Introduction to Data and Knowledge Engineering SS09 – SLD

TECHNISCHE UNIVERSITÄT DARMSTADT

SLD Abarbeitung

TECHNISCHE UNIVERSITÄT DARMSTADT

Ziel: eine einfache Datenstruktur für die Abarbeitung von Queries in Prolog.

Motivation:

- Wiederholung: Wie bearbeitet Prolog Queries?
- Wenn ich SLD programmieren müsste, wie könnte ich es angehen
- Welche Schritte/Zusatzinformationen unterschlagen
  - SLD-Suchbaum aus Übung
  - trace-Ausgabe des Interpreter

  und wie kann ich diese beiden Darstellung aufeinander abbilden?

SLD Abarbeitung
Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

Beispielprogramm

```
a(X) :- b(X),c(X).
b(1).
b(2).
c(2).
```

Abarbeitung der Query `a(Y).`

	Teilziele	Zähler	Subst.	
0	a(Y).	0/1		↓
1	b(_1),c(_1).	0/2	Y/_1.	↓
2	b(1),c(1).	0/1	_1/1.	↓
3	c(1).	0/0		↑
2	b(1),c(1).	1/1	_1/1.	↑
1	b(_1),c(_1).	1/2	Y/_1.	↓
2	b(2),c(2).	0/1	_1/2.	↓
3	c(2).	0/1		↓
4	.	0/0		→

SLD Abarbeitung

Zustände

TECHNISCHE UNIVERSITÄT DARMSTADT

Globaler Zustand C der Abarbeitung besteht aus

- Kette von lokalen Zuständen $C = (Q_1, \ldots, Q_r)$
- letzter lokale Zustand Q_r ist *aktiver Zustand*

Lokaler Zustand Q besteht aus

- Liste von Teilzielen (Literalen) $L = L_1, \ldots, L_s$.
- Zähler k/K: die k ersten der K möglichen Auslösung von L_1 wurden schon probiert
- Liste von Substitionen $S = \theta_1, \ldots, \theta_t$, die beim Erzeugen von Q verwendet wurden

Beispiel globaler Zustand:

```
a(Y).            0/1
b(_1),c(_1).     0/2   Y/_1.
b(1),c(1).       0/1   _1/1.
```

Beispiel lokaler Zustand:

```
b(_1),c(_1).     0/2   Y/_1.
```

Redundanzen:

- Substitionen S für Q_i ergeben sich aus $Q_1 \ldots Q_{i-1}$ und dem Programm
- K ergibt sich aus L_1 und dem Programm

SLD Abarbeitung
Zustandsübergänge

TECHNISCHE UNIVERSITÄT DARMSTADT

Übergang von globalen Zustand C mit aktiven Zustand Q in globalen Zustand $\bar{C}$:

- falls $k < K$ in Q, wird ein lokaler Zustand $\bar{Q}$ angehängt ($\downarrow$)
- falls $k = K$ in Q, wird Q abhängt/gelöscht ($\uparrow$)

Anängen:

- ersetze erstes Literal L_1 von Q durch Body-Literale der $k+1$-ten Regel, deren Head mit L_1 unifiziert werden kann
- die Regel bekommt dafür immer frische Variablen
- Substitionen werden auf alle Literale von Q angewandt
- setze Zähler von $\bar{Q}$ auf $0/\bar{K}$, wobei $\bar{K}$ die Zahlöglichen Unifikationen für das erste Literal $\bar{L}_1$ von $\bar{Q}$ ist

Abängen:

- nach dem Abängen wird Zähler k jetzt aktiven Zustands wird um Eins erhöht

SLD Abarbeitung
Ausstieg

TECHNISCHE UNIVERSITÄT DARMSTADT

Ende der Abarbeitung:

- ist Liste der Literale L im aktiven Zustand leer, wurde eine Lösung für die Query gefunden ($\rightarrow$)
- gibt user ';' ein, wird weiter abgearbeitet: der aktive Zustand wird abgehängt
- sonst wird Abarbeitung abgebrochen
- wenn erste Zustand der Kette C abgehängt wurde, ist die Abarbeitung beendet

Substitutionen in lokalen Zustand Schreiben:

- Ziel ist es, im Zustand, der eine Lösung findet, einfach an die konkreten Werte der Query-Variablen zu kommen
- wende iterativ die Substitionen entlang der Kette an

SLD Abarbeitung
Suchbaum und Zustände

TECHNISCHE UNIVERSITÄT DARMSTADT

Die Abarbeitung entspricht Traversierung eines Baumes.

- ein Knoten ist ein lokale Zustand Q
- K ist die Anzahl der Kinder von Q und k gibt an wieviele davon schon besucht wurden
- globaler Zustand C entspricht einem Pfad, der in der Wurzel beginnt
- Lösungen werden in Blättern gefunden
- nicht jedes Blatt ist eine Lösung

SLD Abarbeitung
Suchbaum Darstellung in der Übung

TECHNISCHE UNIVERSITÄT DARMSTADT

Darstellung der Suche in der Übung (kürzere Form):

```
a(Y) → b(Y),c(Y)
     b(Y) → Y=1, Exit
     c(1) → Fail
     b(Y) → Y=1, Exit
     c(2) → Exit
```

- entsteht durch Substiution ein Fakt, wurde dessen Auflösung übersprungen
- nur erstes Literal wird wiederholt notiert
- backtracking/Äbhängen (↑) wird nicht explizit notiert
- neue Variablen werden nur verwendet, wenn nötig
- für `b(Y) → Y=1, Exit` hatten wir auch `b(Y) → Exit:{Y/1}` geschrieben

SLD Abarbeitung

Trace Ausgabe eines Interpreters

TECHNISCHE UNIVERSITÄT DARMSTADT

GNU-Prolog 1.3.0

- `Redo:` nächste mögliche Unifikation wird verwendet
- aber statt `Redo:b(_16)` steht dort `Redo:b(1)`, d.h letztes *Unifikat* anstelle des *Unifikanten*
- backtracking (↑) wird immer explizit ausgegeben

```
| ?- a(Y).
   1        1  Call:  a(_16) ?
   2        2  Call:  b(_16) ?
   2        2  Exit:  b(1) ?
   3        2  Call:  c(1) ?
   3        2  Fail:  c(1) ?
   2        2  Redo:  b(1) ?
   2        2  Exit:  b(2) ?
   3        2  Call:  c(2) ?
   3        2  Exit:  c(2) ?
   1        1  Exit:  a(2) ?
```