

# Cut und Negation

---

Heute:

- der Cut
- if-then-else
- Negation

# Zur Erinnerung: Suchbäume

`p(X) :- a(X).`

`p(X) :- b(X), c(X), d(X).`

`p(X) :- f(X).`

`a(1).`

`b(1).`

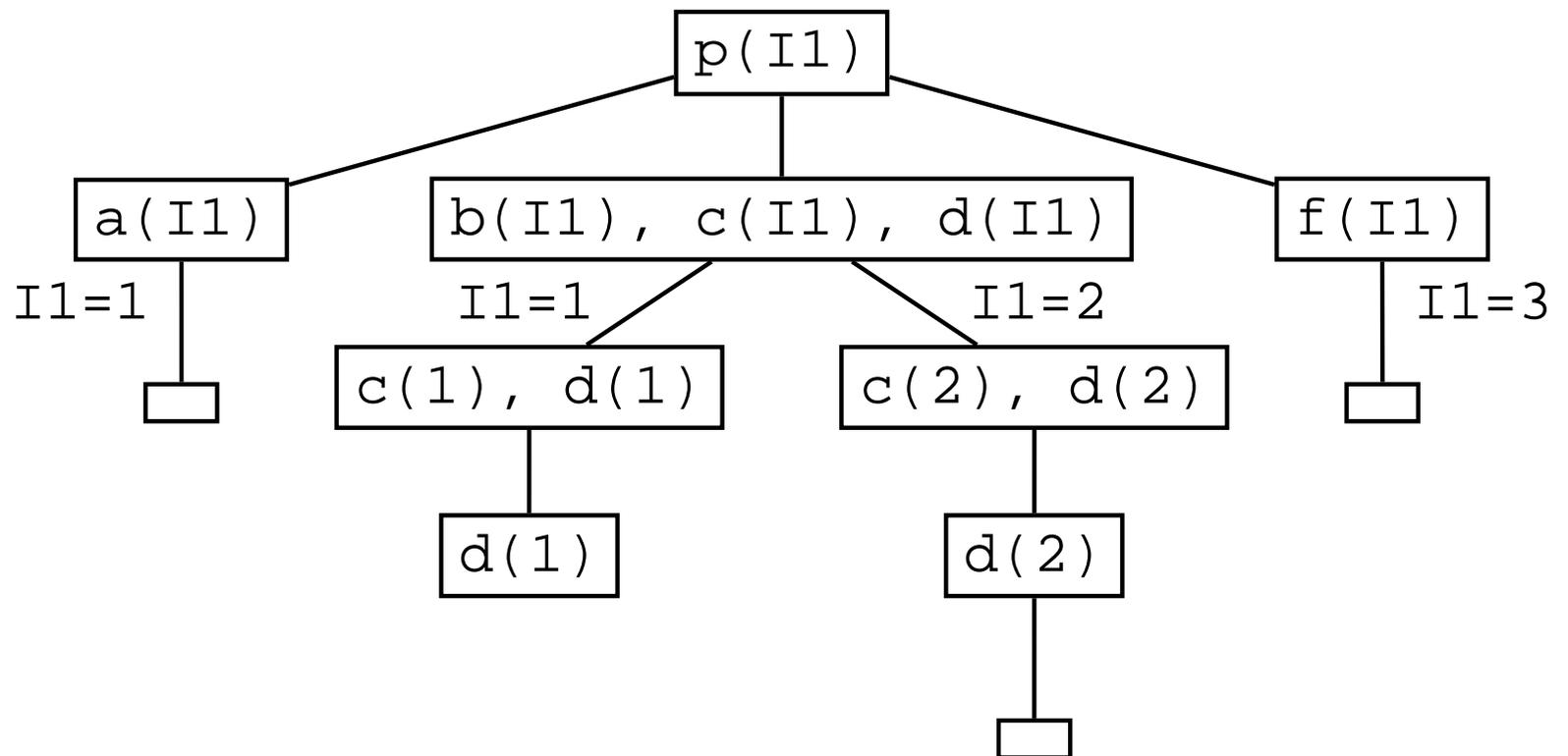
`c(1).`

`b(2).`

`c(2).`

`d(2).`

`f(3).`



# Zu viel Backtracking

```
delete(_, [], []).
```

```
delete(X, [X|T], R) :- delete(X, R, T).
```

```
delete(X, [Y|T], [Y|R]) :- delete(X, R, T).
```

```
?- element(a, [a, b, c], R).
```

```
R = [b, c] ;
```

```
R = [a, b, c] ;
```

```
No
```

Zwei mögliche Lösungen:

1. In die zweite Klausel den Test  $X\bar{Y}$  einbauen.
2. Backtracking verbieten, wenn die zweite Klausel gematcht hat.

## Der Cut: !

Der Cut ist ein eingebautes Prologprädikat, das immer wahr ist, aber einen Seiteneffekt hat: es schränkt Backtracking ein.

$$p(X) \text{ :- } b(X), c(X), !, d(X), e(X).$$

Wenn Prolog bei seiner Beweissuche den Cut (!) erreicht, dann können Entscheidungen links von dem Cut nicht mehr rückgängig gemacht werden. D.h. es kann kein backtracking über die Ziele  $p(X)$ ,  $b(X)$ , und  $c(X)$  mehr geben.

# Beispiel 1

`p(X) :- a(X).`

`p(X) :- b(X), c(X), d(X).`

`p(X) :- f(X).`

`a(1).`

`b(1).`

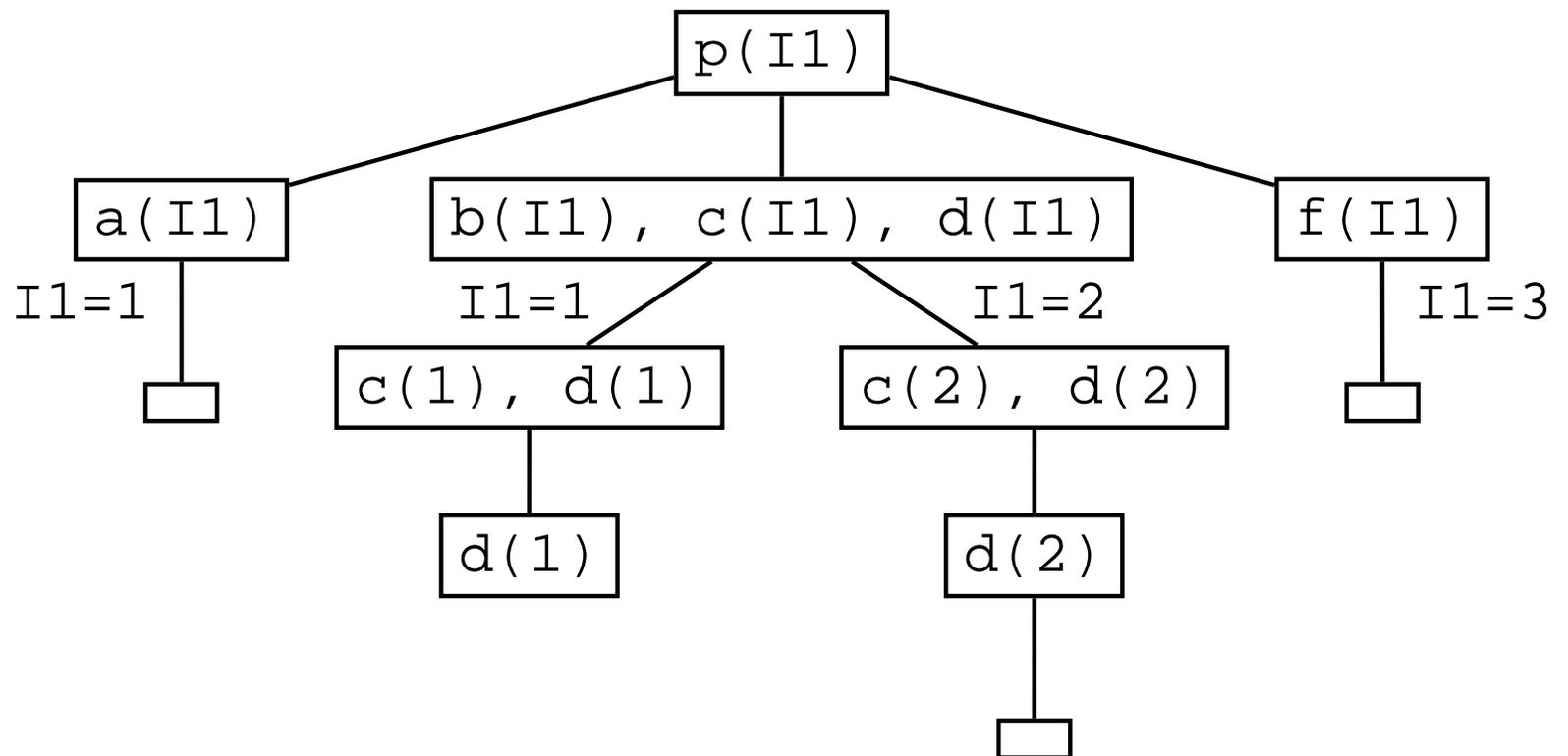
`c(1).`

`b(2).`

`c(2).`

`d(2).`

`f(3).`



# Beispiel 1 (mit Cut)

`p(X) :- a(X).`

`p(X) :- b(X), c(X), !, d(X).`

`p(X) :- f(X).`

`a(1).`

`b(1).`

`c(1).`

`b(2).`

`c(2).`

`d(2).`

`f(3).`

# Beispiel 1 (mit Cut)

`p(X) :- a(X).`

`p(X) :- b(X), c(X), !, d(X).`

`p(X) :- f(X).`

`a(1).`

`b(1).`

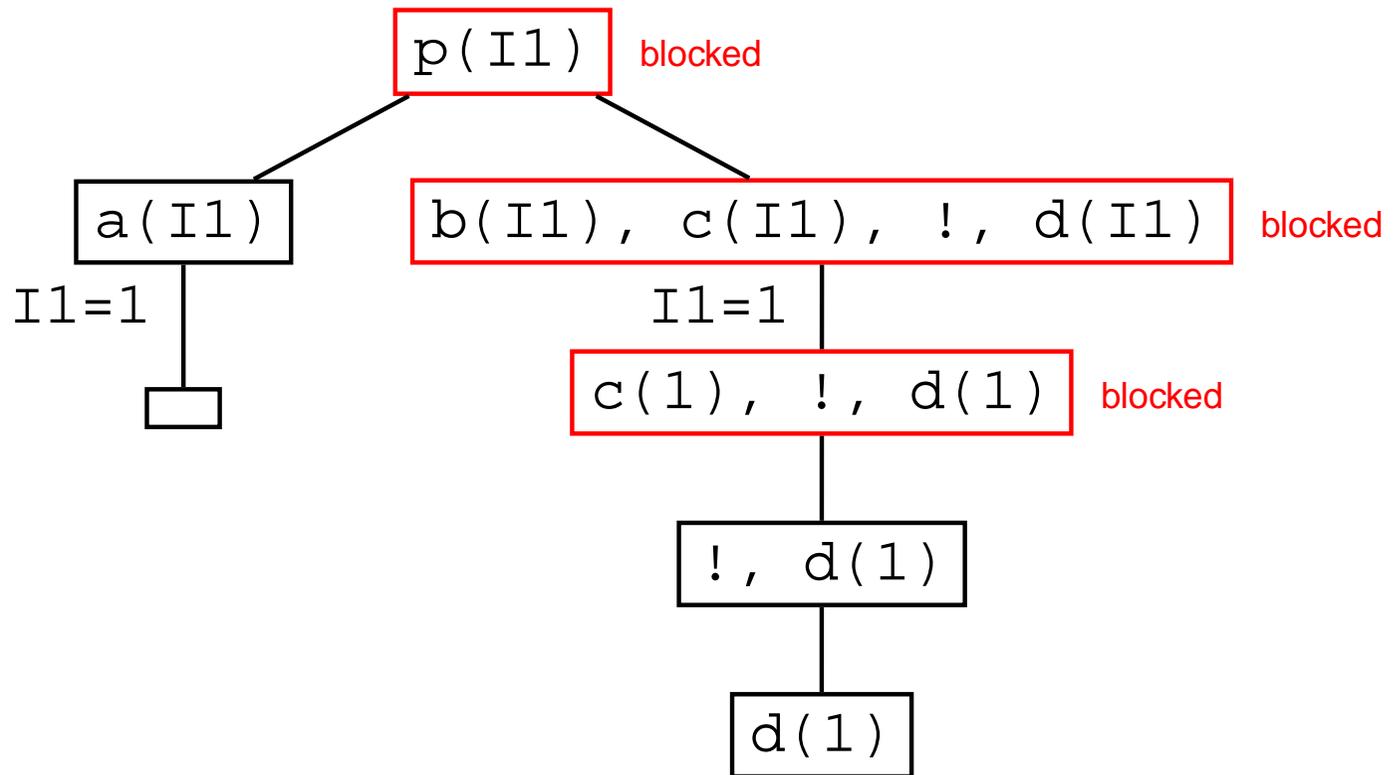
`c(1).`

`b(2).`

`c(2).`

`d(2).`

`f(3).`



# Nochmal: Zu viel Backtracking

---

```
delete(_, [], []).
```

```
delete(X, [X|T], R) :- !, delete(X, R, T).
```

```
delete(X, [Y|T], [Y|R]) :- delete(X, R, T).
```

```
?- element(a, [a, b, c], R).
```

```
R = [b, c] ;
```

```
No
```

# Rote und grüne Cuts

---

Man unterscheidet zwei Arten von Cuts:

**rote Cuts** schneiden mögliche Lösungen weg. D.h. die Antworten, die Prolog auf eine Anfrage an ein Prädikat gibt, können anders sein, als Anfragen an die cut-lose Version des Prädikats.

**grüne Cuts** schneiden nur Äste weg, die in einem 'fail' enden. Grüne Cuts sind also nur dafür da, Programme effizienter oder (für den Programmierer) besser lesbar zu machen.

## Zwei Versionen von `max/3`

`max(Zahl1, Zahl2, Zahl3)`: Zahl3 ist die größere der beiden Zahlen Zahl1 und Zahl2.

mit grünem Cut:

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

`max(X, Y, X) :- X > Y.`

mit rotem Cut:

`max(X, Y, Z) :- X =< Y, !, Z=Y.`

`max(X, _, X).`

# Vorsicht mit roten Cuts

richtig:

```
max(X, Y, Z) :- X =< Y, !, Z=Y.
```

```
max(X, _, X).
```

```
?- max(2, 3, 2).
```

No.

falsch:

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

```
max(X, _, X).
```

```
?- max(2, 3, 2).
```

Yes.

# if-then-else

In Prolog gibt es eine Konstruktion, um Bedingungen der Form “wenn B, dann X, sonst Y” auszudrücken.

```
( A -> B ; C )
```

Wenn A wahr ist dann beweise B, ansonsten beweise C.

# max mit if-then-else

max mit if-then-else ( A -> B ; C ):

```
max(X, Y, Z) :-  
    ( X =< Y  
    -> Z = Y  
    ; Z = X  
    ).
```

max mit (grünem) Cut:

```
max(X, Y, Y) :- X =< Y, !.  
max(X, Y, X) :- X > Y.
```

# fail

Das eingebaute Prädikat `fail/0` schlägt immer fehl.

Beispiel:

```
g(a) :- fail.
```

```
g(b).
```

```
?- g(X).
```

```
X=b ;
```

```
No
```

Wozu ist das gut?

# Negation as Failure

---

Mit Hilfe vom Cut und dem Prädikat fail kann man 'eine Art' Negation bauen.

```
% Falls das Ziel 'Goal' erfüllbar ist, schlage  
% fehl.
```

```
neg(Goal) :- Goal, !, fail.
```

```
% Sonst (falls das Ziel nicht erfüllbar ist)  
% antworte 'yes'.
```

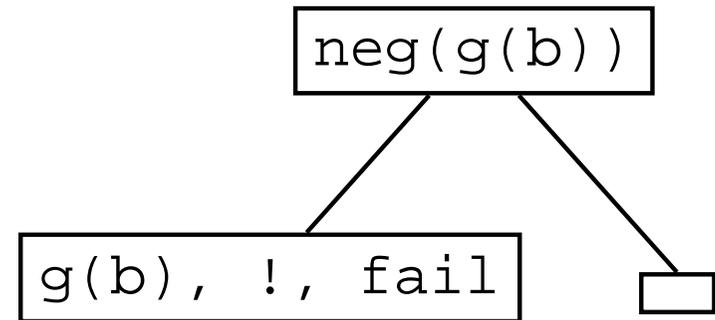
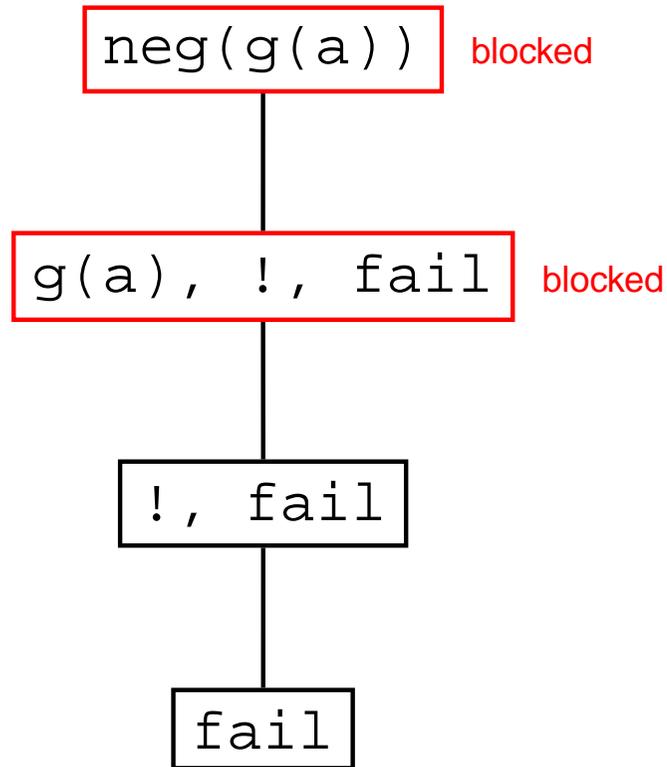
```
neg(_).
```

g(a).	?- g(a).	?- g(b).
	Yes	No
	?- neg(g(a)).	?- neg(g(b)).
	No	Yes

# Negation as Failure: Suchbäume

```
neg(Goal) :- Goal, !, fail.  
neg(_).
```

`g(a).`





Der eingebaute Operator (ein Präfixoperator)  $\backslash+$  ist wie unser `neg` Prädikat definiert. Also können wir z.B.

`?- \+ g(a).`

statt

`?- neg(g(a)).`

schreiben.

# Cut oder Negation?

---

$p :- a, b.$

$p :- a, !, b.$

$p :- \backslash+ a, c.$

$p :- c.$

# Cut oder Negation?

---

$p :- a, b.$

$p :- a, !, b.$

$p :- \backslash+ a, c.$

$p :- c.$

Normalerweise ist Negation roten Cuts vorzuziehen.

Aber im Beispiel ist die Variante mit dem roten Cut vorzuziehen, wenn  $a$  sehr komplex ist.

# Aufgaben

1. `bruder(X,Y) :- sohn(X,V), sohn(Y,V).`  
`sohn(hugo,emil).`  
`sohn(herbert,emil).`  
`sohn(hubert,emil).`

Wie antwortet Prolog auf die Anfrage `bruder(X,Y)`? Und wie würde Prolog auf diese Anfrage antworten, wenn wir die erste Klausel durch die Klausel a (b, c) ersetzen würden?

- (a) `bruder(X,Y) :- !, sohn(X,V), sohn(Y,V).`
- (b) `bruder(X,Y) :- sohn(X,V), !, sohn(Y,V).`
- (c) `bruder(X,Y) :- sohn(X,V), sohn(Y,V), !.`

2. Wie könnte man die Regel *mia mag burger* in Prolog ausdrücken?  
Und wie sähe dann die Regel *vincent mag burger außer big-kahuna-burger* aus?

# Zusammenfassung

---

Heute haben wir gesehen:

- wie man mit Hilfe des Cuts Backtracking verhindern kann,
- was der Unterschied zwischen grünen und roten Cuts ist,
- wie man Cuts benutzen kann, um eine Art von Negation ('Negation as Failure') zu definieren.

Nächste Woche: Manipulation der Wissensbasis.

**Übungsaufgaben:** Die Aufgaben aus Kapitel 10.4 von 'LPN!'. Abgabe ist am nächsten Freitag.