Prädikatenlogik als Programmiersprache

Hai-Lam Bui

Vortrag im Hauptseminar
Programmiersprachentheorie,
Sommersemester 2007
8.5.2007

Inhalt

- Zwei klassische Forschungsarbeiten von 1974
- Prozedurale Interpretation der Prädikatenlogik als Programmiersprache
- Syntax, Berechnungen, Abläufe, Nichtdeterminismus, Eingabe-Ausgabe-Relation, Laufzeiteffizienz
- Operationale Semantik, modelltheoretische Semantik, Fixpunktsemantik, Gleichwertigkeit dieser Semantiken

Überblick, Eigenschaften der Prädikatenlogik als Programmiersprache

- vollständig nutzerorientiert
- menschliche Logik vs. mathematische Logik von Funktionen

Syntax, Programme als Formelmengen, Hornformeln

- Programm: Menge von speziellen Formeln, den Hornformeln
- Hornformel: Disjunktion von Literalen mit höchstens einem positiven Literal
- Literal: Atomare Formel (sog. positives Literal) oder Negation einer atomaren Formel (sog. negatives Literal)

Syntax, Spezialfälle von Hornformeln

- Allgemeiner Fall einer Hornformel: $B \leftarrow A_1, \ldots, A_n$ mit $n \geq 0$ und B ein Literal oder $\bot$. B wird *Prozedurname* genannt, die Menge $\{A_1, \ldots, A_n\}$ wird *Prozedurrumpf* genannt. Eine Formel dieser Art wird *Prozedurdeklaration* genannt.
- Spezialfall 1: $B \leftarrow$, eine sog. *Zusicherung*.
- Spezialfall 2: $\leftarrow A_1, \ldots, A_n$, eine sog. *Zielanweisung*.
- Spezialfall 3: $\square$, die sog. *Haltanweisung*.

Syntax, Gestalt eines Programms

- Programm: Konsistente Menge von Hornformeln
- Für eine Berechnung wird eine Zielanweisung hinzugefügt
- Mit Hilfe eines geeigneten Inferenzsystems wird ein Gegenbeispiel gefunden, welches zugleich das Ergebnis der Berechnung darstellt (durch Bindung von Variablen an Terme)

Fakultätsfunktion

```
(F1) Fact(0,s(0)) <-
(F2) Fact(s(x),u) <- Fact(x,v),Times(s(x),v,u)
```

Um die Fakultät von 2 zu berechnen, wird eine dritte Zeile hinzugefügt:

```
(F3) <- Fact(s(s(0)),x)
```

Inferenzsystem und Berechnungen

- Inferenzsystem: keine Axiome und nur eine Regel, die *Resolutionsregel*:
- Falls $\leftarrow A_1, \ldots, A_{i-1}, A_i, A_{i+1}, \ldots, A_n$ und $B \leftarrow B_1, \ldots, B_m$ gegeben sind, dann lässt sich $\leftarrow (A_1, \ldots, A_{i-1}, B_1, \ldots, B_m, A_{i+1}, \ldots, A_n)\theta$ herleiten (falls geeignetes θ existiert)
- $m = 0$ ist möglich

Inferenzsystem, Beispiel

Als Beispiel seien wieder die Zeilen `(F2)` und `(F3)` gegeben:

```
(F3) <- Fact(s(s(0)),x)
(F2) Fact(s(x),u) <- Fact(x,v),Times(s(x),v,u)
```

Hier lässt sich die folgende Zielanweisung ableiten:

```
<- Fact(s(0),v), Times(s(s(0)),v,x)
```

Eine *Berechnung* in dieser Sprache ist eine Folge von Zielanweisungen. Eine Berechnung ist erfolgreich, falls die letzte Zielanweisung $\square$ ist. Eine Berechnung terminiert erfolglos, falls es in der letzten Zielanweisung keinen Prozeduraufruf gibt, der mit einem Prozedurnamen angeglichen werden kann.

Beispiel einer Berechnung

```
(F1) Fact(0,s(0)) <-
(F2) Fact(s(x),u) <- Fact(x,v),Times(s(x),v,u)
(*) Times(s(0),s(0),s(0)) <-
(**) Times(s(s(0)),s(0),s(s(0))) <-
```

Im Folgenden wird nun jeweils die erste Prozedur aufgerufen, d.h. mit dem Namen einer anderen Prozedur angeglichen und durch den Rumpf dieser Prozedur ersetzt.

Aktuelles Ziel	*Aufruf*	*Substitutionen*
`Fact(s(s(0)),x)`		
`Fact(s(0),v)` `Times(s(s(0)),v,x)`	`(F2)`	

`Fact(0,v')`
`Times(s(0),v',v)`
`Times(s(s(0)),v,x)`
	`(F2)`	

`Times(s(0),s(0),v)` `Times(s(s(0)),v,x)`	`(F1)`	`v' := s(0)`
`Times(s(s(0)),s(0),x)`	`(*)`	`v := s(0)`
□	`(**)`	`x := s(s(0))`

Nichtdeterminismus durch Musterangleich

```
(M1) Member(x,cons(x,y)) <-
(M2) Member(z,cons(x,y)) <- Member(z,y)
```

Hier gibt es drei gültige Berechnungen für die Zielanweisung

```
(M3) <- Member(x,cons(a,cons(b,nil)))
```

wobei eine nicht terminiert und alle drei Ergebnisse verschieden sind:

Aktuelles Ziel	*Aufruf*	*Substitutionen*
□	`(M1)`	`x := a`

Aktuelles Ziel	*Aufruf*	*Substitutionen*
`Member(x,cons(b,nil))`	`(M2)`	
□	`(M1)`	`x := b`

Aktuelles Ziel	*Aufruf*	*Substitutionen*
`Member(x,cons(b,nil))`	`(M2)`	
`Member(x,nil)`	`(M2)`	

Nichtdeterminismus

- berechnet werden Relationen
- Funktionen als Spezialfälle von Relationen
- Bemerkung: Selektorfunktionen wie aus SML bekannte `hd` und `tl` sind nicht notwendig

Eingabe und Ausgabe

- Substitution des Musterangleichs θ besteht aus zwei Teilen
- Eingabetransfer: Instanziierung von Variablen in B
- Ausgabetransfer: Instanziierung von Variablen in $A_1, \ldots, A_{i-1}, A_{i+1}, A_n$
- Partieller Transfer: Variable wird instanziiert, jedoch nicht mit einem Grundterm

Austauschbarkeit von Eingabe und Ausgabe, Beispiel

- `(F3) <- Fact(s(s(0)),x)` berechnet deterministisch die Fakultät von 2
- `(F4) <- Fact(x,s(0))` berechnet nichtdeterministisch die Zahlen, deren Fakultät 1 ist
- In der Prädikatenlogik sind Eingabe- und Ausgabepositionen nicht festgelegt!

Reihenfolge der Prozeduraufrufe

- Zielanweisung: Menge von Prozeduraufrufen
- Die Aufrufreihenfolge ist weder festgelegt, noch festlegbar, und verändert das Ergebnis nicht
- Trotzdem nicht unwichtig, da verschiedene Aufrufreihenfolgen verschieden effizient sind

Reihenfolge der Prozeduraufrufe, Beispiel

- Programm: `Sort(x,y) <- Perm(x,y),Ord(y)`

- Möglichkeit 1: `Perm(x,y)` vor `Ord(y)`
- Möglichkeit 2: `Ord(y)` vor `Perm(x,y)`
- Deutliche Effizienzunterschiede sichtbar!

Reihenfolge der Prozeduraufrufe, Verbesserung durch Parallelisierung

- Aufrufe ohne gemeinsame Variablen können parallel verarbeitet werden
- Bei Aufrufen mit gemeinsamen Variablen kann die Kontrolle an geeigneten Stellen an einen anderen Aufruf übergeben werden

Reihenfolge der Prozeduraufrufe, Beispiel 2

- Für das Sortieren von x, erzeuge, mit der leeren Teilliste nil beginnend, eine Teilliste einer Permutation von x, dann teste ob die Teilliste geordnet ist.
- Ist sie nicht geordnet, erzeuge eine neue Teilliste, sofern es noch eine nicht erzeugte Teilliste gibt.
- Ist sie geordnet, aber noch keine komplette Permutation, dann füge der Teilliste ein weiteres Element hinzu und teste ob die neue Teilliste geordnet ist.
- Ist sie geordnet und eine komplette Permutation von x, dann ist es die gewünschte sortierte Version von x.

- Hinweis: Die Äste dieses Entscheidungsbaums brechen früher ab als bei den anderen Verfahren, daher effizienter!

Reihenfolge der Prozeduraufrufe, Probleme

- Die Effizienz einer Aufrufreihenfolge hängt von den Eingabe- und Ausgabepositionen ab; eine allgemeingültige Lösung gibt es nicht
- Betrachten wir als Beispiel die beiden Programme mit t als variablenfreiem Term und x, y, z als Variablen:
  `(R1) R(t,z) <- P(t,y),Q(y,z)`
  `(R2) R(x,t) <- P(x,y),Q(y,t)`
  Während es in `(R1)` effizienter wäre, `P` vor `Q` auszuführen, so gilt in `(R2)` das umgekehrte.
- Für die Praxistauglichkeit bräuchte es eine Hilfssprache, oder gar einen intelligenten Interpreter

Semantik, Arten der Semantik

- Operationale Semantik (Top-Down)
- Modelltheoretische Semantik
- Fixpunktsemantik (Bottom-Up)
- Die Definitionen werden sich als gleichwertig herausstellen

Operationale Semantik

- Definition eines implementierungsunabhängigen Interpreters; entspricht dem Beweisverfahren in der Prädikatenlogik
- Notation: $\mathcal{F} \vdash_I X$ gdw. X ist bezüglich des Inferenzsystems I aus $\mathcal{F}$ herleitbar (I wird bei Eindeutigkeit weggelassen)
- Semantik eines Prädikatenlogik-Programms entspricht der Semantik der vom Programm charakterisierten Prädikatensymbole

Operationale Semantik

- Zu jedem Prädikatensymbol gehören Eingabe-Ausgabe-Tupel; dies induziert eine Eingabe-Ausgabe-Relation
- Operationale Semantik eines Prädikatensymbols P in einer Klauselmenge $\mathbf{A}$:
  $\mathbf{D}_1(P) = \{(t_1, \ldots, t_n) | \mathbf{A} \vdash P(t_1, \ldots, t_n)\}$

Operationale Semantik

Folgerung: Ist $\mathbf{A}$ ein widerspruchsfreies Programm, und ist $(t_1, \ldots, t_n) \in \mathbf{D}_1$, so gibt es zu jedem „Bruchteil“ dieses Tupels als Eingabe eine Berechnung, die das Tupel berechnet.

Unser Inferenzsystem lässt sich als eine Top-Down-Beweisprozedur betrachten, da sie von der Behauptung ausgehend den Widerspruch herleitet.

Modelltheoretische Semantik

- $\mathbf{D}_2(P) = \{(t_1, \ldots, t_n) | \mathbf{A} \models P(t_1, \ldots, t_n)\}$
- Nach dem Gödelschen Vollständigkeitssatz gilt $\mathcal{F} \vdash F$ gdw. $\mathcal{F} \models F$ für alle $\mathcal{F}, F$, woraus sofort $\mathbf{D}_1 = \mathbf{D}_2$ folgt.

Benötigte Begriffe aus der Logik

Signatur einer Prädikatenlogik besteht aus einer Menge von Symbolen mit Stelligkeiten; Beispiele sind das 0-stellige Funktionssymbol 0, das 1-stellige Funktionssymbol s und das 2-stellige Prädikatensymbol $Fact$.

Grundformeln sind variablenfreie Formeln.

Benötigte Begriffe aus der Logik

Atomare Grundformeln sind vom Typ $P(t_1, \ldots, t_n)$, wobei P ein Prädikatensymbol der Signatur und $t_1, \ldots, t_n$ variablenfreie Terme sind.

Herbrand-Basis bezeichnet die Menge aller atomaren Grundformeln.

Herbrand-Interpretation bezeichnet eine beliebige Teilmenge der Herbrand-Basis.

Benötigte Begriffe aus der Logik

Eine atomare Grundformel A ist wahr in einer Herbrand-Interpretation I genau dann wenn $A \in I$. Der Wert zusammengesetzter Grundformeln ergibt sich aus den Atomen.

Eine Formelmenge $\mathcal{F}$ ist wahr in einer Herbrand-Interpretation I, wenn jede Formel dieser Menge wahr ist. In diesem Fall ist I ein *Herbrand-Modell* von $\mathcal{F}$. Nach dem Satz von Löwenheim-Skolem gilt: Eine Formelmenge hat ein Modell genau dann wenn sie ein Herbrand-Modell hat.

Modelltheoretische Semantik, ausgedrückt als Herbrand-Modell

Sei $\mathbf{A}$ ein Programm, dann bezeichnet $\mathbf{M}(\mathbf{A})$ die Menge aller Herbrand-Modelle von $\mathbf{A}$. $\cap\mathbf{M}(\mathbf{A})$ bezeichne dann den Schnitt aller Herbrand-Modelle, enthält also alle atomaren Grundformeln, die in jedem Herbrand-Modell wahr sind. Dann ist die modelltheoretische Semantik definiert als:

$$\mathbf{D}_2(P) = \{(t_1, \ldots, t_n) | P(t_1, \ldots, t_n) \in \cap\mathbf{M}(\mathbf{A})\}$$

Fixpunktsemantik

- Beschreibt Bottom-Up-Berechnungen
- Aus den Zusicherungen werden im jedem Schritt Folgerungen erzeugt, bis die Menge vollständig ist

Fixpunktsemantik

Sei P_j ein Prädikatensymbol der Signatur des Programms. Die Transformation T_j, welche Herbrand-Interpretationen I auf Herbrand-Interpretationen abbildet, ist definiert wie folgt:

> $T_j(I)$ enthält eine atomare Grundformel A der Herbrand-Basis genau dann wenn A mit dem Prädikatensymbol P_j beginnt und es eine Grundinstanz $C\sigma$ einer Klausel C aus $\mathbf{A}$ gibt mit $C\sigma = A \leftarrow \wedge^*(A_1, \ldots, A_m)$ und $A_1, \ldots, A_m \in I, m \geq 0$.

Fixpunktsemantik, Transformation

Die zu einem Programm zugehörige Transformation einer Herbrand-Interpretation I ist $T = T_1(I) \cup \cdots \cup T_n(I)$, wobei n der Zahl der Prädikatensymbole in der Signatur entspricht. T wird im Papier als monoton angenommen.

Es kann passieren, dass I eine echte Teilmenge von $T(I)$ ist. Definieren wir nun $\mathbf{C}(\mathbf{A})$ als die Menge aller Herbrand-Interpretationen I, die abgeschlossen unter der von $\mathbf{A}$ induzierten Transformation T sind (d.h., $T(I) \subseteq I$). Dann ist die Fixpunktsemantik definiert als:

$$\mathbf{D_3}(P) = \{(t_1, \ldots, t_n) | P(t_1, \ldots, t_n) \in \cap \mathbf{C}(\mathbf{A})\}$$

Gleichheit von modelltheoretischer Semantik und Fixpunktsemantik

Es ist einfach genug, $\mathbf{M}(\mathbf{A}) = \mathbf{C}(\mathbf{A})$ zu zeigen, obwohl $\cap\mathbf{M}(\mathbf{A}) = \cap\mathbf{C}(\mathbf{A})$ genügen würde.

> Ist $\mathbf{A}$ eine Menge von Prozedurdeklarationen, dann ist $\mathbf{M}(\mathbf{A}) = \mathbf{C}(\mathbf{A})$, das heißt $\models_I \mathbf{A}$ gdw. $T(I) \subseteq I$ für alle Herbrand-Interpretationen I von $\mathbf{A}$.

Gleichheit von modelltheoretischer Semantik und Fixpunktsemantik

Beweis $\models_I \mathbf{A}$ impliziert $T(I) \subseteq I$: Angenommen, I ist ein Modell von $\mathbf{A}$. Sei nun $A \in T(I)$; nach der Definition von T gibt es dann eine Grundinstanz $C\sigma$ einer Klausel C aus $\mathbf{A}$ mit

$$C\sigma = A \vee \leftarrow \wedge^*(A_1, \ldots, A_n) \text{ und } A_1, \ldots, A_n \in I$$

Da I ein Modell von $\mathbf{A}$ ist, ist $C\sigma$ wahr in I. Dann ist aber auch $A \in I$, da $\neg A_1, \ldots, \neg A_n$ in I falsch sind. Damit ist gezeigt, dass für alle $A \in T(I)$ gilt, dass $A \in I$.

Gleichheit von modelltheoretischer Semantik und Fixpunktsemantik

Beweis $T(I) \subseteq I$ impliziert $\models_I \mathbf{A}$: Angenommen, I wäre kein Modell von $\mathbf{A}$. Es soll gezeigt werden, dass $T(I) \not\subseteq I$. Es gibt eine Grundinstanz $C\sigma$ einer Klausel C von $\mathbf{A}$ mit $A \leftarrow \wedge^*(A_1, \ldots, A_m), m \geq 0$, die falsch ist. Daher ist $A \notin I$ und $A_1, \ldots, A_n \in I$. Aus letzterem folgt jedoch nach der Definition von T, dass $A \in T(I)$. Also ist $T(I) \not\subseteq I$.

□

Anmerkung: Daraus lässt sich folgern, dass $\cap\mathbf{C}(\mathbf{A})$ unter T abgeschlossen ist.

Gleichheit der operationalen Semantik und der Fixpunktsemantik

Die Gleichheit $\mathbf{D}_1 = \mathbf{D}_3$ folgt aus den Gleichheiten $\mathbf{D}_1 = \mathbf{D}_2$ und $\mathbf{D}_2 = \mathbf{D}_3$. Die Bedeutung dieser Gleichheit ist abhängig von dem $\mathbf{D}_1$ bestimmenden Inferenzsystem. Wir betrachten nun ein anderes Top-Down-Inferenzsystem mit zwei Regeln; zum einen die Regel, dass Eingabeklauseln zu Grundformeln instanziiert werden können, zum anderen die *Hyperresolutionsregel*:

Gleichheit der operationalen Semantik und der Fixpunktsemantik

Eine atomare Formel A ist der *Hyperresolvent* der Formelmenge

$$\{A \leftarrow \wedge^*(A_1, \ldots, A_n)\} \cup \{A_1\} \cup \cdots \cup \{A_n\}$$

A wird aus diesen Formeln durch *Hyperresolution* abgeleitet.

Operationale Semantik zur Hyperresolution

- Nun: Beweis der Gleichheit der operationalen Semantik und der Fixpunktsemantik
- Entspräche der Gleichheit des Top-Down- mit dem Bottom-Up-Systems
- Hyperresolution sehr ähnlich zur Transformation T der Fixpunktsemantik; daher folgt:

Operationale Semantik zur Hyperresolution

Eine Formel A ist aus Grundinstanzen einer Formelmenge $\mathbf{A}$ per Hyperresolution ableitbar genau dann wenn $A \in \bigcup_{m=0}^{\infty} T^m(\emptyset)$, wobei $T^0(\emptyset) = \emptyset$ und $T^{m+1}(\emptyset) = T(T^m(\emptyset))$.

Die zu diesem Inferenzsystem zugehörige operationale Semantik ist also definiert als:

$$\mathbf{D}_1^H(P) = \{(t_1, \ldots, t_n) | P(t_1, \ldots, t_n) \in \bigcup_{m=0}^{\infty} T^m(\emptyset)\}$$

Gleichheit der operationalen Semantik (Hyperresolution) und der Fixpunktsemantik

Theorem 1 $\mathbf{D}_1^H = \mathbf{D}_3$.

Sei $\mathbf{U}$ eine Abkürzung für $\bigcup_{m=0}^{\infty} T^m(\emptyset)$.

($\mathbf{U} \subseteq \cap\mathbf{C}(\mathbf{A})$) T ist monoton (siehe Abschnitt über Fixpunktsemantik). Sei μ der kleinste Fixpunkt von T. Allgemein lässt sich durch Induktion nach n zeigen, dass $T^n(\emptyset) \subseteq \mu$ für alle n. Daher ist auch $\mathbf{U} \subseteq \mu$.

($\cap\mathbf{C}(\mathbf{A}) \subseteq \mathbf{U}$) Dies wird gezeigt, indem wir zeigen, dass $\mathbf{U}$ unter T abgeschlossen ist, da dann $\mathbf{U} \in \mathbf{C}(\mathbf{A})$, und daraus folgt $\cap\mathbf{C}(\mathbf{A}) \subseteq \mathbf{U}$. Angenommen $A \in T(\mathbf{U})$. Nach der Definition ist A entweder eine Instanz einer

Zusicherung; dann ist $A \in \mathbf{U}$, da $A \in T^m(\emptyset), m > 0$. Oder es gibt eine instanziierte Klausel $A \leftarrow \wedge^*(A_1, \ldots, A_n)$ und $A_1, \ldots, A_n \in \mathbf{U}$. Dann gibt $A_1, \ldots, A_n \in T^N(\emptyset), N \geq 0$, und damit $A \in T^{N+1}(\emptyset)$, und daher $A \in \mathbf{U}$. Daher ist $\mathbf{U}$ unter T abgeschlossen.
□

Konsequenzen aus den Ergebnissen in der Semantik

- Operationale Semantik als Spezialfall der Beweistheorie
- Fixpunktsemantik als Spezialfall der Modelltheorie
- Allgemeinerer Blick auf die Semantiken
- Anwendbarkeit von Regeln aus der Logik z.B. zur Verifikation von Programmen

Bibliographie

- Robert A. **Kowalski**: Predicate Logic as Programming Language. IFIP Congress 1974: 569-574
- Maarten H. **van Emden**, Robert A. **Kowalski**: The Semantics of Predicate Logic as a Programming Language. J. ACM 23(4): 733-742 (1976)