

Logische und funktionale Programmierung

Vorlesung 8: Prolog - Einflußnahme auf das Backtracking

Babeş-Bolyai Universität, Department für Informatik, Cluj-Napoca
csacarea@cs.ubbcluj.ro

24. November 2017



BACKTRACKING:

SUCHE NACH EINER LÖSUNG

- Zur Erinnerung: Bisher haben wir Anfragen an die Wiba gestellt, die mit der Meldung **IC-Verbindung existiert** (im Fall der Ableitbarkeit des Goals) bzw. **IC-Verbindung existiert nicht** (im Fall der Nicht-Ableitbarkeit des Goals) beantwortet wurden.
- Bei der Ableitbarkeits-Prüfung setzte **Backtracking** einzig und allein dann ein, wenn es um den Nachweis ging, ob überhaupt eine Ableitung **möglich** ist.
- Bei einer erfolgreichen Ableitung wurde die Programmausführung beendet, ohne dass ein weiteres Backtracking erfolgte.



BACKTRACKING:

SUCHE NACH ALLEN LÖSUNGEN

- Im folgenden werden wir darstellen, wie das Backtracking der Inferenzkomponente beeinflusst werden kann, um z.B. Fragestellungen nach **sämtlichen** Lösungsalternativen - als insgesamt aus der Wiba ableitbarem Wissen über IC-Verbindungen - bearbeiten lassen zu können.
- Dazu betrachten wir die folgende Aufgabenstellung:
- Durch die Angabe eines Abfahrtsorts sollen **alle möglichen Ankunftsorte** angezeigt werden, die über IC-Verbindungen erreicht werden können



BEISPIEL

- Anfrage nach IC-Verbindungen; Anforderung zur Eingabe von Abfahrts- und Ankunftsort über internes Goal mit dem Prädikat `anfrage`;



BEISPIEL

dic(ha,kö).

dic(ha,fu).

dic(kö,ka).

dic(kö,ma).

dic(fu,mü).

dic(ma,fr).

ic(Von,Nach):-dic(Von,Nach).

ic(Von,Nach):-dic(Von,Z),ic(Z,Nach).

anfrage:-write(' Gib Abfahrtsort: '),nl,

 ttyread(Von),

 write(' Gib Ankunftsart: '),nl,

 ttyread(Nach),

 ic(Von,Nach),

 write(' IC-Verbindung existiert ').

anfrage:-write(' IC-Verbindung existiert nicht ').

:- anfrage.



ERKLÄRUNGEN

- Das PROLOG-System fordert durch ?- immer ein externes Goal an, sofern es nicht durch den Eintrag eines **internen Goals** innerhalb des PROLOG-Programms daran **gehindert** wird.
- Wir legen ein internes Goal dadurch fest, dass wir die Zeichen :- ans Ende der Wiba und dahinter - mit abschließendem Punkt . - geeignete Prädikate für eine Anfrage eintragen, die beim Programmstart als Goal aufgefaßt werden soll.



ERKLÄRUNGEN

- Das PROLOG-System fordert durch ?- immer ein externes Goal an, sofern es nicht durch den Eintrag eines **internen Goals** innerhalb des PROLOG-Programms daran **gehindert** wird.
- Wir legen ein internes Goal dadurch fest, dass wir die Zeichen :- ans Ende der Wiba und dahinter - mit abschließendem Punkt . - geeignete Prädikate für eine Anfrage eintragen, die beim Programmstart als Goal aufgefaßt werden soll.
- **:-anfrage.**



ERKLÄRUNGEN

- Um einen Bezug zu unseren bisherigen Regeln herzustellen, muss das Prädikat `anfrage` der Kopf einer Regel sein, in deren Rumpf das Prädikat `ic` - zusammen mit weiteren Prädikaten - für den Dialog mit dem Anwender enthalten ist.
- Wichtig für den Dialog mit dem Anwender ist das Standard-Prädikat `write`, mit dem sich Text auf dem Bildschirm anzeigen lässt.
- Dieser Text muss als Argument des Prädikats `write` aufgeführt werden.
- Er wird in dem Augenblick ausgegeben, in dem das Prädikat `write` **unifiziert** wird.



ERKLÄRUNGEN

- Bei der Unifizierung des Prädikats `write(Gib Abfahrtsort:)` wird der Text **Gib Abfahrtsort:** am Bildschirm angezeigt.

```
anfrage:- write(' Gib Abfahrtsort: '),nl,
          ttyread(Von),
          write(' Gib Ankunftsart: '),nl,
          ttyread(Nach),
          ic(Von,Nach),
          write(' IC-Verbindung existiert ').
```



ERKLÄRUNGEN

- Bei der Unifizierung des Prädikats `ttyread` wird eine Eingabe von der Tastatur angefordert.
- Die als Argument aufgeführte Variable `Von` wird mit demjenigen Wert instanziiert, der über die Tastatur eingegeben wird.
- Das Prädikat `n1` beeinflusst die Bildschirmausgabe.
- Die Unifizierung dieses Prädikats, das kein Argument besitzt, hat zur Folge, dass der Cursor auf den Anfang der nächsten Bildschirmzeile positioniert wird.



BACKTRACKING

- Die Standard-Prädikate `write`, `ttyread` und `nl` von der Inferenzkomponente werde **niemals** als Backtracking-Klauseln markiert, da sie keine Alternativen für eine Unifizierung zulassen.
- Diese Prädikate sind **nicht Backtracking fähig**



ERWEITERUNG 1

WIE WERDEN MÖGLICHE ZWISCHENSTATIONEN GEZEIGT?

```
ic(Von,Nach):-dic(Von,Nach).  
ic(Von,Nach):-dic(Von,Z),  
write(mögliche Zwischenstation: ),  
write(Z),nl, ic(Z,Nach).
```



ERWEITERUNG 2

- Wie können wir neben den möglichen Zwischenstationen zusätzlich die Nummern der unifizierten Fakten (von 1 bis 6) sowie die Nummern der unifizierten Regelköpfe (1 oder 2) anzeigen lassen?



ERWEITERUNG 2

- Wie können wir neben den möglichen Zwischenstationen zusätzlich die Nummern der unifizierten Fakten (von 1 bis 6) sowie die Nummern der unifizierten Regelköpfe (1 oder 2) anzeigen lassen?
- Wir verändern die Prädikate `dic` und `ic` indem wir jeweils ein zusätzliches Argument mit der zugehörigen Positionsnummer als 1. Argument einfügen.
- Zusätzlich tragen wir in die Regeln mit dem Prädikatsnamen `ic` die Prädikate `write` und `nl` zur Ausgabe der jeweiligen Regel-Nummer an den Anfang des Regelrumpfes ein.



ERWEITERUNG 2

```
dic(1,ha,kö).  
dic(2,ha,fu).  
dic(3,kö,ka).  
dic(4,kö,ma).  
dic(5,fu,mü).  
dic(6,ma,fr).
```

```
ic(1,Von,Nach):-write(' Regel: 1 '),nl,  
                dic(Nr,Von,Nach),  
                write(' Fakt: '),write(Nr),nl.  
ic(2,Von,Nach):-write(' Regel: 2 '),nl,  
                dic(Nr,Von,Z),  
                write(' Fakt: '),write(Nr),nl,  
                write(' mögliche Zwischenstation: '),write(Z),nl,  
                ic( _ ,Z,Nach).
```



ERWEITERUNG 3

- Anforderung zur Eingabe von Abfahrts- und Ankunftsort über internes Goal mit dem Prädikat `anfrage`;
- Ausgabe der Fakten-Nummern und der Regel-Nummern, sowie der möglichen Zwischenstationen als Erklärung für die Arbeit der Inferenzkomponente;



ERWEITERUNG 3

```
dic(1,ha,kö).  
dic(2,ha,fu).  
dic(3,kö,ka).  
dic(4,kö,ma).  
dic(5,fu,mü).  
dic(6,ma,fr).
```

```
ic(1,Von,Nach):-write(' Regel: 1 '),nl,  
    dic(Nr,Von,Nach),  
    write(' Fakt: '),write(Nr),nl.  
ic(2,Von,Nach):-write(' Regel: 2 '),nl,  
    dic(Nr,Von,Z),  
    write(' Fakt: '),write(Nr),nl,  
    write(' mögliche Zwischenstation: '),write(Z),nl,  
    ic(-,Z,Nach).
```

```
anfrage:-write(' Gib Abfahrtsort: '),nl,  
    ttyread(Von),  
    write(' Gib Ankunftsart: '),nl,  
    ttyread(Nach),  
    ic(-,Von,Nach),  
    write(' IC-Verbindung existiert ').  
anfrage:-write(' IC-Verbindung existiert nicht ').
```

```
:- anfrage.
```



SUCHE NACH ALLEN LÖSUNGEN

- Durch die Angabe eines Abfahrtsorts sollen **alle möglichen Ankunftsorte** angezeigt werden, die über IC-Verbindungen erreicht werden können
- Die zugehörige Lösung entwickeln wir aus dem Programm mit dem wir die Existenz einer IC-Verbindung zwischen einem Abfahrts- und einem Ankunftsart prüfen konnten.
- Dazu sind Änderungen bei der Formulierung des internen Goals und der Regeln mit dem Regelkopf anfrage erforderlich.



```
anfrage:-write(' Gib Abfahrtsort: '),nl,  
        ttyread(Von),  
        write(' Gib Ankunftsort: '),nl,  
        ttyread(Nach),  
        ic(Von,Nach),  
        write(' IC-Verbindung existiert '),nl.  
anfrage:-write(' IC-Verbindung existiert nicht '),nl.
```



- Kein Ankunftsort \rightarrow löschen des 4., 5. und 6. Prädikats.
- Was passiert jetzt?



- Kein Ankunftsort \rightarrow löschen des 4., 5. und 6. Prädikats.
- Was passiert jetzt? Variable `Nach` wird jetzt nicht mehr instanziiert.
- Das Subgoal `ic(Von, Nach)` wird immer ableitbar ist, wenn es eine weitere Instanzierung für die Variable `Nach` gibt.
- Damit derartige Instanzierungen - bei einem internen Goal - angezeigt werden, können wir das Standard-Prädikat `write` mit der Variablen `Nach` als Argument hinter dem Prädikat `ic(Von, Nach)` im Regelrumpf aufführen.



```
anfrage:-write(' Gib Abfahrtsort: '),nl,  
        ttyread(Von),  
        write(' mögliche(r) Ankunftsort(e): '),nl,  
        ic(Von,Nach),  
        write(Nach),nl.  
anfrage:-nl,write(' Ende ').
```



Gib Abfahrtsort:

kö.

mögliche(r) Ankunftsort(e):

ka

yes



- Eine Antwort wird geliefert.
- Wie erzwingen wir **alle** Antworten ohne einen neuen Programmstart?
- Wir lösen dieses Problem durch den Einsatz des argumentlosen Standard-Prädikats `fail`, bei dem jeder Unifizierungs Versuch fehlschlägt.
- `:- anfrage, fail.`



- Dadurch wird - sofern das 1. Subgoal `anfrage` erstmals abgeleitet wurde - durch das 2. Subgoal `fail` ein Backtracking erzwungen, d.h. es wird zur letzten Backtracking-Klausel - also zur abgeleiteten Klausel mit dem Klauselkopf `ic (Von, Nach)` im Regelrumpf der 1. Regel des Prädikats `anfrage` - zurückgekehrt und von dort aus versucht, das Prädikat `ic (kö, Nach)` mit einer alternativen Instanzierung der Variablen `Nach` erneut zu unifizieren.



- Ist dieser Unifizierungs-Versuch wiederum erfolgreich, so wird der instanziierte Wert der Variablen `Nach` als nächster, möglicher Ankunftsort angezeigt und daraufhin durch das Prädikat `fail` ein erneutes Backtracking **erzwungen**.
- Dieser Prozess wird solange wiederholt, bis es keine weiteren Instanzierungen der Variable `Nach` mehr gibt, mit der sich das Prädikat `ic(kö, Nach)` ableiten läßt.
- Daraufhin wird ein Backtracking beim Prädikat `anfrage` durchgeführt, so dass der Text `Ende` angezeigt und die Ableitbarkeits - Prüfung des internen Goals `anfrage, fail.` erfolglos beendet wird.



dic(ha,kö).
dic(ha,fu).
dic(kö,ka).
dic(kö,ma).
dic(fu,mü).
dic(ma,fr).

ic(Von,Nach):-dic(Von,Nach).
ic(Von,Nach):-dic(Von,Z),ic(Z,Nach).

anfrage:-write(' Gib Abfahrtsort: '),nl,
 ttyread(Von),
 write(' mögliche(r) Ankunftsort(e): '),nl,
 ic(Von,Nach),
 write(Nach),nl.
anfrage:-nl,write(' Ende ').

:- anfrage, fail.



ERWEITERUNG 4

- Es sollen **alle** möglichen IC-Verbindungen zwischen **allen** möglichen Abfahrts- und Ankunftsorten angezeigt werden.
- Die erste Regel wird geändert:
- `anfrage:-write(mögliche IC-Verbindung(en) :
) , nl ,
ic(Von , Nach) ,
write(Abfahrtsort :) , write(Von) , nl ,
write(Ankunftsort :) , write(Nach) , nl .`



- Läßt sich nämlich eine IC-Verbindung aus der Wiba ableiten, so wird diese IC-Verbindung unmittelbar in die dynamische Wiba eingetragen.
- Dabei wird nicht geprüft, ob der abgeleitete Fakt bereits Bestandteil der dynamischen Wiba ist.

Aufgabe

Eine IC-Verbindung, die bereits als Fakt in der dynamischen Wiba enthalten ist, soll nicht erneut in die dynamische Wiba eingetragen werden. Außerdem sollen bereits abgeleitete IC-Verbindungen der dynamischen Wiba entnommen und nicht von neuem aus der statischen Wiba abgeleitet werden.



DYNAMISCHE WIBA SICHERN

- Ein abgeleitetes Faktum aus der statischen Wiba kann in einer dynamischen Wiba aufgenommen werden, um es für weitere Inferenzen bereitzustellen.
- Für die Übernahme von Fakten in die dynamische Wiba stehen die Standard-Prädikate `asserta` und `assertz` zur Verfügung.
- Bei der Unifizierung wird das Argument von `asserta` (`assertz`) **vor (hinter)** allen anderen Fakten in die dynamische Wiba eingefügt.
- Soll eine Regel in die dynamische Wiba eingefügt werden, so sind diese Prädikate mit 2 Argumenten - zur Angabe des Regelkopfs und des Regelrumpfs - zu verwenden.



DYNAMISCHE WIBA SICHERN

- Für das Eintragen bereits abgeleiteter IC-Verbindungen in die dynamische Wiba und den späteren Zugriff auf diese IC-Verbindungen führen wir ein neues Prädikat ein `ic_db`.
- Durch den Einsatz des Standard-Prädikats `asserta` läßt sich die Tatsache, dass `ic(ha, fr)` aus der (statischen) Wiba ableitbar ist, in der Form `asserta(ic_db(ha, fr))` als Fakt in die dynamische Wiba eintragen.
- Bei der Unifizierung des Standard-Prädikats `asserta` wird der Fakt `ic_db(ha, fr)` in die dynamische Wiba übernommen.
- Bei einer späteren Anfrage nach einer IC-Verbindung von `ha` nach `fr` kann diese Anfrage mit dem in die dynamische Wiba eingetragenen Fakt `ic_db(ha, fr)` in Verbindung gebracht werden, so dass keine erneute, schrittweise Ableitung vorgenommen werden muss.



- Wir erweitern nun unsere Prolog-Programme, so dass abgeleitete Fakten aus der Wiba mitgenutzt werden.

```
anfrage:-write( Gib Abfahrtsort: ),nl,  
read(Von),  
write( Gib Ankunftsort: ),nl,  
read(Nach),  
verb(Von,Nach),  
write( IC-Verbindung existiert ),nl,  
asserta(ic db(Von,Nach)).  
anfrage:-write( IC-Verbindung existiert nicht  
) ,nl.
```



- Falls eine IC-Verbindung über eine oder mehrere Zwischenstationen ableitbar ist, wird anschließend das Standard-Prädikat `asserta` unifiziert und folglich das im Argument aufgeführte Prädikat als Fakt mit dem Prädikatsnamen `ic_db` und den Zwischenstationen als instanziierten Variablenwerten in die dynamische Wiba eingetragen.

PRÄDIKAT VERB

- Suchen in der statischen *oder* dynamischen Wiba.
- Durch die 1. Regel wird eine Ableitung aus der dynamischen Wiba und durch die 2. Regel eine Ableitung aus der statischen Wiba versucht.

```
verb(Von,Nach) :- ic db(Von,Nach),  
write( ableitbar aus dynamischer Wiba ), nl.  
verb(Von,Nach) :- ic(Von,Nach) .
```



- Wollen wir uns anschließend die jeweils abgeleiteten IC-Verbindungen, die als Fakten in die dynamische Wiba aufgenommen wurden, anzeigen lassen, so setzen wir das Standard-Prädikat `listing` ein:

```
? listing(ic db).
```

- Durch die Unifizierung dieses Prädikats werden alle Klauseln mit dem Prädikatsnamen `ic_db` angezeigt, die aktuell in der Wiba enthalten sind.



- Problem: Jedes Mal wenn wir das Programm beenden, geht der Inhalt der Wiba verloren.
- Aufgabe: Sicherung der dynamischen Wiba in einer Sicherungs-Datei.
- Lösung: Einsetzen der Prädikate `tell`, `listing`, `told`.

```
sichern_ic:-ic db(_ ,_ ), tell(ic.pl),  
listing(ic db), told.  
sichern_ic.
```



- Wir überprüfen **vor** einer Übertragung, ob in der Wiba **überhaupt** Klauseln mit dem Prädikatsnamen `ic_db` vorhanden sind.
- Durch die Unifizierung des Prädikats `tell(ic.pl)` werden alle nachfolgenden Ausgaben des PROLOG-Systems solange in die Datei `ic.pl` übertragen, bis das Prädikat `told` unifiziert wird.
- Da durch die vorausgehende Unifizierung des Prädikats `listing(ic_db)` alle Klauseln mit dem Prädikatsnamen `ic_db` angezeigt werden, wird durch die Ableitung des Prädikats `sichern_ic` die Übertragung aller Klauseln mit dem Prädikatsnamen `ic_db` in die Sicherungsdatei `ic.pl` durchgeführt.



- Um den Inhalt einer Sicherungs-Datei in die aktuelle Wiba zu übernehmen, setzen wir das Standard-Prädikat `consult` in der Form `consult(ic.pl)` ein.
- Durch die Unifizierung dieses Prädikats werden zunächst alle diejenigen Klauseln aus der aktuellen Wiba gelöscht, für die gleichnamige Klauselköpfe in der Sicherungs-Datei `ic.pl` enthalten sind.
- Anschließend wird die Wiba um den Inhalt von `ic.pl` ergänzt



- Vor dem Zugriff auf den Inhalt einer Sicherungs-Datei sollte überprüft werden, ob die Datei auch tatsächlich vorhanden ist.
- Dazu läßt sich das Standard-Prädikat `exist_file` in der Form
`exist_file(ic.pl) .`
einsetzen.
- Dieses Prädikat läßt sich in dieser Form nur dann unifizieren, wenn die Datei `ic.pl` existiert und auf ihren Inhalt ein lesender Zugriff erlaubt ist.



- Für die Übertragung der Fakten aus der Sicherungs-Datei `ic.pl` vereinbaren wir die folgenden Klauseln:
`lesen:-exists('ic.pl'),`
`consult(ic.pl).`
`lesen.`
- Die zweite Klausel haben wir deswegen angegeben, damit in der Situation, in der die Datei `ic.pl` noch nicht existiert oder auf ihren Inhalt kein Lese-Zugriff erlaubt ist, der Regelkopf trotzdem - durch Backtracking - ableitbar ist.
- Das interne Goal wird zu
`:- lesen, anfrage, sichern_ic.`



- Der Nachteil dieses Programms besteht darin, dass jedesmal, wenn eine Anfrage nach einer IC-Verbindung positiv beantwortet wird, ein Eintrag in die dynamische Wiba durch das Standard-Prädikat `asserta` vorgenommen wird.
- Dies führt dazu, dass die dynamische Wiba bei jeder erfolgreichen Ableitung um einen Fakt erweitert wird, so dass bereits abgeleitete IC-Verbindungen **mehrfach** in die dynamische Wiba eingetragen werden.
- Dies kann man mit Hilfe des Prädikats `cut` unterbinden.



ANFRAGE

- Eine IC-Verbindung, die bereits als Fakt in der dynamischen Wiba enthalten ist, wird nicht erneut in die dynamische Wiba eingetragen.
- Ausserdem sollen bereits abgeleitete IC-Verbindungen der dynamischen Wiba entnommen und nicht von neuem aus der statischen Wiba abgeleitet werden .
- Dazu muss das Prädikat `anfrage` geeignet geändert werden.
- Wir verwenden dazu das Prädikat `not: \+`.



- `Not` ist genau dann ableitbar, wenn das als Argument aufgeführte Prädikat in der Wiba nicht unifiziert werden kann.
- Das Prädikat `not` kann nicht dazu eingesetzt werden, um für ein Prädikat Variablen-Instanzierungen zu bestimmen, die sich nicht aus der Wiba ableiten lassen.



- Ist also das Prädikat $\text{verb}(\text{Von}, \text{Nach})$ ableitbar, so ist - vor einem Eintrag in die dynamische Wiba - zu prüfen, ob das Prädikat $\text{ic_db}(\text{Von}, \text{Nach})$ für die aktuellen Instanzierungen der Variablen Von und Nach bereits als Fakt in der dynamischen Wiba enthalten ist.
- Dazu fügen wir das Prädikat $\setminus+(\text{ic_db}(\text{Von}, \text{Nach}))$ wie folgt in die oben angegebenen Regeln ein:



```
anfrage:-write( Gib Abfahrtsort: ),nl,  
read(Von),  
write( Gib Ankunftsart: ),nl,  
read(Nach),  
verb(Von,Nach),  
write( IC-Verbindung existiert ),nl,  
\+(ic_db(Von,Nach)),  
asserta(ic_db(Von,Nach)).  
anfrage:-write( IC-Verbindung existiert nicht  
,nl.
```



- In dieser Situation läßt sich das Prädikat $\setminus +$ nur dann unifizieren, wenn $ic_db(Von, Nach)$ mit den aktuellen Instanzierungen der Variablen Von und $Nach$ nicht aus der dynamischen Wiba ableitbar ist.
- In diesem Fall wird das Prädikat $ic_db(Von, Nach)$ (mit den aktuellen Instanzierungen der Variablen Von und $Nach$) durch die nachfolgende Unifizierung des Standard-Prädikats $asserta$ als Fakt in die dynamische Wiba aufgenommen.



- Ist dagegen $ic_db(Von, Nach)$ - z.B. für die Instanzierungen der Variablen Von mit ha und der Variablen $Nach$ mit mu - bereits als Fakt in der dynamischen Wiba enthalten, so kann $ic_db(ha, mu)$ unifiziert und damit $\setminus+(ic_db(ha, mu))$ **nicht** abgeleitet werden.
- Daraufhin setzt Backtracking ein.
- Dies führt dazu, dass das Prädikat $verb(ha, mu)$ - durch ein daraufhin eingeleitetes Backtracking - mit dem Kopf der 2. Regel des Prädikats $verb(Von, Nach)$ $verb(Von, Nach) :- ic(Von, Nach)$. unifiziert wird, woraufhin die Ableitbarkeit des Prädikats $ic(ha, mu)$ aus den Fakten der (statischen) Wiba erneut nachgewiesen wird.



- Im Anschluss daran erweist sich wiederum das Subgoal $\backslash+(ic_db(ha,mu))$ als nicht unifizierbar.
- Da kein weiteres Backtracking innerhalb der 1. Regel (mit dem Prädikat `anfrage` im Regelkopf) mehr möglich ist, erfolgt Backtracking zur 2. Regel mit dem Prädikat `anfrage` im Regelkopf, woraufhin der Text IC-Verbindung existiert nicht angezeigt wird.
- Dies ist jedoch nicht das von uns erwünschte Ergebnis.



- Damit die Ableitbarkeits-Prüfung nicht in der eben beschriebenen Form durchgeführt wird, muss das Backtracking zum Subgoal `verb` und das anschließende Backtracking zur 2. Regel des Prädikats `anfrage`, das durch die Nicht-Ableitbarkeit des Prädikats `\+` erzwungen wird, verhindert werden.
- Zur Einschränkung des Backtrackings steht das Standard-Prädikat `cut` zur Verfügung, das in einem PROLOG-Programm durch das Ausrufungszeichen `!` gekennzeichnet wird.



- Das Prädikat **cut** läßt sich beim ersten Ableitbarkeits-Versuch - **von links kommend** - erfolgreich unifizieren.
- Wird das Prädikat **cut** unifiziert, so sind alle alternativen Regeln - mit dem gleichen Prädikat im Regelkopf - durch Backtracking nicht mehr erreichbar (sie sind gesperrt).
- Dies bedeutet, dass keine Alternativen zur aktuellen Regel betrachtet werden können, so dass das aktuelle Parent-Goal nur über die aktuelle Regel und nicht über alternative Regeln ableitbar ist.



- Folgt dem **cut** ein weiteres Prädikat innerhalb einer logischen UND Verbindung, und schlägt die Ableitbarkeits-Prüfung dieses Prädikats fehl, so ist, wenn durch Backtracking das Prädikat **cut** (*von rechts kommend*) erneut erreicht wird, das Prädikat **cut** bei diesem 2. Ableitbarkeits-Versuch nicht unifizierbar.
- Demzufolge ist kein Backtracking zu einem *links* vom **cut** stehenden Prädikat mehr möglich, so dass das Parent-Goal nicht ableitbar ist.



- Das Prädikat **cut** wirkt also wie eine Mauer, die *von links kommend* überwunden werden kann.
- Wird diese Mauer anschließend *von rechts kommend* - durch Backtracking - erreicht, so kann sie nicht *übersprungen* werden.
- In dieser Situation werden keine Alternativen für ein Backtracking mehr berücksichtigt, so dass das Parent-Goal nicht unifizierbar ist.



```
anfrage:-write( Gib Abfahrtsort: ),nl,  
read(Von),  
write( Gib Ankunftsart: ),nl,  
read(Nach),  
verb(Von,Nach),  
write( IC-Verbindung existiert ),nl,  
!,  
\+(ic db(Von,Nach)),  
asserta(ic db(Von,Nach)).  
anfrage:-write( IC-Verbindung existiert nicht  
) ,nl.
```



- Ist nämlich das Subgoal $\backslash+(ic_db(Von, Nach))$ nicht unifizierbar, so verhindert der **cut** ein Backtracking zum Subgoal $verb(Von, Nach)$ und damit auch ein Backtracking zur 2. Klausel des Prädikats `anfrage`.
- Damit ist der Rumpf der 1. Regel nicht ableitbar.
- Da die 2. Regel durch das erstmalige Ableiten des **cut** gesperrt ist, kann demzufolge der Regelkopf mit dem Prädikat `anfrage` (als Parent-Goal) **nicht** abgeleitet werden.

- Internes Goal: :- lesen,!, anfrage, sichern.ic.
- Dies ist erforderlich, weil sichergestellt werden muss, dass **kein Backtracking** beim Scheitern der Ableitbarkeits-Prüfung des Subgoals anfrage zum Subgoal lesen durchgeführt wird.
- Dies hätte zur Folge, dass ein Backtracking beim Prädikat lesen durchgeführt würde, sofern die Datei ic.pl beim Start des Programms vorhanden ist.
- Weil in diesem Fall das Prädikat lesen erfolgreich abgeleitet werden kann, wurde für das Prädikat anfrage eine **erneute** - nicht erwünschte - Ableitbarkeits-Prüfung durchgeführt.



- Das Standard-Prädikat **cut** hat, nachdem es einmal abgeleitet wurde, Auswirkungen auf die weitere Arbeitsweise des Inferenz-Algorithmus.
- Es besitzt einen **Seiteneffekt**, da es (nachfolgende) alternative Klauseln für Backtracking **sperrt**, so dass das Parent-Goal **nicht über alternative Klauseln ableitbar ist**.



BEISPIEL

b.

c:-fail.

d.

a:~!,b.

a:-d.

test:-a,c.



- Bei der Ableitbarkeits-Prüfung des Goals `test .` wird zunächst das 1. Subgoal `a` mit der 1. Regel mit dem Regelkopf `a` unifiziert.
- Anschließend wird das Standard-Prädikat `cut` unifiziert und dadurch die 2. Klausel mit dem Prädikat `a` im Klauselkopf für Backtracking gesperrt.
- Da das Prädikat `b` als Fakt auftritt, ist das 1. Subgoal `a` ableitbar.
- Da das 2. Subgoal `c` im Regelrumpf des Prädikats `test -` wegen der Regel `c :- fail .` nicht ableitbar ist, musste Backtracking zur 2. Regel mit dem Prädikat `a` im Regelkopf durchgeführt werden.
- Dies ist jedoch nicht möglich, weil dieses Backtracking zuvor durch den `cut` gesperrt wurde, so dass folglich das externe Goal `test .` nicht abgeleitet werden kann.



UNTERBINDEN DES BACKTRACKINGS MIT DEN PRÄDIKATEN CUT UND FAIL

CUT-FAIL-KOMBINATION

- Fail → erschöpfendes Backtracking
- Cut → eingeschränktes Backtracking
- Oftmals besteht Interesse, im Rahmen der Ableitbarkeits-Prüfung das Scheitern eines Parent-Goals herbeizuführen, indem ein noch mögliches Backtracking unterbunden wird.
- Dazu müssen wir eine Kombination der Prädikate **cut** und **fail** als **cut-fail** in der Form von **!,fail.** einsetzen.



UNTERBINDEN DES BACKTRACKINGS MIT DEN PRÄDIKATEN CUT UND FAIL

CUT-FAIL-KOMBINATION

- Wird diese Prädikat-Kombination bei der Ableitbarkeits-Prüfung einer UND-Verbindung erreicht, so wird der Regelrumpf und damit auch der zugehörige Regelkopf, d.h. das Parent-Goal, als **nicht ableitbar** erkannt.



ANWENDUNG

- Es ist ein Programm zu entwickeln, das **solange** eine Eingabe von der Tastatur **anfordert**, bis die Eingabe einer **bestimmten** Text-Konstanten **erfolgt**.



BEISPIEL

```
auswahl(s):-nl,write( Ende ).
auswahl(X):-write( Eingabe von: ),write(X),nl,fail.
anforderung:-write( Gib Buchstabe (Abbruch bei Eingabe von s
):),nl,
read(Buchstabe),
auswahl(Buchstabe),
!,fail.
anforderung:-anforderung.
:- anforderung.
```



PROGRAMMLAUF

- Programm startet, wir geben e ein.
- Regelkopf $\text{auswahl}(X)$ (durch die Instanzierung $X:=e$) wird unifiziert.
- Die Ableitbarkeits-Prüfung des Regelrumpfs schlägt fehl, da er durch das Prädikat fail abgeschlossen wird.



PROGRAMMLAUF

- Da beim Ableiten des internen Goals im 1. Regelrumpf des Prädikats **anforderung** das Prädikat **cut** in diesem Regelrumpf noch nicht erreicht wurde, erfolgt Backtracking zur 2. Regel mit dem Regelkopf **anforderung**.
- Anschließend wird erneut - mit einem neuen Exemplar der Wiba - der Rumpf der 1. Regel mit dem Kopf **anforderung** auf Ableitbarkeit untersucht.
- Dies führt durch die Unifizierung des Subgoals **auswahl(Buchstabe)** zu einer neuen Eingabeanforderung.



PROGRAMMLAUF

- Erst wenn der Buchstabe **s** erstmalig eingegeben wird, erfolgt die Unifizierung des Prädikats **auswahl(s)**, so dass die Prädikat-Kombination **cut-fail** erreicht wird.
- Da der Regelrumpf - wegen des Prädikats **fail** - nicht ableitbar ist und Backtracking zur 2. Regel des Prädikats **anforderung** durch das Prädikat **cut** verhindert wird, ist das Parent-Goal **anforderung** und somit das interne Goal nicht ableitbar.
- Folglich läßt sich die Programmschleife durch die Eingabe des Buchstabens **s** abbrechen.



ROTE UND GRÜNE CUTS

- 1 kopf :- a , ! , b .
- 2 kopf :- c , ! .
- 3 kopf :- d , ! , fail .
- 4 kopf :- ! , e .



Ableitung von "a" möglich:			
Ableitung von "b" möglich	Parent-Goal ableitbar ((2),(3) und (4) sind für (seichtes) Backtracking gesperrt)		
Ableitung von "b" <i>nicht</i> möglich	Parent-Goal <i>nicht</i> ableitbar ((2),(3) und (4) sind für (seichtes) Backtracking gesperrt)		

Ableitung von "a" *nicht* möglich:

Ableitung von "c" möglich	Parent-Goal ableitbar ((3) und (4) sind für (seichtes) Backtracking gesperrt)		
Ableitung von "c" <i>nicht</i> möglich	(seichtes) Backtracking zu (3)		

Ableitung von "c" *nicht* möglich:

Ableitung von "d" möglich	Parent-Goal <i>nicht</i> ableitbar ((4) ist für (seichtes) Backtracking gesperrt)	
Ableitung von "d" <i>nicht</i> möglich	(seichtes) Backtracking zu (4)	
Ableitung von "d" <i>nicht</i> möglich:		
	Ableitung von "e" möglich	Parent-Goal ableitbar
	Ableitung von "e" <i>nicht</i> möglich	Parent-Goal <i>nicht</i> ableitbar



WANN WIRD CUT EINGESETZT?

- Es soll sichergestellt werden, dass nur dann die Möglichkeit bestehen darf, das Parent-Goal durch die aktuell betrachtete Regel abzuleiten, wenn innerhalb des zugehörigen Regelrumpfs ein bestimmtes Prädikat ableitbar ist.
- Dazu muss das Prädikat **cut** unmittelbar links von diesem Prädikat eingefügt werden, so dass nachfolgende alternative Regeln - durch Backtracking - nicht mehr untersucht werden können.
- Das bedeutet insbesondere, dass die vor dem Ableiten des Prädikats **cut** eingegangenen Instanzierungen von Variablen für die weiteren Ableitbarkeits-Prüfungen, die hinter dem Prädikat **cut** durchgeführt werden, nicht wieder gelöst werden können (sie sind **eingefroren**).



ROTER CUT

- Ein derartiger Einsatz des Prädikats **cut** ändert die **deklarative** und **prozedurale** Bedeutung des Prolog-Programms.
- **Roter Cut**



PROZEDURALE BEDEUTUNG EINES PROLOG-PROGRAMMS

- Wird bestimmt durch den Ablauf der Ableitbarkeitsprüfung.
- Der Inferenz-Algorithmus kann bei einer veränderten Reihenfolge der Klauseln oft auch einen anderen Ablauf nehmen, kann sich die prozedurale Bedeutung eines PROLOG-Programms entsprechend ändern → endlos Schleife.



DEKLARATIVE BEDEUTUNG EINES PROLOG-PROGRAMMS

- Wird durch die Prädikate in den einzelnen Klauseln bestimmt, unabhängig von der Reihenfolge in der die Klauseln angegeben sind.
- Die deklarative Bedeutung besteht somit darin, **ob** und nicht **wie** ein Goal ableitbar ist.



ZUSAMMENFASSUNG: DEKLARATIVE UND PROZEDURALE BEDEUTUNG

Der **deklarative Aspekt** eines Prolog Programms betrifft nur die Relationen, die im Programm definiert sind.

Der **prozedurale Aspekt** betrifft die Art und Weise wie der Output des Programms zustande kommt.

Vorteil von Prolog:

übernimmt weite Teile der prozeduralen Aspekte selbst. Damit kann sich der Programmierer auf die normalerweise leichter zu verstehenden deklarativen Aspekte konzentrieren. Späterhin kommt man jedoch nicht umhin sich auch mit dem internen Programmablauf zu befassen und diesen etwa durch geschickte Anordnung der Klauseln oder direkte Eingriffe wie den Cut zu beeinflussen.



DEKLARATIV - WAS

Gegeben ein Programm und ein goal G , dann ist G wahr (oder aus dem Programm erfüllbar) gdw.

- 1 es existiert eine Klausel C im Programm, so dass
- 2 es existiert eine Instanz I von C , so dass
 - 1 das Kopf von I ist identisch mit G , und
 - 2 alle goals aus dem Rumpf von I wahr sind.



PROZEDURAL - WIE

Die Beantwortung einer Frage = Liste von goals zu erfüllen: die Variablen werden so instantiiert, dass alle goals logisch aus dem Programm folgen.

Die prozedurale Steuerung in Prolog ist eine Prozedur für eine Liste von goals.

EXECUTE bedeutet: versuche die goals zu erfüllen.



GRÜNER CUT

- Neben der Unterbindung des Backtrackings, weil nur so die gegebene Aufgabenstellung gelöst werden kann, gibt es einen weiteren Grund, das Prädikat **cut** in einem PROLOG-Programm oder einem Goal einzusetzen.
- Oftmals kann durch das Prädikat **cut** eine Effizienzsteigerung bei der Ausführung eines Programms erreicht werden, indem verhindert wird, dass alternative Lösungen weiter untersucht werden.
- **Grüner Cut.**



GRÜNER CUT

- Bei einem **grünen Cut** wird die **prozedurale** Bedeutung eines PROLOG-Programms, nicht jedoch die **deklarative** Bedeutung geändert.



```
anfrage:-write( Gib Abfahrtsort: ),nl,  
read(Von),  
dic(Von,- ),  
!,  
write( Es gibt eine Direktverbindung: ),nl.  
anfrage:-write( Es gibt keine Direktverbindung  
) ,nl,  
?-anfrage.
```



- Damit kann man die Behauptung **es gibt mindestens eine Direktverbindung von h_a aus** beantworten.



- Durch den Einsatz des Prädikats **cut** - nachdem die erste Instanzierung der Variablen x mit der Konstanten k_0 vorgenommen wurde - werden keine weiteren Ableitbarkeits-Prüfungen mehr durchgeführt.
- Setzen wir das Prädikat **cut** in dieser Form ein, so gehen keine Lösungen verloren, da wir lediglich an der Existenz mindestens einer Direktverbindung von ha aus interessiert sind.



- Grundsätzlich ist der Einsatz des Prädikats **cut nicht unproblematisch**.
- Es ist stets die jeweils zugrundeliegende Anfrage zu beachten, zu deren Lösung das jeweilige PROLOG-Programm entwickelt wurde.
- Der Einsatz des Prädikats **cut** - in der Wirkung eines **roten Cut** - kann leicht dazu führen, dass mögliche Lösungen durch unterbundenenes seichtes und tiefes Backtracking **nicht abgeleitet** oder aber **falsche Ergebnisse** angezeigt werden.



BEISPIEL 1

$\max(X, Y, Y) : -X =< Y, !.$

$\max(X, Y, X) : -Y =< X.$

?-max(5,10,Maximum).



BEISPIEL 1

- `Maximum = 10;`
`no`
- Dabei wird die Anfrage mit der 1. Klausel erfolgreich abgeleitet, so dass die Variable `Maximum` bzw. `Y` mit dem größeren der beiden Werte instanziiert ist.
- Da die Bedingung $X \leq Y$ erfüllt ist, wird anschließend das Prädikat `cut` abgeleitet und somit Backtracking zur 2. Klausel unterbunden.



BEISPIEL 1

- Entfernen wir den Cut in der 1. Klausel und stellen die gleiche Anfrage, so erhalten wir das gleiche Ergebnis angezeigt.
- Dies liegt daran, dass jetzt - nach der Eingabe eines Semikolons - Backtracking zur 2. Klausel stattfindet.
- Die Ableitung der Anfrage mit der 2. Klausel scheitert jedoch, da die Bedingung $10 = < 5$ nicht erfüllt ist.
- Da wir in beiden Programmversionen das gleiche (erwartete) Ergebnis erhalten, liegt ein **grüner Cut** vor.



BEISPIEL 2

$\max(X, Y, Y) : -X = < Y, !.$

$\max(X, Y, X).$

?- $\max(5, 10, \text{Maximum}).$



BEISPIEL 2

- Entfernen wir den Cut in der 1. Klausel und stellen wiederum die gleiche Anfrage, so wird die Anfrage mit einem falschen Ergebnis in der Form

Maximum = 10;

Maximum = 5;

no

beantwortet.



BEISPIEL 2

- Bei der Ableitbarkeits-Prüfung wird die Anfrage zunächst mit der 1. Klausel und - nach der Eingabe des Semikolons - auch mit der 2. Klausel erfolgreich abgeleitet.
- Somit erhalten wir für die Variable `Maximum` die beiden Werte `10` und `5` angezeigt.
- Da keine weitere Instanzierung der Variablen `Maximum` mehr möglich ist, führt die Eingabe eines weiteren Semikolons zur abschließenden Ausgabe des Textes `no.`
- Da wir in beiden Programmversionen nicht das gleiche Ergebnis, sondern in der Version ohne Einsatz des `Cut` ein unerwartetes und falsches Ergebnis angezeigt bekommen, handelt es sich hier um einen **roten Cut**.



CUT

- Das Standard-Prädikat **cut** ist das einzige Prädikat, das die Arbeit der Inferenzkomponente direkt beeinflusst.
- Die Farbe des Cut läßt sich stets dadurch bestimmen, dass eine Anfrage an das ursprüngliche PROLOG-Programm und anschließend die gleiche Anfrage an das gleiche Programm - ohne den Cut - gestellt wird.
- Führt die Anfrage an das derart geänderte Programm zu unerwarteten oder gar falschen Ergebnissen, so liegt ein **roter Cut** vor.
- Ansonsten handelt es sich um einen **grünen Cut**.



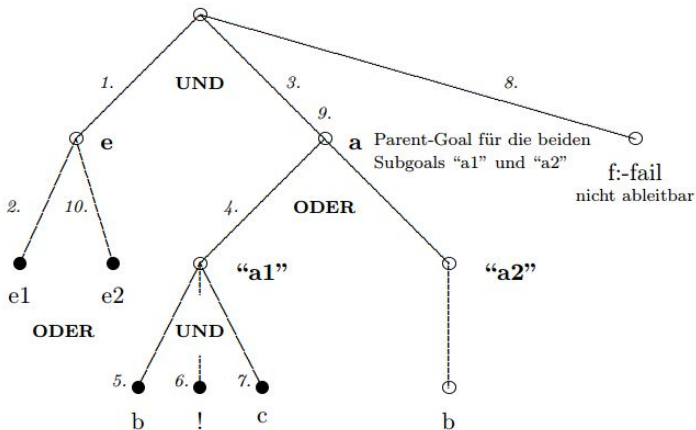
BEISPIEL 3: DEMONSTRATION DES PRÄDIKATS CUT

```
b.  
c.  
e1.  
e2.  
e:-e1.  
e:-e2.  
f:-fail.  
a:-b,!,c.  
a:-b.  
:-e,a,f.
```



BEISPIEL 3: DEMONSTRATION DES PRÄDIKATS CUT

internes Goal: e,a,f.



BEISPIEL 3: DEMONSTRATION DES PRÄDIKATS CUT

- Zur Überprüfung der Ableitbarkeit des internen Goals wird als erstes das Subgoal e abgeleitet, weil das Prädikat e_1 unifizierbar ist.
- Anschließend muss die Unifizierbarkeit des 2. Subgoals a - als Parent-Goal für die beiden zugehörigen Subgoals $b, !, c$ (im Ableitungsbaum ist dieses Parent-Goal durch a_1 gekennzeichnet) und b (im Ableitungsbaum durch a_2 gekennzeichnet) in den Regelrumpfen der beiden Regeln mit dem Regelkopf a - untersucht werden.



BEISPIEL 3: DEMONSTRATION DES PRÄDIKATS CUT

- Dazu wird in der 1. Regel - mit dem Prädikat a im Regelkopf zunächst das Prädikat b und anschließend das Prädikat **cut** unifiziert, so dass die 2. Regel $a : -b .$ für jeden zu einem späteren Zeitpunkt einsetzenden Unifizierungs-Versuch des Parent-Goals a gesperrt ist.
- Da das Prädikat c in der Regel $a : -b , ! , c .$ ebenfalls ableitbar ist, erweist sich der gesamte Regelrumpf und damit der Regelkopf a als ableitbar.
- Anschließend wird das 3. Subgoal f des internen Goals überprüft.
- Wegen der Nicht-Ableitbarkeit des Subgoals f erfolgt Backtracking.



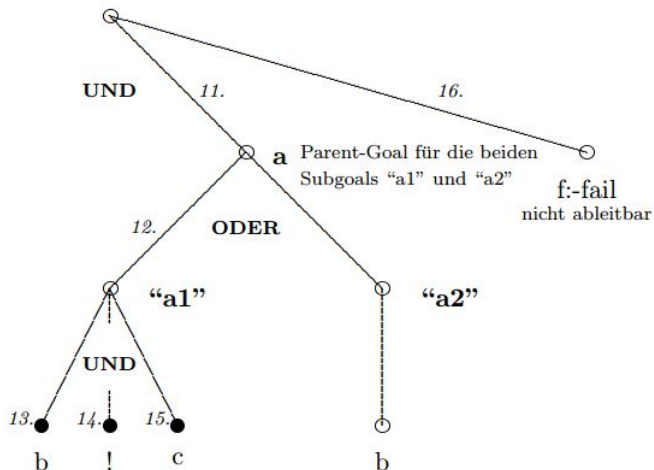
BEISPIEL 3: DEMONSTRATION DES PRÄDIKATS CUT

- Da ein Backtracking zur 2. Regel $a : -b .$ - wegen der vorausgegangenen Unifizierung des Standard-Prädikats `cut` - nicht mehr möglich ist, wird das Parent-Goal `a` als nicht ableitbar erkannt.
- Dadurch wird ein Backtracking im internen Goal zum Prädikat `e` vorgenommen, für das die 1. Regel $e : -e1 .$ als Backtracking-Klausel markiert ist.
- Nach Backtracking erweist sich der Regelrumpf der 2. Regel $e : -e2 .$ als ableitbar.
- Somit wird ein erneuter Unifizierungs-Versuch des 2. Subgoals `a` im internen Goal versucht.
- Da hierbei mit einem neuen Exemplar der Wiba gearbeitet wird, hat folglich die ursprünglich eingetretene Wirkung des Standard-Prädikats `cut`, das nur noch Backtracking hinter dem `cut` zuließ, keine Bedeutung.



BEISPIEL 3: DEMONSTRATION DES PRÄDIKATS CUT

internes Goal: e,a,f.



BEISPIEL 3: DEMONSTRATION DES PRÄDIKATS CUT

- Diese erneute Ableitbarkeits-Prüfung für das 2. Subgoal mit dem Prädikat a wird genauso vorgenommen, wie wir es oben für den 1. Versuch beschrieben haben.
- Folglich ist das Subgoal a (im Schritt 15) wiederum unifizierbar.
- Der nachfolgende Versuch der Unifizierung des 3. Subgoals f schlägt wieder fehl, so dass Backtracking versucht wird.
- Da wiederum - wegen des bereits unifizierten Prädikats cut - kein Backtracking für das 2. Subgoal a durchgeführt werden kann und zusätzlich auch keine Alternative für ein Backtracking zum Ableiten des 1. Subgoals e mehr existiert, wird (im 16. Schritt) endgültig das Scheitern des internen Goals festgestellt.

