# Kapitel L:V

## V. Erweiterungen und Anwendungen zur Logik

- Produktionsregelsysteme
- Inferenz für Produktionsregelsysteme
- Produktionsregelsysteme mit Negation
- Regeln mit Konfidenzen
- Nicht-monotones Schließen
- Logik und abstrakte Algebren
- Verifikation
- Verifikation mit dem Hoare-Kalkül
- Hoare-Regeln und partielle Korrektheit
- Terminierung

# Produktionsregelsysteme

## Vergleich von Deduktions- und Produktionsregelsystem

Deduktionssysteme:

- Verwendung von Resolution bzw. einem anderen vollständigen Inferenzverfahren, um Sätze in Prädikatenlogik erster Stufe (PL1) zu beweisen.
- Beantwortung einer Anfrage geschieht durch Variableninstantiierung beim Beweis eines Satzes.
- Formeln müssen keine spezielle Form haben.

# Produktionsregelsysteme

## Vergleich von Deduktions- und Produktionsregelsystem

Produktionsregelsysteme:

- Zentrale Repräsentationsform ist die Implikation (Regelform) – d. h. die Formeln in der Datenbasis haben eine spezielle Form.
- Die rechte Seite einer Regel wird als Aktion interpretiert. Typische Aktionen sind die Hinzunahme und das Löschen von Fakten in einer Datenbasis sowie Ein- und Ausgabeoperationen.
- Wichtigster Schlussfolgerungsmechanismus ist die Vorwärtsverkettung. Stichwort: *Produktion*.
- Die Semantik der Regeln ist am Anwendungsbereich (Domäne) orientiert.
- Es gibt einen Konfliktauflösungsmechanismus, falls mehrere Aktionen zur Auswahl stehen.

# Produktionsregelsysteme

## Definition 1 (Produktionsregelsystem)

Sei $\Sigma_P$ eine endliche Menge von Atomen, gebildet aus einer endlichen Menge von Objekten $O_P$, den Vergleichsoperatoren $\{=, \neq\}$ und einer endlichen Menge von Werten $V_P$.

1. Ein Atom in $\Sigma_P$ hat die Form $o = v$ bzw. $o \neq v$ und wird interpretiert als „$o$ ist gleich $v$“ bzw. „$o$ ist ungleich $v$“.

2. $P = (D, R)$ ist ein Produktionsregelsystem; $D \subseteq \Sigma_P$ definiert die Datenbasis, $R$ definiert eine endliche Menge von Regeln.
   Die Atome in $D$ werden als Fakten bezeichnet.

3. Eine Regel $r \in R$ hat die Form „$\texttt{IF}\ \alpha\ \texttt{THEN}\ \kappa$“. $\alpha$ ist eine Formel, zusammengesetzt aus Atomen aus $\Sigma_P$ und den Junktoren $\wedge$ und $\vee$. $\kappa$ ist ein Atom aus $\Sigma_P$.
   $\alpha$ wird als Bedingung oder Prämisse und $\kappa$ als Konklusion der Regel bezeichnet.

Bemerkungen:

- Eine Konklusion als Konjunktion mehrerer Atome ist nicht zugelassen – jedoch

$$\texttt{IF } \alpha \texttt{ THEN } \kappa_1 \wedge \kappa_2 \qquad\qquad \begin{array}{l}\texttt{IF } \alpha \texttt{ THEN } \kappa_1 \\ \texttt{IF } \alpha \texttt{ THEN } \kappa_2\end{array}$$

- Die Negation in der Regel ist nicht zugelassen.
- Anstatt Objekt-Wert-Tupeln als Atome sind auch Objekt-Attribut-Wert-Tripel (OAW-Tripel) denkbar und üblich. Beispiel: EMYCIN.
- Produktionregeln können einfach um ein Konfidenzfaktorkonzept erweitert werden, das die Sicherheit von Konklusionen bewertet oder miteinander verrechnet.

# Produktionsregelsysteme

**Definition 2 (Semantik Produktionsregelsystem)**

Eine Bedingung $\alpha$ ist genau dann erfüllt (wahr) bzgl. einer Datenbasis $D$, wenn gilt:

1. $\alpha$ ist ein Atom und es gilt $\alpha \in D$.
2. $\alpha$ hat die Form $\alpha_1 \wedge \alpha_2$ und es gilt $\alpha_1$ ist wahr und $\alpha_2$ ist wahr bzgl. einer Datenbasis $D$.
3. $\alpha$ hat die Form $\alpha_1 \vee \alpha_2$ und es gilt $\alpha_1$ ist wahr oder $\alpha_2$ ist wahr bzgl. einer Datenbasis $D$.
4. Nur Bedingungen, die gemäß 1 bis 3 wahr sind, sind wahr.

Eine Regel `IF` $\alpha$ `THEN` $\kappa$, deren Bedingung $\alpha$ wahr ist bzgl. einer Datenbasis $D$, heißt anwendbar für $D$.

# Produktionsregelsysteme

## Definition 3 (Ableitung)

Seien $P = (D, R)$ und $P' = (D', R)$ zwei Produktionsregelsysteme.

Dann gilt: „$P'$ ist in einem Schritt aus $P$ herleitbar“, in Zeichen: $(D, R) \mid\!\frac{1}{PS}\, (D', R)$, genau dann, wenn eine Regel $r \in R$ existiert, $r = \texttt{IF}\ \alpha\ \texttt{THEN}\ \kappa$ mit

1. $\alpha$ ist wahr bzgl. $D$, d.h. $r$ ist anwendbar für $D$

2. $D' = D \cup \{\kappa\}$

Abkürzend: $(D, R) \mid\!\frac{1}{PS}\, \kappa$

$(D, R) \mid\!\frac{}{PS}\, (D', R)$ bezeichnet die reflexive und transitive Hülle der Einschritt-Ableitung $(D, R) \mid\!\frac{1}{PS}\, (D', R)$.

Bemerkungen:

- Eine Regel kann genau dann angewendet (gefeuert) werden, wenn die Bedingung $\alpha$ bzgl. der aktuellen Datenbasis erfüllt ist.
- Offensichtlich stellt die Konklusion einer gefeuerten Regel eine Folgerung dar.
- Die Wirkung einer Regelanwendung ist, dass die Konklusion $\kappa$ der Regel mit in die Datenbasis aufgenommen wird.
- Der transitive Abschluss entspricht der Verkettung im Sinne der Hintereinanderausführung von Regeln.
- Ein Regelsystem ist prozedural: Implizit enthält das Feuern einer Regel eine Aktion: *„Füge Fakt zur Datenbasis hinzu“*.
- Herleitungen in einem Produktionsregelsystem sind nicht deterministisch.

# Produktionsregelsysteme

## Regelinterpreter

Die Definition der Ableitung in Produktionsregelsystemen beschreibt den Kern eines *Regel-Interpreters*, den sogenannten Recognize-Act-Zyklus:

1. Bestimmung der Konfliktmenge der anwendbaren Regeln.
2. Auswahl einer Regel aus der Konfliktmenge durch ein Selektionsverfahren.
3. Feuern der Regel.

Insbesondere legt der Interpreter fest, wie die Konfliktmenge gebildet werden kann und aus welchen Kriterien das Selektionsverfahren aufgebaut ist.

# Produktionsregelsysteme

## Regelinterpreter

Die Definition der Ableitung in Produktionsregelsystemen beschreibt den Kern eines *Regel-Interpreters*, den sogenannten Recognize-Act-Zyklus:

1. Bestimmung der Konfliktmenge der anwendbaren Regeln.
2. Auswahl einer Regel aus der Konfliktmenge durch ein Selektionsverfahren.
3. Feuern der Regel.

Insbesondere legt der Interpreter fest, wie die Konfliktmenge gebildet werden kann und aus welchen Kriterien das Selektionsverfahren aufgebaut ist.

Zwei Möglichkeiten, um einen Finalzustand zu ereichen:

1. Alle herleitbaren Fakten sind abgeleitet; keine Regel mehr anwendbar.
   Paradigma: „Finde soviel wie möglich heraus.“
2. Ein gesuchter Fakt ist zu $D$ hinzugefügt worden.
   Paradigma: „Ist ein bestimmtes Ziel folgerbar?“

# Produktionsregelsysteme

## Definition 4 (kommutativ)

Ein Produktionsregelsystem $P = (D, R)$ heißt kommutativ, falls für jede Datenbasis $D_i$, die aus $P$ ableitbar ist, gilt:

Eine für $D_i$ anwendbare Regel ist auch für jede Datenbasis $D_i'$ anwendbar, für die $(D_i, R) \mathrel{\underset{PS}{\vdash}} (D_i', R)$ gilt.

# Produktionsregelsysteme

## Definition 4 (kommutativ)

Ein Produktionsregelsystem $P = (D, R)$ heißt kommutativ, falls für jede Datenbasis $D_i$, die aus $P$ ableitbar ist, gilt:

Eine für $D_i$ anwendbare Regel ist auch für jede Datenbasis $D_i'$ anwendbar, für die $(D_i, R) \mid\!\frac{\;\;}{PS}\, (D_i', R)$ gilt.

## Lemma 5

Für ein kommutatives Produktionsregelsystem $P = (D, R)$ und zwei Datenbasen $D_1, D_2$, die aus $P$ ableitbar sind, gilt die folgende Eigenschaft:

Sei $D_1'$ eine Datenbasis, die aus $D_1$ ableitbar ist, so existiert eine Folge von Regelanwendungen, um $D_1' \cup D_2$ aus $D_2$ abzuleiten. Die Generierung der Fakten in $D_1'$ ist also unabhängig von der Anwendungsreihenfolge der anwendbaren Regeln.

## Satz 6

Produktionsregelsysteme ohne Negation sind kommutativ.

# Produktionsregelsysteme

## Realisierung des Interpreters durch Regelverkettung

$$\underline{D_0,\ \texttt{IF}\ \alpha_1\ \texttt{THEN}\ \kappa_1}\quad \text{(und } \alpha_1 \text{ wahr bzgl. } D_0\text{)}$$

$$\underline{D_1 = D_0 \cup \{\kappa_1\},\ \texttt{IF}\ \alpha_2\ \texttt{THEN}\ \kappa_2}\quad \text{(und } \alpha_2 \text{ wahr bzgl. } D_1\text{)}$$

$$\underline{D_2 = D_1 \cup \{\kappa_2\},\ \texttt{IF}\ \alpha_3\ \texttt{THEN}\ \kappa_3}\quad \text{(und } \alpha_3 \text{ wahr bzgl. } D_2\text{)}$$

$$\underline{D_3 = D_2 \cup \{\kappa_3\},\ \ldots}$$

$$\ddots$$

Die Kommutativität wird hier insofern ausgenutzt, als dass die Reihenfolge der Regelanwendungen keinen Einfluss auf die Menge der abgeleiteten Fakten hat.

# Produktionsregelsysteme

## Realisierung des Interpreters durch Regelverkettung

$$\cfrac{D_0,\ \texttt{IF}\ \alpha_1\ \texttt{THEN}\ \kappa_1}{\cfrac{D_1 = D_0 \cup \{\kappa_1\},\ \texttt{IF}\ \alpha_2\ \texttt{THEN}\ \kappa_2}{\cfrac{D_2 = D_1 \cup \{\kappa_2\},\ \texttt{IF}\ \alpha_3\ \texttt{THEN}\ \kappa_3}{\cfrac{D_3 = D_2 \cup \{\kappa_3\},\ \ldots}{\ddots}}}}$$

(und $\alpha_1$ wahr bzgl. $D_0$)
(und $\alpha_2$ wahr bzgl. $D_1$)
(und $\alpha_3$ wahr bzgl. $D_2$)

Die Kommutativität wird hier insofern ausgenutzt, als dass die Reihenfolge der Regelanwendungen keinen Einfluss auf die Menge der abgeleiteten Fakten hat.

Vergleiche Modus Ponens in der Logik ($\alpha, \beta, \gamma, \delta, \ldots$ beliebige logische Formeln) :

$$\cfrac{\alpha, \quad \alpha \to \beta}{\cfrac{\beta, \quad \beta \to \gamma}{\cfrac{\gamma, \quad \gamma \to \delta}{\cfrac{\delta, \quad \ldots}{\ddots}}}}$$

# Inferenz für Produktionsregelsysteme

## Vorwärtsverkettende Verfahren (Forward-Chaining)

Ausgehend von $D_0$ wird versucht, einen gegebenen Fakt $\kappa_i$ bzw. die Menge aller ableitbaren Fakten zu abzuleiten. Stichwort: *datengetriebene Suche*

$$\underline{D_0,\ \texttt{IF}\ \alpha_1\ \texttt{THEN}\ \kappa_1}$$

$$\underline{D_1 = D_0 \cup \{\kappa_1\},\ \texttt{IF}\ \alpha_2\ \texttt{THEN}\ \kappa_2 \ldots}$$

$$\underline{D_2 = D_1 \cup \{\kappa_2\},\ \ldots}$$

$$\ddots$$

# Inferenz für Produktionsregelsysteme

## Vorwärtsverkettende Verfahren (Forward-Chaining)

Ausgehend von $D_0$ wird versucht, einen gegebenen Fakt $\kappa_i$ bzw. die Menge aller ableitbaren Fakten zu abzuleiten. Stichwort: *datengetriebene Suche*

$$\underline{D_0,\ \texttt{IF}\ \alpha_1\ \texttt{THEN}\ \kappa_1}$$

$$\underline{D_1 = D_0 \cup \{\kappa_1\},\ \texttt{IF}\ \alpha_2\ \texttt{THEN}\ \kappa_2 \ldots}$$

$$\underline{D_2 = D_1 \cup \{\kappa_2\},\ \ldots}$$

$$\ddots$$

## Rückwärtsverkettende Verfahren (Backward-Chaining)

Ausgehend von einem zu bestimmenden Fakt $\kappa_i$ wird versucht, diesen über Regeln auf eine Teilmenge der vorhandenen Startdatenbasis $D_0$ zurückzuführen. Stichwort: *zielgetriebene Suche*

$$\underline{D_0,\ \texttt{IF}\ \alpha_1\ \texttt{THEN}\ \kappa_1}$$

$$\underline{D_1 = D_0 \cup \{\kappa_1\},\ \texttt{IF}\ \alpha_2\ \texttt{THEN}\ \kappa_2 \ldots}$$

$$\underline{D_2 = D_1 \cup \{\kappa_2\},\ \ldots}$$

$$\ddots$$

# Inferenz für Produktionsregelsysteme

## Recognize-Act-Zyklus realisiert als Forward-Chaining

1. Recognize: Konstruktion der Konfliktmenge $R^*$
   Bestimmung aller Regeln, deren Bedingung wahr ist.

2. Act: Feuern einer Regel
   Auswahl einer Regel aus $R^*$ und Ausführung ihrer Konklusion.

Algorithm: FC
Input: Startdatenbasis $D$, Regelmenge $R$
Output: Menge aller aus $D$ mit $R$ ableitbaren Fakten $D^*$

BEGIN
  $D^* = D$

  **REPEAT**
    $D_{\text{tmp}} = D^*$
    $R^* = \{(\texttt{IF}\ \alpha\ \texttt{THEN}\ \kappa) \in R \mid \alpha \text{ wahr bzgl. } D^*\}$
    $D^* = D^* \cup \{\kappa \mid (\texttt{IF}\ \alpha\ \texttt{THEN}\ \kappa) \in R^*\}$
  **UNTIL** $D^* = D_{\text{tmp}}$

  RETURN ($D^*$)
END

# Inferenz für Produktionsregelsysteme

## Eigenschaften des Algorithmus FC

- FC terminiert bei jeder Eingabe.

  Die Größe von $D^*$ ist beschränkt durch die endliche Menge der möglichen Atome in $P$.

- FC bestimmt genau die Menge aller ableitbaren Fakten.

  Beweis über die Kommutativität von $P$.

- FC benötigt höchstens quadratische Zeit in der Größe von $P = (D, R)$.

  Lineare Zeit für jeden Schleifendurchlauf (falls Test, ob ein Fakt für $D^*$ wahr ist, sowie das Hinzufügen von Fakten in einem Schritt möglich sind); die Größe von $D^*$ bestimmt die Anzahl der Schleifendurchläufe.

# Inferenz für Produktionsregelsysteme

## Ableitbarkeitstest durch Forward-Chaining

Algorithm: FC-test. Überprüft, ob ein Atom $\kappa^*$ ableitbar ist.

Input: Startdatenbasis $D$, Regelmenge $R$, Atom $\kappa^*$

Output: *true*, falls $(D, R) \mid_{\overline{PS}} \kappa^*$, *false* sonst

```
BEGIN
  D* = D

  REPEAT
    D_tmp = D*
    R* = {(IF α THEN κ) ∈ R | α wahr bzgl. D*}
    D* = D* ∪ {κ | (IF α THEN κ) ∈ R*}
  UNTIL D* = D_tmp OR κ* ∈ D*

  IF κ* ∈ D*
  THEN RETURN (true)
  ELSE RETURN (false)
END
```

Bemerkungen:

- Eine Verbesserung der Effzienz ist dadruch möglich, dass Regeln, die gefeuert haben, aus der Konfliktmenge entfernt werden.
  - Warum bleibt Korrektheit?
  - Wie verhält sich die Laufzeit?

# Inferenz für Produktionsregelsysteme

## Recognize-Act-Zyklus realisiert als Backward-Chaining

1. Recognize: Konstruktion der Konfliktmenge $R^*$

   Bestimmung aller Regeln, die das zu prüfende Atom als Konklusion haben, bzw. Prüfung, ob das Atom in der Startdatenbasis enthalten ist.

2. Act: Feuern einer Regel

   Auswahl einer bestimmten Regel aus $R^*$ und Generierung neuer Ziele aus der Bedingung $\alpha$ dieser Regel.

Situationen mit Nichtdeterminismen:

- Eventuell existieren mehrere Regeln mit der Konklusion $\kappa$.
- Die Bedingung kann zusammengesetzt sein – dann ist die Reihenfolge der Bearbeitung entscheidend:

  Konjunktion: Welche Teilformel ist nicht ableitbar?
  Disjunktion: Welche Teilformel ist (schnell) ableitbar?

# Inferenz für Produktionsregelsysteme

## Definition 7 (Und-Oder-Baum)

Zu einem Produktionsregelsystem $P = (D, R)$ und einem Ziel $G$ wird ein Und-Oder-Baum $AOT_P(G)$ durch folgende Konstruktionsvorschrift induktiv definiert:

1. Die Wurzel von $AOT_P(G)$ erhält den Label $G$.
2. Ist der Label eines Knotens ein Atom, so erhält der Knoten einen Nachfolger
   - mit Label $\Box$, falls $\kappa \in D$,
   - mit Label $\alpha$ für jede Regel `IF` $\alpha$ `THEN` $\kappa$ in $R$.

   Die Kanten zu den Nachfolgern sind vom Typ ODER.
3. Hat ein Knoten einen Label mit der Struktur $\alpha_1 \land \ldots \land \alpha_n$, so erhält der Knoten $n$ Nachfolger mit den Labeln $\alpha_1$ bis $\alpha_n$.
   Die Kanten zu den Nachfolgern sind vom Typ UND.
4. Hat ein Knoten einen Label mit der Struktur $\alpha_1 \lor \ldots \lor \alpha_n$, so erhält der Knoten $n$ Nachfolger mit den Labeln $\alpha_1$ bis $\alpha_n$.
   Die Kanten zu den Nachfolgern sind vom Typ ODER.
5. $AOT_P(G)$ enthält keine anderen Knoten und Kanten.

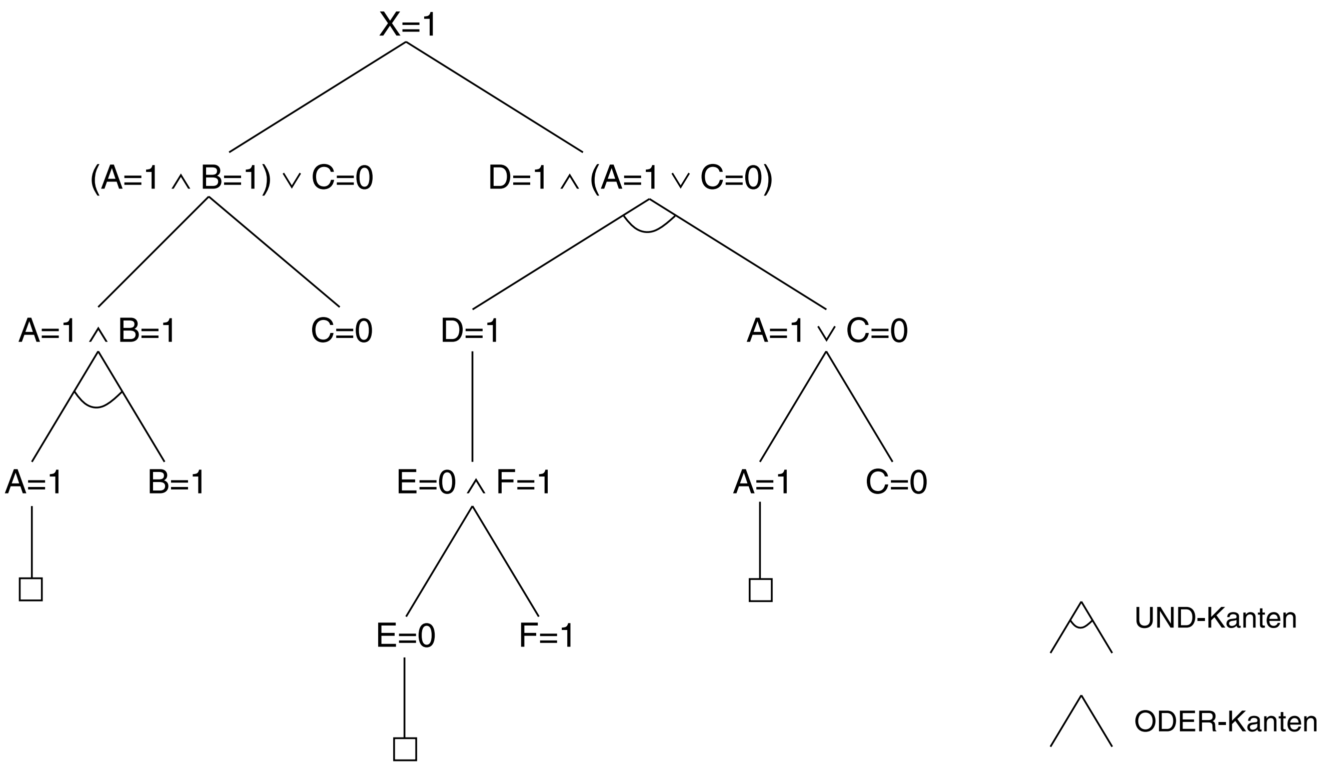
# Inferenz für Produktionsregelsysteme

## Beispiel: Und-Oder-Baum

Gegeben sei folgendes Produktionsregelsystem $P = (D, R)$:

$$D = \{A = 1, E = 0\} \qquad R = \{ \begin{array}{l} r_1 : \texttt{IF } (A = 1 \wedge B = 1) \vee C = 0 \texttt{ THEN } X = 1, \\ r_2 : \texttt{IF } D = 1 \wedge (A = 1 \vee C = 0) \texttt{ THEN } X = 1, \\ r_3 : \texttt{IF } E = 0 \vee F = 1 \texttt{ THEN } D = 1\} \end{array}$$

Für Ziel $X = 1$:



# Inferenz für Produktionsregelsysteme

## Algorithmus für das Backward-Chaining

Algorithm: BC-DFS

Input: Startdatenbasis $D$, Regelmenge $R$, Formel $\alpha$

Output: *true*, falls $\alpha$ ableitbar, *false* sonst; evtl. Endlosschleife

BEGIN
  IF $\alpha = \alpha_1 \wedge \alpha_2$ THEN RETURN (*BC-DFS*($\alpha_1$) AND *BC-DFS*($\alpha_2$)) ENDIF
  IF $\alpha = \alpha_1 \vee \alpha_2$ THEN RETURN (*BC-DFS*($\alpha_1$) OR *BC-DFS*($\alpha_2$)) ENDIF
  IF $\alpha \in D$ THEN RETURN (*true*) ENDIF

# Backward-Chaining

## Algorithmus für das Backward-Chaining

Algorithm: BC-DFS

Input: Startdatenbasis $D$, Regelmenge $R$, Formel $\alpha$

Output: *true*, falls $\alpha$ ableitbar, *false* sonst; evtl. Endlosschleife

```
BEGIN
  IF α = α1 ∧ α2 THEN RETURN (BC-DFS(α1) AND BC-DFS(α2)) ENDIF
  IF α = α1 ∨ α2 THEN RETURN (BC-DFS(α1) OR BC-DFS(α2)) ENDIF
  IF α ∈ D THEN RETURN (true) ENDIF

  R* = {r | r = (IF γ THEN α) und r ∈ R}
  stop=false

  WHILE R* ≠ ∅ AND stop=false DO
      r = choose(R*)
      IF BC-DFS(premise(r))=true
      THEN stop=true
      ELSE R* = R* \ {r}
  END

  IF stop=true
  THEN RETURN (true)
  ELSE RETURN (false)

END
```

# Inferenz für Produktionsregelsysteme

## Bedingungen ohne Disjunktion

Im UND-ODER-Baum wird nicht zwischen alternativen Regeln und Disjunktionen unterschieden.

- $\Rightarrow$ Die ausschließliche Verwendung von Konjunktionen ist ohne Einschränkung hinsichtlich der Ausdrucksstärke.
- $\Rightarrow$ Konstruktion eines Ableitungsbaums, der nur Konjunktionen enthält.
- $\Rightarrow$ Aufspaltung von Regeln mit Disjunktion (Fortsetzung Beispiel):

$$r_1 : \texttt{IF } (A = 1 \wedge B = 1) \vee C = 0 \texttt{ THEN } X = 1,$$
$$r_2 : \texttt{IF } D = 1 \wedge (A = 1 \vee C = 0) \texttt{ THEN } X = 1,$$
$$r_3 : \texttt{IF } E = 0 \vee F = 1 \texttt{ THEN } D = 1$$

Aus $r_1$ wird:

$$r_{1.1} : \texttt{IF } A = 1 \wedge B = 1 \texttt{ THEN } X = 1$$
$$r_{1.2} : \texttt{IF } C = 0 \texttt{ THEN } X = 1$$

Aus $r_3$ wird:

$$r_{3.1} : \texttt{IF } E = 0 \texttt{ THEN } D = 1$$
$$r_{3.2} : \texttt{IF } F = 1 \texttt{ THEN } D = 1$$

# Inferenz für Produktionsregelsysteme

## Bedingungen ohne Disjunktion (Fortsetzung)

Zwei Möglichkeiten, um $r_2$ umzuformen:

1. Einführung von Atomen *aux* $= 1$, die bisher nicht in $P$ nicht existieren.

$$\begin{aligned} r_{2.1} &: \texttt{IF } D = 1 \wedge \textit{aux} = 1 \texttt{ THEN } X = 1 \\ r_{2.2} &: \texttt{IF } A = 1 \texttt{ THEN } \textit{aux} = 1, \\ r_{2.3} &: \texttt{IF } C = 0 \texttt{ THEN } \textit{aux} = 1 \end{aligned}$$

2. Erzeugung der disjunktiven Normalform durch iterative Anwendung der Distributivgesetze:

$$\begin{aligned} t \wedge (t_1 \vee \ldots \vee t_n) &\approx (t \wedge t_1) \vee \ldots \vee (t \wedge t_n) \\ t \vee (t_1 \wedge \ldots \wedge t_n) &\approx (t \vee t_1) \wedge \ldots \wedge (t \vee t_n) \end{aligned}$$

$\rightsquigarrow$

$$\begin{aligned} r_{2.1} &: \texttt{IF } D = 1 \wedge A = 1 \texttt{ THEN } X = 1 \\ r_{2.2} &: \texttt{IF } D = 1 \wedge C = 0 \texttt{ THEN } X = 1 \end{aligned}$$

# Inferenz für Produktionsregelsysteme

**Definition 8 (Ableitungsbaum)**

Zu einem Produktionsregelsystem $P = (D, R)$ und einem Ziel $G$ wird ein Ableitungsbaum $T_P(G)$ durch folgende Konstruktionsvorschrift induktiv definiert:

1. Die Wurzel von $T_P(G)$ erhält den Label $G$.
2. Jeder Knoten mit Label $\alpha = \alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n$ erhält einen Nachfolger
   - mit Label $\alpha_2 \wedge \ldots \wedge \alpha_n$, falls $\alpha_1 \in D$ bzw.
   - mit Label $\Box$, falls $\alpha_1 \in D$ und $n = 1$
3. Jeder Knoten mit Label $\alpha = \alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n$ erhält einen Nachfolger für jede Regel „$\texttt{IF}\ \beta_1 \wedge \ldots \wedge \beta_r\ \texttt{THEN}\ \alpha_1$“ $\in R$ mit dem Label $\beta_1 \wedge \ldots \wedge \beta_r \wedge \alpha_2 \wedge \ldots \wedge \alpha_n$.
4. $T_P(G)$ enthält keine anderen Knoten und Kanten.

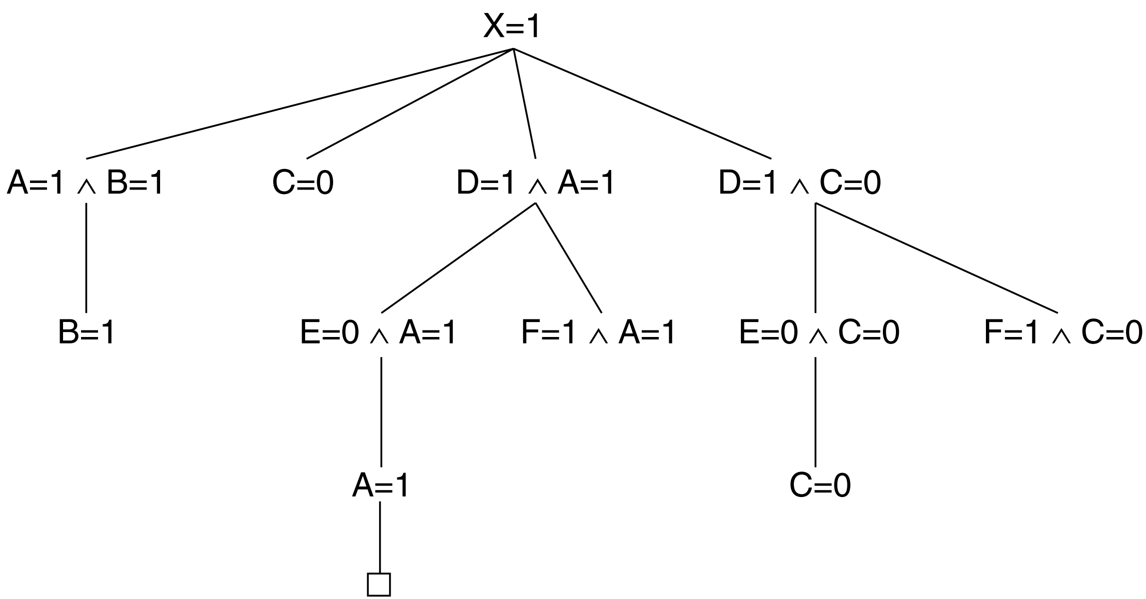
# Inferenz für Produktionsregelsysteme

## Beispiel: Ableitungsbaum für Backward-Chaining

Gegeben sei folgendes Produktionsregelsystem $P = (D, R)$:

$$D = \{A = 1, E = 0\} \qquad R = \{ \begin{array}{l} r_{1.1} : \texttt{IF } A = 1 \wedge B = 1 \texttt{ THEN } X = 1 \\ r_{1.2} : \texttt{IF } C = 0 \texttt{ THEN } X = 1 \\ r_{2.1} : \texttt{IF } D = 1 \wedge A = 1 \texttt{ THEN } X = 1 \\ r_{2.2} : \texttt{IF } D = 1 \wedge C = 0 \texttt{ THEN } X = 1 \\ r_{3.1} : \texttt{IF } E = 0 \texttt{ THEN } D = 1 \\ r_{3.2} : \texttt{IF } F = 1 \texttt{ THEN } D = 1 \end{array}$$

Für Ziel $X = 1$:



## Definition 9 (Regelgraph)

Sei $R$ eine Regelmenge ohne Disjunktionen. Ein Regelgraph $G_R = \langle V, E \rangle$ ist ein gerichteter Graph, der wie folgt definiert ist:

1. Für jedes in $R$ vorkommende Atom $\kappa$ existiert ein Knoten $v_\kappa$ in $V$.
2. Für jede Regel $r \in R$ existiert ein Knoten $v_r$ in $V$.
3. Für jede Regel $r = \texttt{IF}\ \alpha_1 \wedge \ldots \wedge \alpha_n\ \texttt{THEN}\ \kappa$ existieren $n$ Kanten von $v_{\alpha_i}$ nach $v_r$ $(i = 1, \ldots, n)$ und eine Kante von $v_r$ nach $v_\kappa$ in $E$.
4. $G_R$ enthält keine anderen Knoten und Kanten.

# Backward-Chaining

## Analyse von Backward-Chaining mit Regelgraphen

**Definition 9 (Regelgraph)**

Sei $R$ eine Regelmenge ohne Disjunktionen. Ein Regelgraph $G_R = \langle V, E \rangle$ ist ein gerichteter Graph, der wie folgt definiert ist:

1. Für jedes in $R$ vorkommende Atom $\kappa$ existiert ein Knoten $v_\kappa$ in $V$.
2. Für jede Regel $r \in R$ existiert ein Knoten $v_r$ in $V$.
3. Für jede Regel $r = \texttt{IF}\ \alpha_1 \land \ldots \land \alpha_n\ \texttt{THEN}\ \kappa$ existieren $n$ Kanten von $v_{\alpha_i}$ nach $v_r$ $(i = 1, \ldots, n)$ und eine Kante von $v_r$ nach $v_\kappa$ in $E$.
4. $G_R$ enthält keine anderen Knoten und Kanten.

**Definition 10 (zyklenfreie Regelmenge)**

Eine Regelmenge $R$ heißt zyklenfrei, wenn der zugehörige Regelgraph $G_R$ keine Zyklen enthält.

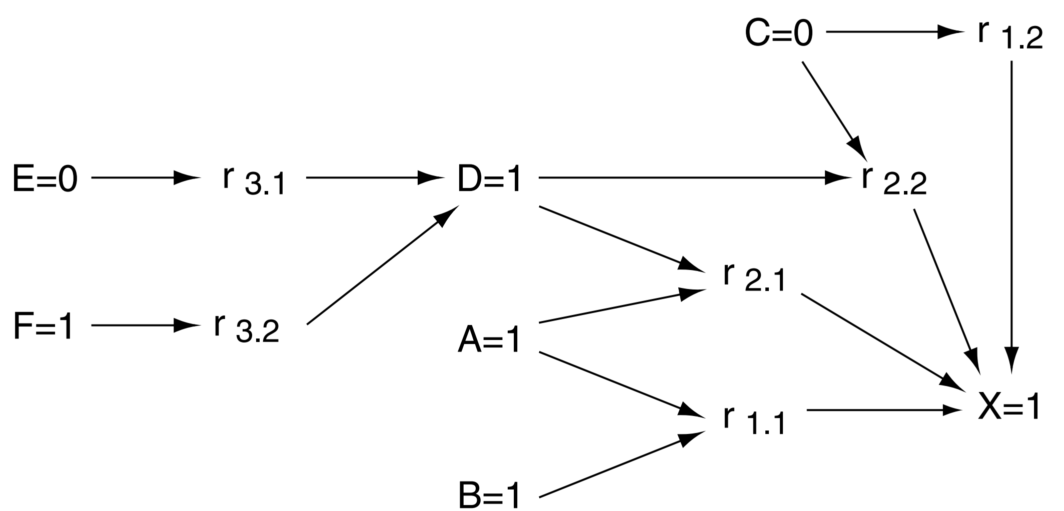
# Inferenz für Produktionsregelsysteme

## Beispiel: Regelgraph

Gegeben sei folgende Regelmenge:

$$
\begin{aligned}
R = \{ r_{1.1} &: \texttt{IF } A = 1 \land B = 1 \texttt{ THEN } X = 1 \\
r_{1.2} &: \texttt{IF } C = 0 \texttt{ THEN } X = 1 \\
r_{2.1} &: \texttt{IF } D = 1 \land A = 1 \texttt{ THEN } X = 1 \\
r_{2.2} &: \texttt{IF } D = 1 \land C = 0 \texttt{ THEN } X = 1 \\
r_{3.1} &: \texttt{IF } E = 0 \texttt{ THEN } D = 1 \\
r_{3.2} &: \texttt{IF } F = 1 \texttt{ THEN } D = 1
\end{aligned}
$$

Zugehöriger Regelgraph $G_R$:



Bemerkungen:

- Die Zyklenfreiheit eines zusammenhängenden gerichteten Graphen kann in linearer Zeit ($O(E)$) festgestellt werden.
- Der Algorithmus BC-DFS ist korrekt für zyklenfreie Regelmengen.
- Die Voraussetzung „zyklenfrei“ ist notwendig, da Tiefensuche auf unendlichen Graphen keine vollständige Suchstrategie darstellt.

$\Rightarrow$ Im Zusammenhang mit rückwärtsverkettenden Verfahren und der Kontrollstrategie Tiefensuche ist es notwendig, Schleifen während der Abarbeitung zu erkennen.

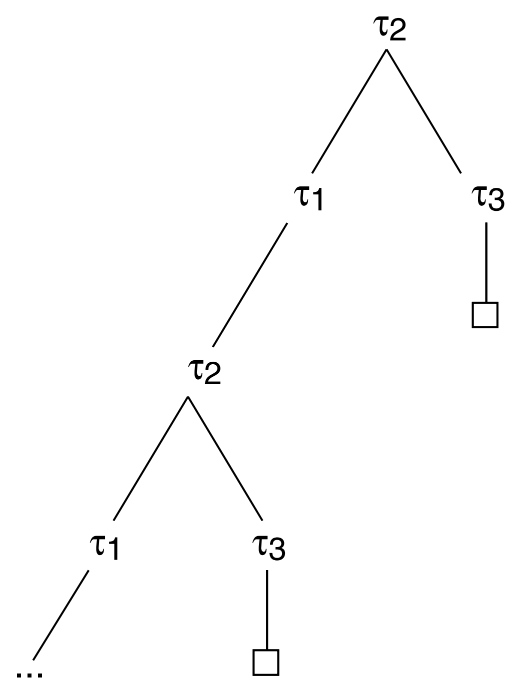
# Inferenz für Produktionsregelsysteme

## Beispiel: Zyklische Regelmenge

Sei folgendes Produktionsregelsystem $P = (D, R)$ gegeben:

$$D = \{\kappa_3\} \qquad R = \{r_1 : \texttt{IF } \kappa_1 \texttt{ THEN } \kappa_2$$
$$r_2 : \texttt{IF } \kappa_2 \texttt{ THEN } \kappa_1$$
$$r_3 : \texttt{IF } \kappa_3 \texttt{ THEN } \kappa_2\}$$

Ableitungsbaum $T_P(\kappa_2)$ für Ziel $\kappa_2$:



# Inferenz für Produktionsregelsysteme

## Laufzeit für Backward-Chaining

### Satz 11

Die Laufzeit des Algorithmus BC-DFS ist auch bei zyklenfreien Regelmengen $R$ nicht durch ein Polynom in Abhängigkeit von der Größe von $P = (D, R)$ beschränkt.

### Beweis

Gegeben sei folgendes Produktionsregelsystem $P_n = (D, R_n)$:

$$D = \{\kappa_0\} \qquad R_n = \{\texttt{IF } \alpha_{i,1} \wedge \alpha_{i,2} \texttt{ THEN } \kappa_i, \\ \texttt{IF } \kappa_{i-1} \texttt{ THEN } \alpha_{i,1}, \\ \texttt{IF } \kappa_{i-1} \texttt{ THEN } \alpha_{i,2} \mid 1 \leq i \leq n\}$$

(Die Anzahl der Regeln ist $3n$.)

Sei $\mu(n)$ die Anzahl der Aufrufe von BC-DFS für das Ziel $\kappa_n$.

# Inferenz für Produktionsregelsysteme

## Beweis Laufzeit BC-DFS (Fortsetzung).

- $n = 0$:

$$\mu(0) = 1, \text{ da } \kappa_0 \in D.$$

- $n = 1$:

$$\begin{array}{l}
\longrightarrow \texttt{BC-DFS}(\kappa_1) \\
\qquad \longrightarrow \texttt{BC-DFS}(\alpha_{1,1}) \\
\qquad\qquad \longrightarrow \texttt{BC-DFS}(\kappa_0) \quad (\Rightarrow \text{Anzahl der Aufrufe für } \kappa_0) \\
\qquad \texttt{AND} \\
\qquad \longrightarrow \texttt{BC-DFS}(\alpha_{1,2}) \\
\qquad\qquad \longrightarrow \texttt{BC-DFS}(\kappa_0) \quad (\Rightarrow \text{Anzahl der Aufrufe für } \kappa_0)
\end{array}$$

- Allgemein für $n > 0$:

$$\mu(n) = 3 + 2 \cdot \mu(n-1) = 3 \cdot \sum_{i=0}^{n-1} 2^i + 2^{n+1} = 3\frac{1-2^n}{1-2} + 2^{n+1}$$

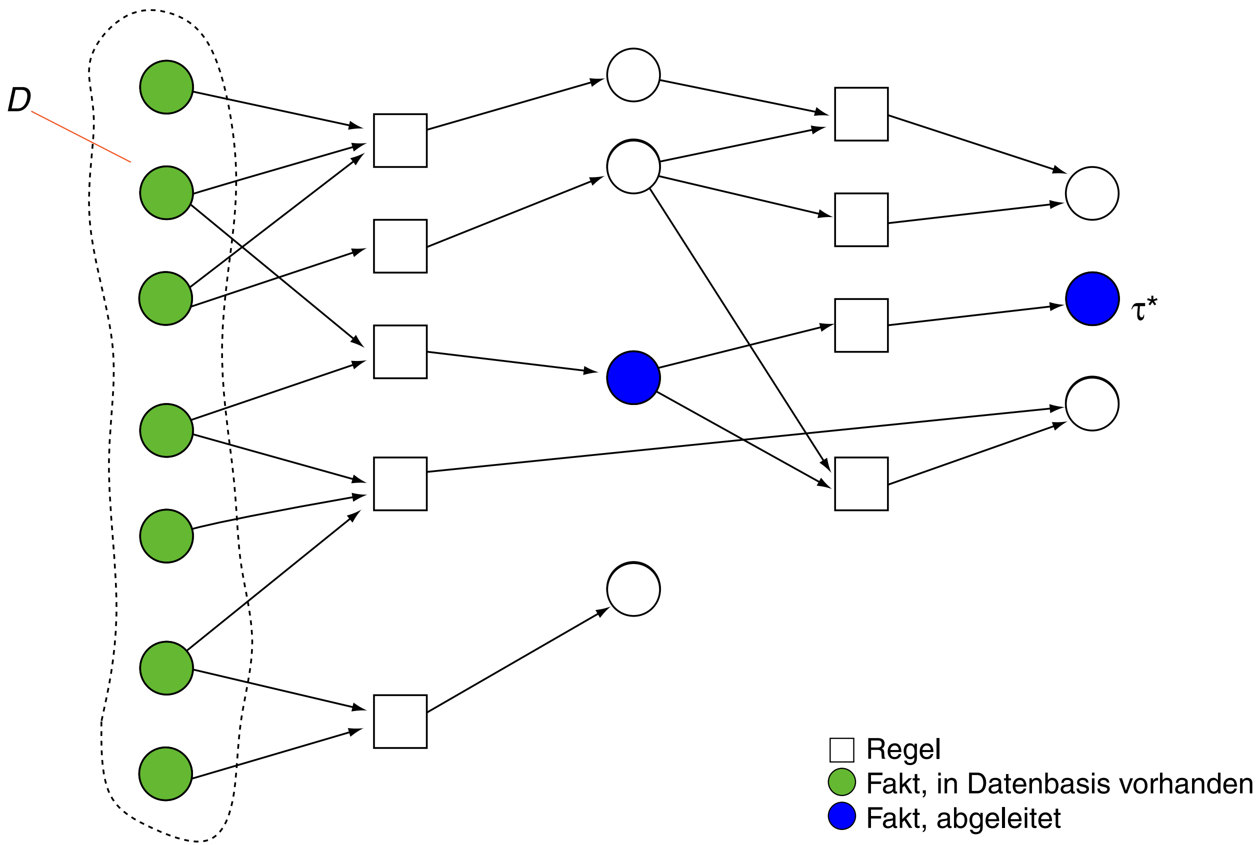
$$= 3 \cdot 2^n + 2^{n+1} - 3 \quad \geq 2^n$$

# Inferenz für Produktionsregelsysteme

## Verkettungsstrategien

Frage: Ist ein Atom ableitbar?

Bevorzugte Strategie: Backward-Chaining

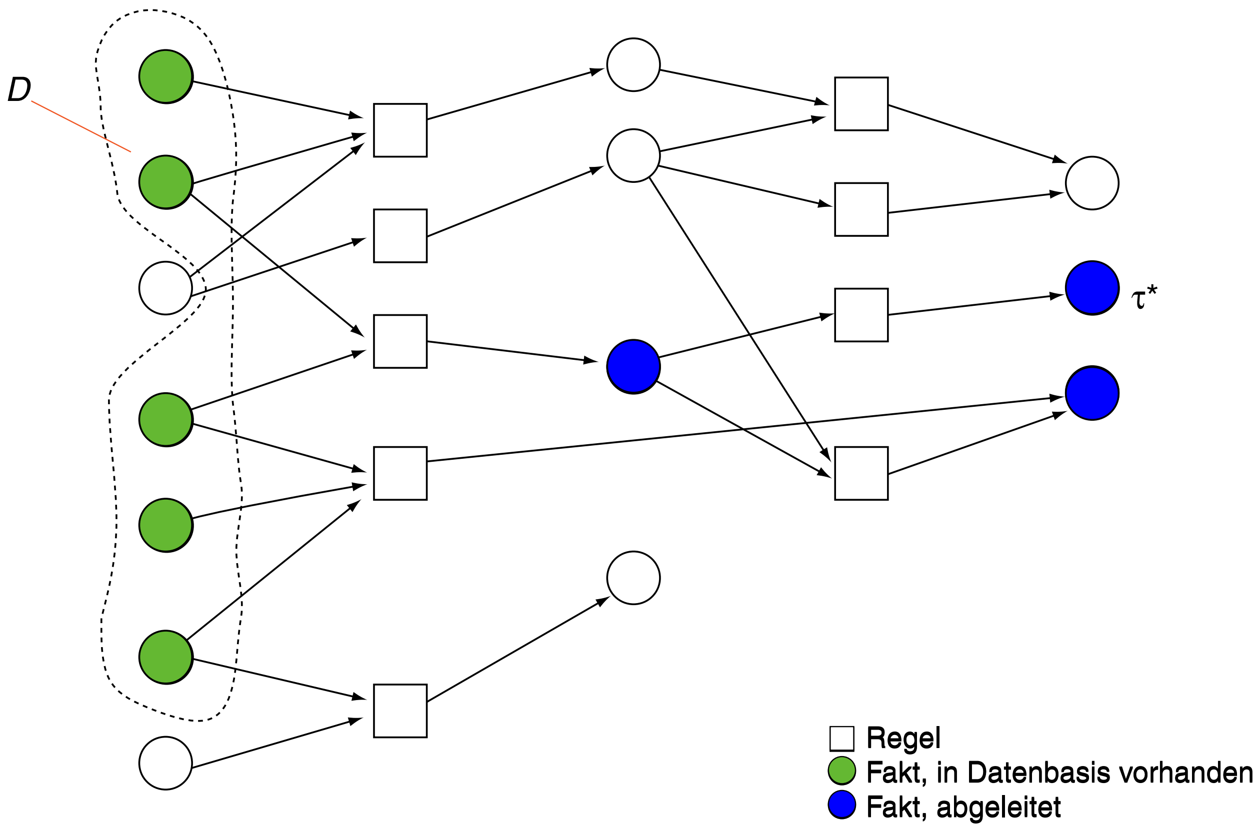


# Inferenz für Produktionsregelsysteme

## Verkettungsstrategien (Fortsetzung)

Frage: Wie sieht die Welt nach Anwendung aller Regeln aus?

Bevorzugte Strategie: Forward-Chaining



# Inferenz für Produktionsregelsysteme

## Verkettungsstrategien (Fortsetzung)

Neben einer reinen Backward-Chaining oder Forward-Chaining-Strategie können auch Kombinationen hieraus sinnvoll sein: Inferenz rückwärts vom Ziel und „gleichzeitig“ vorwärts von den Fakten.

Gemischte Strategie:

1. *Fokussierung* durch Erzeugung einer Teilwelt $D'$ durch Forward-Chaining.
2. Überprüfung von Hypothesen in $D'$ mit Hilfe von Backward-Chaining.

# Produktionsregelsysteme mit Negation

**Definition 12 (PS mit Negation)**

Ein Produktionsregelsystem mit Negation $P_N = (D, R_N)$ ist ein Produktionsregelsystem, bei dem der Bedingungsteil von Regeln auch die Negation `NOT` enthalten kann.

Beispiel:

$$\texttt{IF NOT}\ X \neq a \wedge \texttt{NOT}\ (Y = a \wedge Z \neq b)\ \texttt{THEN}\ W = a$$

Zwei Paradigmen zur Interpretation von `NOT`:

1. Negation-as-Failure
2. bezogen auf eine aktuelle, „statische“ Datenbasis

# Produktionsregelsysteme mit Negation

## Vereinfachung von Bedingungsteilen mit NOT

1. Mit Hilfe von de Morgan lassen sich Regeln mit NOT so umformen, dass die Negation nur bei Atomen steht. (Negationsnormalform des Bedingungsteils)

$$\texttt{NOT}\,(\alpha_1 \wedge \ldots \wedge \alpha_n) \;\approx\; \texttt{NOT}\,(\alpha_1) \vee \ldots \vee \texttt{NOT}\,(\alpha_n)$$
$$\texttt{NOT}\,(\alpha_1 \vee \ldots \vee \alpha_n) \;\approx\; \texttt{NOT}\,(\alpha_1) \wedge \ldots \wedge \texttt{NOT}\,(\alpha_n)$$

   Beispiel:

$$\texttt{IF NOT}\,X \neq a \wedge \texttt{NOT}\,(Y = a \wedge Z \neq b)\;\texttt{THEN}\;W = a$$
$$\approx\; \texttt{IF NOT}\,X \neq a \wedge (\texttt{NOT}\,Y = a \vee \texttt{NOT}\,Z \neq b)\;\texttt{THEN}\;W = a$$

# Produktionsregelsysteme mit Negation

## Vereinfachung von Bedingungsteilen mit $\texttt{NOT}$

1. Mit Hilfe von de Morgan lassen sich Regeln mit $\texttt{NOT}$ so umformen, dass die Negation nur bei Atomen steht. (Negationsnormalform des Bedingungsteils)

$$\texttt{NOT}\,(\alpha_1 \wedge \ldots \wedge \alpha_n) \;\approx\; \texttt{NOT}\,(\alpha_1) \vee \ldots \vee \texttt{NOT}\,(\alpha_n)$$
$$\texttt{NOT}\,(\alpha_1 \vee \ldots \vee \alpha_n) \;\approx\; \texttt{NOT}\,(\alpha_1) \wedge \ldots \wedge \texttt{NOT}\,(\alpha_n)$$

   Beispiel:

$$\texttt{IF NOT}\,X \neq a \wedge \texttt{NOT}\,(Y = a \wedge Z \neq b)\;\texttt{THEN}\;W = a$$
$$\approx\;\texttt{IF NOT}\,X \neq a \wedge (\texttt{NOT}\,Y = a \vee \texttt{NOT}\,Z \neq b)\;\texttt{THEN}\;W = a$$

2. Darauf aufbauend lässt sich die disjunktive Normalform herstellen und die Regeln aufspalten:

$$\texttt{IF NOT}\,X \neq a \wedge (\texttt{NOT}\,Y = a \vee \texttt{NOT}\,Z \neq b)\;\texttt{THEN}\;W = a$$
$$\approx\;\begin{array}{l}\texttt{IF NOT}\,X \neq a \wedge \texttt{NOT}\,Y = a\;\texttt{THEN}\;W = a\\ \texttt{IF NOT}\,X \neq a \wedge \texttt{NOT}\,Z \neq b\;\texttt{THEN}\;W = a\end{array}$$

# Produktionsregelsysteme mit Negation

## Interpretation von `NOT` als Negation-as-Failure

- wird in der Programmiersprache PROLOG verwandt
- hier rein aussagenlogischer Fall:
  Die „Bedingung `NOT` $\kappa$“ für ein Atom $\kappa$ ist erfüllt, falls $\kappa$ nicht ableitbar ist.
- Hintergrund dieser Interpretation ist die Closed World Assumption (CWA).

### Annahme:

- Die Diskurswelt (Domäne, Situation) ist vollständig durch $P_N = (D, R_N)$ beschrieben.

$\Rightarrow$ Alle Fakten, die in der Diskurswelt gültig sind, sind auch ableitbar.

$\Rightarrow$

Failure bzgl. des Ableitens von $\kappa$

$\Leftrightarrow$

„`NOT` $\kappa$“ gilt in der Diskurswelt

# Produktionsregelsysteme mit Negation

## Interpretation von `NOT` als Negation-as-Failure

**Definition 13 (Semantik von `NOT` unter CWA)**

In einem Produktionsregelsystem $P_N = (D, R_N)$ ist eine Bedingung $\texttt{NOT}\,\alpha$ genau dann erfüllt (wahr), wenn $\alpha$ nicht aus $P_N$ ableitbar ist. Das heißt:

1. Ist $\alpha$ eine Konjunktion von Teilformeln $\alpha_i$ darf mindestens ein $\alpha_i$ nicht ableitbar sein, damit $\texttt{NOT}\,\alpha$ erfüllt ist.

2. Ist $\alpha$ eine Disjunktion von Teilformeln $\alpha_i$ so darf kein $\alpha_i$ ableitbar sein, damit $\texttt{NOT}\,\alpha$ erfüllt ist.

Bemerkungen:

- Dieser Erfüllbarkeitsbegriff kann unmittelbar in den Algorithmus BC-DFS integriert werden.
- Mit Negation-as-Failure wird eine neue Schlussregel eingeführt – in Zeichen:

$$(\alpha \;\not\vdash_{PS}\; \kappa) \;\; \vdash_{\substack{PS_N \\ CWA}} \;\; \neg\kappa$$

  In Worten: Falls $\kappa$ aus $\alpha$ nicht mittels $\vdash_{PS}$ ableitbar ist, so ist $\neg\kappa$ unter der Closed-World-Assumption ableitbar.

# Produktionsregelsysteme mit Negation

Algorithm: BC-DFS-N
Input: Startdatenbasis $D$, Regelmenge $R$, Formel $\alpha$
Output: *true*, falls $\alpha$ ableitbar unter CWA, *false* sonst; evtl. Endlosschleife

```
BEGIN
  IF α = NOT α1 THEN RETURN (NOT BC-DFS-N(α1)) ENDIF

  IF α = α1 ∧ α2 THEN RETURN (BC-DFS-N(α1) AND BC-DFS-N(α2)) ENDIF
  IF α = α1 ∨ α2 THEN RETURN (BC-DFS-N(α1) OR BC-DFS-N(α2)) ENDIF
  IF α ∈ D THEN RETURN (true) ENDIF
```

# Produktionsregelsysteme mit Negation

Algorithm: BC-DFS-N

Input: Startdatenbasis $D$, Regelmenge $R$, Formel $\alpha$

Output: *true*, falls $\alpha$ ableitbar unter CWA, *false* sonst; evtl. Endlosschleife

```
BEGIN
  IF α = NOT α₁ THEN RETURN (NOT BC-DFS-N(α₁)) ENDIF

  IF α = α₁ ∧ α₂ THEN RETURN (BC-DFS-N(α₁) AND BC-DFS-N(α₂)) ENDIF
  IF α = α₁ ∨ α₂ THEN RETURN (BC-DFS-N(α₁) OR BC-DFS-N(α₂)) ENDIF
  IF α ∈ D THEN RETURN (true) ENDIF

  R* = {r | r = (IF γ THEN α) und r ∈ R}
  stop=false

  WHILE R* ≠ ∅ AND stop=false DO
      r = choose(R*)
      IF BC-DFS-N(premise(r))=true
      THEN stop=true
      ELSE R* = R* \ {r}
  END

  IF stop=true
  THEN RETURN (true)
  ELSE RETURN (false)

END
```

# Produktionsregelsysteme mit Negation

## Zyklische Regelmengen und NOT

Sei folgendes Produktionsregelsystem $P_N = (D, R_N)$ gegeben:

$$D = \{\} \qquad R_N = \{r_1 : \texttt{IF NOT}\, X = a \;\texttt{THEN}\; Y = b,$$
$$r_2 : \texttt{IF NOT}\, Y = b \;\texttt{THEN}\; X = a\}$$

- $R_N$ enthält eine Schleife für das Ziel $Y = b$ und für das Ziel $X = a$.
- Schleifen (unendliche Ableitungen) dürfen nicht mit der Nicht-Ableitbarkeit eines Faktes gleichgesetzt werden.

# Produktionsregelsysteme mit Negation

## Negation-as-Failure und Vorwärtsverkettung

Bei der Vorwärtsverkettung hängt die Erfüllung einer Bedingung von der aktuellen Datenbasis $D$ ab.

$\Rightarrow$ Die Bildung der Konfliktmenge hängt vom aktuellen $D$ ab.

Im Widerspruch dazu steht Negation-as-Failure:

- Die Erfüllung einer Bedingung hängt von der Ableitbarkeit ab.

$\Rightarrow$ Für $(D, R_N)$ macht ein rein vorwärtsverkettendes Verfahren keinen Sinn, weil bei negierten Bedingungen die Ableitbarkeit von Atomen getestet werden muss.

$\Rightarrow$ Die Integration eines rückwärtsverkettenden Verfahrens und die Kombination beider Verkettungsstrategien ist notwendig.

# Produktionsregelsysteme mit Negation

## Negation-as-Failure und Vorwärtsverkettung (Fortsetzung)

Sei folgendes Produktionsregelsystem $P_N = (D, R_N)$ gegeben:

$$D = \{\} \qquad R_N = \{r_1 : \texttt{IF}\ Z = a\ \texttt{THEN}\ X = b,$$
$$r_2 : \texttt{IF NOT}\ Y = b\ \texttt{THEN}\ Z = a,$$
$$r_3 : \texttt{IF}\ U = 1\ \texttt{THEN}\ Y = b\}$$

- Test, ob $r_2$ in die Konfliktmenge kommt.

$\Rightarrow$ Test, ob $Y = b$ abgeleitet werden kann.

$\Rightarrow$ Backward-Chaining

### Alternative

Anwendung einer anderen Interpretation der Negation bei vorwärtsverkettenden Verfahren. Idee: Konfliktmengenbildung bei statischer Datenbasis $D$.

# Produktionsregelsysteme mit Negation

## Interpretation von `NOT` bzgl. statischer Datenbasis und Vorwärtsverkettung

**Definition 14 (Semantik von `NOT` unter $D$)**

Eine Bedingung `NOT` $\alpha$ ist in Bezug auf eine Datenbasis $D$ genau dann erfüllt, wenn $\alpha$ in Bezug auf $D$ nicht erfüllt ist:

- Ist $\alpha$ ein Atom, so muss $\alpha \notin D$ gelten.
- Andernfalls wird das Erfülltsein von $\alpha$ entsprechend der Junktoren auf das Erfülltsein der Teilformeln zurückgeführt.

# Produktionsregelsysteme mit Negation

## Lemma 15

Produktionsregelsysteme mit Negation und der Interpretation der Negation in Bezug auf die Datenbasis sind nicht kommutativ.

## Beweis

Sei folgendes Produktionsregelsystem $P_N = (D, R_N)$ gegeben:

$$D = \{\} \qquad R_N = \{r_1 : \texttt{IF NOT}\, X = a \;\texttt{THEN}\; Y = b, \\ r_2 : \texttt{IF NOT}\, Y = b \;\texttt{THEN}\; X = a\}$$

Wegen $D = \emptyset$ ist sowohl $r_1$ als auch $r_2$ anwendbar. Wähle $r_1$.

$\Rightarrow$ $(D, R_N) \mid\!\frac{1}{PS}\, (D_1, R_N)$ mit $D_1 = \{Y = b\}$.

$\Rightarrow$ Für $D_1$ ist die Bedingung von $r_2$ nicht länger erfüllt.

$\Rightarrow$ $r_2$ ist nicht anwendbar.

$\Rightarrow$ $P_N$ nicht kommutativ.

# Produktionsregelsysteme mit Negation

Algorithm: FC-N-test

Input: Startdatenbasis $D$, Regelmenge $R_N$, Atom $\kappa^*$

Output: *true*, falls $(D, R) \mid_{\overline{PS}} \kappa^*$, *unknown* sonst

```
BEGIN
  D* = D
  R_tmp = R

  REPEAT
    R* = {(IF α THEN κ) ∈ R_tmp | α wahr bzgl. D*}
    IF R* ≠ ∅
    THEN BEGIN
      r = choose(R*)
      D* = D* ∪ {conclusion(r)}
      R_tmp = R_tmp \ {r}
    END
    ELSE R_tmp = ∅
  UNTIL R_tmp = ∅

  IF κ* ∈ D*
  THEN RETURN (true)
  ELSE RETURN (unknown)
END
```

Bemerkungen:

- FC-N-test terminiert bei jeder Eingabe.
- Aufgrund der Nicht-Kommutativität kommt dem Aufruf *choose*($R^*$) eine besondere Bedeutung zu: Nicht jede Auswahl von Regeln liefert das Ergebnis *true*, auch wenn $(D, R_N) \mathrel{|_{\overline{PS}}} \kappa$ gilt.

# Produktionsregelsysteme mit Negation

**Satz 16 (Korrektheit und Vollständigkeit von FC-N-test)**

Es sei $P_N$ ein Produktionsregelsystem mit Negation und $\kappa$ ein Atom. Dann gilt FC-N-test$(D, R_N, \kappa)$ = *true* ist möglich genau dann, wenn sich $\kappa$ aus $P_N$ mit Interpretation der Negation in Bezug auf die Datenbasis ableiten lässt, d.h. $(D, R_N) \mid\!\underline{\;\;}_{PS}\, \kappa$ gilt.

# Produktionsregelsysteme mit Negation

## Beweis (Korrektheit und Vollständigkeit von FC-N-test)

### „⇒“ Korrektheit

Aus FC-N-test$(D, R_N, \kappa)$ = *true* ist möglich folgt $(D, R_N) \mathrel{|}\!\!\underline{\phantom{x}}_{PS}\ \kappa$.

Klar, weil jeder Iterationsschritt des Algorithmus genau einem Schritt der Ableitung $\mathrel{|}\!\!\frac{1}{PS}$ entspricht.

### „⇐“ Vollständigkeit

Aus $(D, R_N) \mathrel{|}\!\!\underline{\phantom{x}}_{PS}\ \kappa$ folgt, dass eine Ableitungsfolge für FC-N-test$(D, R_N, \kappa)$ existiert mit FC-N-test$(D, R_N, \kappa)$ = *true*.

- Nach Voraussetzung existiert eine Folge von Regelanwendungen

  $$(D, R_N) \mathrel{|}\!\!\frac{1}{PS} (D_1, R_N) \mathrel{|}\!\!\frac{1}{PS} \ \ldots \ \mathrel{|}\!\!\frac{1}{PS} (D_k, R_N) \text{ mit } \kappa \in D_k,$$

  wobei $D_i$ aus $D_{i-1}$ durch Anwendung einer Regel entsteht.

- Wähle die entsprechenden Regeln in dieser Reihenfolge für die ersten $k$ Schleifendurchläufe in FC-N-test.

$\Rightarrow$ $D_k \subseteq D^*$

$\Rightarrow$ $\kappa$ wurde abgeleitet.

# Produktionsregelsysteme mit Negation

## Nicht-Determinismus von FC-N-test

- Aus $(D, R_N) \mid_{\overline{PS}} \kappa$ folgt nicht, dass FC-N-test$(D, R_N, \kappa)$ den Rückgabewert *true* liefern muss.
- Im Falle der Nichtableitung von $\kappa$ ist der Rückgabewert von FC-N-test *unknown*.
- Unter der Voraussetzung P $\neq$ NP lässt sich der Nichtdeterminismus von FC-N-test auch nicht so auflösen, dass ein polynomiell beschränktes deterministisches Verfahren zur Bestimmung der Ableitbarkeit entsteht:

**Satz 17 (NP-Vollständigkeit des Ableitbarkeitsproblems)**

Es sei $P_N$ ein Produktionsregelsystem mit Negation und $\kappa$ ein Atom. Das Entscheidungsproblem „Lässt sich $\kappa$ aus $P_N$ ableiten?“ – kurz: „Gilt $P_N \mid_{\overline{PS}} \kappa$ ?“ – ist NP-vollständig.

# Produktionsregelsysteme mit Negation

## Beweis (Skizze: NP-Vollständigkeit des Ableitbarkeitsproblems)

1. Obere Schranke.
   $P_N \mid_{\overline{PS}} \kappa$ ist in NP; Argumentation über FC-N-test.

2. Vollständigkeit. Reduktion von 3SAT auf $P_N \mid_{\overline{PS}} \kappa$.
   Konstruktion einer Menge $R_\alpha$ von Regeln zu einer aussagenlogischen Formel $\alpha$ mit

$$\alpha \text{ erfüllbar} \quad \Leftrightarrow \quad P_\alpha = (\emptyset, R_\alpha) \mid_{\overline{PS}} \kappa, \;\; \kappa = (Y = 1)$$

Argumentation zu Punkt 2:

„⇒" Mit Erfüllbarkeit von $\alpha$ folgt $P_\alpha \mid_{\overline{PS}} (Y = 1)$: Die erfüllende Belegung $\Im$ der Atome in $\alpha$ lässt die Regeln so feuern, dass $Y = 1$ von $P_\alpha$ abgeleitet werden kann.

„⇐" Mit $P_\alpha \mid_{\overline{PS}} (Y = 1)$ folgt die Erfüllbarkeit von $\alpha$: Aus den gefeuerten Regeln folgt eine erfüllende Belegung $\Im$ der Atome in $\alpha$.