte Formalisierungs- und Beweisaufgaben
- In der Abteilung seit ca. 1994 in großem Umfang eingesetzt, insbesondere in umfangreichen Drittmittel-Projekten; z.Zt.:
 - Verifizierte Compiler (“Verifix”)
 - Modellierung und Analyse kritischer Aspekte und Algorithmen in einer verteilten Architektur mit harten Echtzeit-Bedingungen (“TTA”)

PVS (2)

- PVS ist nicht nur ein Beweiser, sondern hat als wesentliche Komponente eine sehr ausdrucks mächtige *Spezifikationssprache*, in der die zu modellierenden Dinge ausgedrückt werden.
- PVS ist ein *interaktives System*
- Beweiser-Komponente kombiniert verschiedene Vorgehensweisen:
 - Benutzer-gesteuerte Beweisentwicklung (*“proof checking”*)
 - Entscheidungsprozeduren (automatische Komponente)
 - Beweis-Strategien (eine Art *“programmierter Beweissuche”*)
 - Termersetzung (aber keine Vervollständigung)

Mechanisierung des Sequenzen-Kalküls in PVS

Die Beweiser-Komponente von PVS implementiert den Sequenzen-Kalkül. Im Prinzip könnten die Schlußregeln des Sequenzen-Kalküls direkt als Kommandos eines Beweisers implementiert werden. Das würde die Konstruktion von Beweisen aber unnötig mühsam machen.

In PVS: einige Kommandos haben jeweils den Effekt von *Gruppen* von Schlußregeln.

Darstellung von Sequenzen in PVS: Eine Sequenz

$$A_1, A_2, \dots, A_n \vdash B_1, \dots, B_m$$

wird repräsentiert in der Form:

$$\begin{array}{l} \{-1\} \quad A_1 \\ \{-2\} \quad A_2 \\ \dots \\ [-n] \quad A_n \\ | \text{-----} \\ [1] \quad B_1 \\ \dots \\ \{m\} \quad B_m \end{array}$$

Mechanisierung des Sequenzen-Kalküls in PVS [2]

Die Index-Nummern der Teilformeln können zur Steuerung des Beweisers benutzt werden (als Argumente zu Beweiser-Befehlen).

Formeln im Antezedent werden negativ durchnummeriert, Formeln im Sukzedent positiv.

Die Index-Nummern von Formeln, die durch den letzten Beweiserschritt verändert oder neu hinzu gekommen sind, sind in $\{ \}$ eingeschlossen.

Beweis-Konstruktion in PVS

- Ein Beweis in PVS entspricht einem Beweisbaum, der von der “Wurzel”, der zu beweisenden Sequenz, ausgehend konstruiert wird.
- Beweise werden konstruiert durch Generierung von ein oder mehreren Unterzielen, entsprechend der “Rückwärtsanwendung” von Regeln. dadurch wird rückwärts ein Beweisbaum aufgebaut.
- Eine Sequenz (*current sequent*) ist zu jeder Zeit der Fokus des Beweisprozesse; auf sie beziehen sich jeweils die Anweisungen an den Beweiser.
Wenn eine Beweiser-Anweisung neue Teilziele generiert, wandert der Fokus auf das “erste” Teilziel.
- Der Fokus kann durch Anweisungen verändert werden (*postpone*); d.h. die Teilziele können in beliebiger Reihenfolge bearbeitet werden.

Beweis-Konstruktion in PVS (2)

- Wenn ein Beweis-Ast als bewiesen erkannt ist (z.B. weil die Sequenz als eine Axiomeninstanz erkannt wurde), wird dieser Ast abgeschlossen, und der Fokus geht auf das nächste unbewiesene Teilziel über.
- Falls es kein solches Teilziel mehr gibt, wird der Beweis als ganzes abgeschlossen.

PVS Beweiser-Kommandos (1)

Beweiser-Kommandos, die den logischen Regeln (Regeln für aussagenlogische Verknüpfungen) entsprechen:

`flatten` “disjunktive Simplifikation”

Disjunktionen im Sukzedenten, und entsprechend Konjunktionen im Antezedenten, werden aufgelöst

Entspricht (Rückwärts-)Anwendungen der Regeln $\vee R, \Rightarrow R, \wedge L$.

`split` “konjunktiv Aufspaltung”

Für eine konjunktive Teilformel im Sukzedenten bzw. eine disjunktive Teilformel im Antecedenten werden entsprechende Teilziele generiert.

Selektion einer Teilformel notwendig (Default ist “erste, die gefunden wird”).

Entspricht (Rückwärts-)Anwendungen der Regeln $\vee L, \Rightarrow L, \wedge R$.

PVS Beweiser-Kommandos (2)

case “Fallunterscheidung” (*case splitting*)

Für einen oder mehrere Argument-Formeln werden Teilziele generiert, in denen jeweils die Formeln als Voraussetzungen (im Antezedenten) bzw. als Schlußfolgerung (im Sukzedenten) hinzugefügt werden.

Entspricht einer oder mehrerer (Rückwärts-)Anwendungen der Regel *cut*.

Negation als führender Operator wird im wesentlichen durch normierte Darstellung von Sequenzen eliminiert: negierte Teilformeln erscheinen als positive Teilformeln auf der jeweils anderen Seite.

PVS Beweiser-Kommandos (3)

Behandlung von Quantoren

(`skolem`) Universell quantifizierte Variablen (im Sukzedenten; im Akzedenten: existentiell quantifizierte Variablen) werden durch Skolem-Konstanten in neuem Teilziel ersetzt.

Entspricht (Rückwärts-)Anwendung der Regel $\forall R, \exists L$.

(`inst`) Instantiierung universell quantifizierter Variablen (im Antezedenten; existentiell quantifizierter Variablen im Sukzedenten)

Entspricht (Rückwärts-)Anwendung der Regel $\forall L, \exists R$.

(`lemma`) Einführung eines Lemmas als weitere Teilformel in den Antezedenten, möglicherweise mit Instanziierung universell quantifizierter Variablen

PVS Beweiser-Kommandos (4)

Es gibt jeweils Versionen, die explizite Referenzen auf Teilformeln und Angabe von neuen Konstanten-Namen bzw. Termen zulassen, oder geeignete Konstanten generieren (`skolem!`) bzw. versuchen, geeignete Terme zu finden (`inst?`).

Das `lemma`-Kommando kann als eine Art “reduzierte Schnittregel” angesehen werden, die eine Voraussetzung (Beweis des eingeführten Lemmas) nicht explizit einführt.

Theorien in PVS

- Alle Beweise spielen sich in einer logischen Theorie ab, die die Signatur der verwendeten Sprache und nicht-logische Axiome festlegt: “Theorie-Präsentation”.
- Eingaben in PVS sind in Form einer syntaktischen Struktur `theory`, die dem herkömmlichen Begriff von Theorie-Präsentation entspricht.
Eine Theorie faßt zusammengehörige Dinge (Namen, Axiome, Theoreme usw.) zusammen. In dieser Hinsicht entspricht das syntaktische Konstrukt der Theorie dem Modul-Begriff in Programmiersprachen.
- Die PVS zugrundeliegende Logik ist *getypt*, d.h. alle in einer Theorie vorkommenden Namen müssen einen eindeutigen Typ (im Sinn der in der Vorlesung eingeführten mehrsortigen Logik) haben.
(Potentielle Namenskonflikte werden häufig bei der Typ-Überprüfung aufgelöst.)
- Der Typ eines Namens wird (in der Regel) durch explizite Deklaration festgelegt.

Theorien in PVS (2)

- Bezüglich der Deklarationen gilt eine strenge Sequentialität der Sichtbarkeit: Namen können nur referenziert werden an Stellen, die textuell der Deklaration folgen.
- Eine Theorie kann eine *Erweiterung* einer anderen Theorie sein. In PVS kann eine Erweiterung durch *Importierung* der zu erweiternden Theorie ausgedrückt werden.
- In PVS kann eine Theorie von *Parametern* abhängen.
~> wird erst später behandelt.