# Negation as Failure

## Allgemeine Logische Programme:

- Bisher: Regeln der Form $A \leftarrow B_1 \wedge \cdots \wedge B_n$.
- Jetzt: $A \leftarrow B_1 \wedge \cdots \wedge B_n \wedge \neg C_1 \wedge \cdots \wedge \neg C_m$.

  Die $B_i$ heißen auch positive, die $\neg C_j$ negative Rumpfliterale. Es sind auch andere Reihenfolgen zulässig (nicht nur: alle $B_i$ vor allen $\neg C_j$).

## Beispiel $\Phi$:

- Gegeben Tabelle mit Bestellungen:

  $\mathrm{bestellung}(\mathrm{meier}, \mathrm{taschenlampe}).$

  $\mathrm{bestellung}(\mathrm{meier}, \mathrm{batterien}).$

  $\mathrm{bestellung}(\mathrm{schmidt}, \mathrm{taschenlampe}).$
- Wer hat die Batterien vergessen?

  $$\begin{aligned} \mathrm{nachfragen}(X) \leftarrow\ & \mathrm{bestellung}(X, \mathrm{taschenlampe}) \\ & \wedge \neg \mathrm{bestellung}(X, \mathrm{batterien}). \end{aligned}$$

## Beachte:

- nachfragen(schmidt) ist keine logische Folgerung aus dem Beispiel $\Phi$, aber es folgt aus comp($\Phi$).
- $\neg$ wird interpretiert als „nicht beweisbar“.
- Die Folgerungen aus $comp(\Phi)$ sind nicht monoton in $\Phi$. Fügt man etwa $\mathrm{bestellung}(\mathrm{schmidt}, \mathrm{batterien})$ ein, so wird obige Folgerung falsch.

# Deklarative Semantik

**Direkte Konsequenzen $T_\Phi$:**

$p(d_1, \ldots, d_k) \in T_\Phi(I)$ gdw. es gibt eine Variablenbelegung $\alpha$ und eine Regel $p(t_1, \ldots, t_k) \leftarrow \psi$ mit

- $(I, \alpha)[\![t_i]\!] = d_i$ (für $i = 1, \ldots, k$),
- $(I, \alpha) \models \psi$.

**Semantik:**

- Clark's Vervollständigung (unverändert).
- Kausales Modell (wie vorher, d.h. $T_\Phi(I) = I$).
- Die beiden Semantiken sind nachwievor äquivalent.

**Probleme:**

- Es gibt allgemeine logische Programme, die kein kausales Modell haben.

  Z.B. $p \leftarrow \neg p$. Lösung: Rekursion durch Negation verbieten.
- Konsequenzen $T_\Phi$ sind nicht mehr monoton.
- Schnitt aller Herbrandmodelle ist nicht immer ein Modell.

  Z.B. $p(a) \leftarrow \neg p(b)$. Lösung: Andere Definition (z.B. stabiles Modell).

# Operationale Semantik: SLDNF-Resolution

## Idee:

- Um negatives Literal $\neg A$ auszuwerten, stelle rekursiv Anfrage $A$. Falls Antwort …
- …„yes“: Beweis von $\neg A$ gescheitert.
- …„no“: $\neg A$ gilt als „bewiesen“.
- Formal: Auswertung von $\neg A$ in SLDNF-Ableitung der Stufe $k$:
- Wenn es für $A$ endlichen SLDNF-Baum der Stufe $k-1$ aus lauter fehlgeschlagenen Ableitungen gibt, so wird $\neg A$ wie ein Faktum behandelt.
- Falls es für $A$ eine erfolgreiche SLDNF-Ableitung der Stufe $k-1$ gibt, so wird $\neg A$ wie Literal ohne anwendbare Regel behandelt.

## Problem:

- Das negative Literal darf keine Variablen enthalten, sonst können sich falsche Antworten ergeben.
- $$\begin{array}{llll} \Phi: & p \leftarrow \neg q(X). & comp(\Phi): & p \leftrightarrow \exists X\, \neg q(X). \\ & q(a). & & q(X) \leftrightarrow X = a. \\ & r(b). & & r(X) \leftrightarrow X = b. \end{array}$$
  Es gilt $comp(\Phi) \not\vdash \neg p$, aber Prolog beantwortet $p$ mit „no“.
- Lösung: Selektionsfunktion darf nur variablenfreie negative Literale auswählen. Falls Anfrage nur noch negative Literale mit Variablen enthält: Abbruch mit Fehlermeldung „query flounders“.

# Korrektheit und Vollständigkeit

## Korrektheit:

- Sei $\Phi$ ein allgemeines logisches Programm und $\theta$ eine mittels SLDNF-Resolution berechnete Antwort auf $\psi$.
- Sind keine negativen Literale mit Variablen ausgewählt worden, so gilt $comp(\Phi) \vdash \forall(\psi\theta)$.

## Vollständigkeit:

- Die Vollständigkeit gilt nur recht eingeschränkt.
- Z.B. ist die SLDNF-Resolution für nichtrekursive (hierarchische) Programme vollständig.

  Natürlich muß man auch das Floundering verhindern, z.B. indem man fordert, daß jede in einer Regel auftretende Variable auch in einem positiven Rumpfliteral vorkommt.
- Ein typisches Beispiel, bei dem SLDNF nicht vollständig ist, ist:

  $\Phi$ : $p \leftarrow q.$

  $p \leftarrow \neg q.$

  $q \leftarrow q.$

  Es gilt $\Phi \vdash p$, aber die Antwort „yes“ auf $p$ kann nicht mittels SLDNF berechnet werden (SLDNF kennt keine Fallunterscheidungen).
- SLDNF (mit fairer Selektionsfunktion) ist vollständig für strikte Programme, bei denen kein Prädikat $p$ ein Prädikat $q$ sowohl über eine gerade als auch über eine ungerade Anzahl von Negationen aufruft (plus obige Bedingung zur Vermeidung des Floundering).