Thom Frühwirth, Slim Abdennadher LMU München

Logikprogrammierung

A logic program is a set of axioms, or rules, defining relationships between objects. A computation of a logic program is a deduction of consequences of the program. A program defines a set of consequences, which is its meaning. The art of logic programming is constructing concise and elegant programs that have desired meaning.
Sterling und Shapiro: The Art of Prolog, Seite 1

$$\text{Algorithm} = \text{Logic} + \text{Control}$$

Robert A. Kowalski, 1979

Mai 1998

Anmerkung

Logic Was
Logische Komponente
Problembeschreibung

Control Wie
Steuerkomponente
Programmablauf

imperativ, prozedural, befehlsorientiert

deklarativ, logikorientiert

Beispiel Liste sortieren:

Durch paarweises Vertauschen

Permutation + Ordnungstest

Spezifikation, Rapid Prototyping mittels logischer Formeln

allgemeines Wissen durch Regeln, konkrete Annahmen als Fakten

Abfragen=Problembeschreibung, mittels Folgerungstest

Atome:	A, B	::=	$p(t_1, \ldots, t_n),\ n \geq 0$
Ziele:	G, H	::=	$\top \mid \bot \mid A \mid G \wedge H$
Klauseln:	K	::=	$A \leftarrow G$
Programme:	P	::=	$\{K_1, \ldots, K_m\},\ m \geq 0$

Anmerkung

Wir verwenden zur Definition der abstrakten Syntax neben einer verbalen Definition auch die Darstellung in einer erweiterten Backus-Naur-Form (BNF). Eine Produktionsregel

Name: $G, H ::= A \mid B,$ Bedingung

bedeutet, daß die Symbole G bzw. H (eventuell mit Indizes, z.B. G_2, oder gestrichen, z.B. G') „Name“ genannt werden und von der Form A oder von der Form B sein können, falls die nachfolgende Bedingung erfüllt ist. Großbuchstaben stehen dabei für Ausdrücke, die wiederum durch Produktionsregeln definiert sind.

LP-Syntax

Definition 1 Ein *Ziel* (engl. goal) ist entweder $\top$ oder $\bot$ oder ein Atom oder eine Konjunktion von Zielen. $\lhd$

Definition 2 Eine *(L-)Klausel (definite Klausel)* hat die Form $A \leftarrow G$, wobei wir das Atom A als *Kopf* (engl. head) und das Ziel G als *Rumpf* (engl. body) der Klausel bezeichnen.

Klauseln von der Form $A \leftarrow \top$ werden *Fakten* genannt, alle anderen *Regeln*.

Ein *Logikprogramm* ist eine endliche Menge von L-Klauseln. $\lhd$

Anmerkung

Zustände

Definition 3 Ein *Zustand* ist ein Paar $\langle G, \theta \rangle$, wobei G ein Ziel ist und θ eine Substitution. Ein *Anfangszustand* ist ein Zustand von der Form $\langle G, \varepsilon \rangle$, wobei ε die Identitätssubstitution ist.

Ein Zustand heißt *erfolgreicher Endzustand*, falls er von der Form $\langle \top, \theta \rangle$ ist. Er heißt *erfolgloser Endzustand (gescheitert)*, falls er von der Form $\langle \bot, \varepsilon \rangle$ ist. $\lhd$

Anmerkung

Ableitungen

Definition 4 Eine Ableitung ist *erfolgreich*, wenn ihr Endzustand erfolgreich ist. Eine Ableitung ist *erfolglos* (*gescheitert*), wenn ihr Endzustand erfolglos (gescheitert) ist. Eine Ableitung ist *unendlich*, wenn sie keinen Endzustand hat.

Ein Ziel G ist *erfolgreich*, wenn es eine erfolgreiche Ableitung beginnend mit $\langle G, \varepsilon \rangle$ hat. Ein Ziel G ist *erfolglos (endlich gescheitert)*, wenn es nur erfolglose Ableitungen beginnend mit $\langle G, \varepsilon \rangle$ hat. $\lhd$

Anmerkung

Logische Beschreibung

Definition 5 Jedem Zustand $\langle H, \theta \rangle$, der in einer Ableitung beginnend mit $\langle G, \varepsilon \rangle$ vorkommt, wird eine Formel $\exists \bar{x} H\theta$ als *logische Beschreibung* zugeordnet, wobei $\bar{x}$ für die Sequenz der Variablen steht, die in $H\theta$, aber nicht in G vorkommen. Die in G vorkommenden Variablen bleiben frei (unquantifiziert). $\lhd$

Anmerkung

Antwort

Definition 6 Eine Substitution θ wird *(berechnete) Antwort eines Zieles* G genannt, falls es eine erfolgreiche Ableitung $\langle G, \varepsilon \rangle \mapsto^* \langle \top, \beta \rangle$ gibt, so daß θ die eingeschränkte Substitution von β auf die Variablen von G ist. $\lhd$

Anmerkung

Im LP-Kalkül werden die Substitutionen explizit mitgeführt. Man ist nämlich nicht nur daran interessiert festzustellen, ob eine Instanz des Ziels G logisch aus einem Programm P folgt, sondern man möchte auch diese Instanz kennen.

Kongruenz

Kommutativität:	$G_1 \wedge G_2$	$\equiv$	$G_2 \wedge G_1$
Assoziativität:	$G_1 \wedge (G_2 \wedge G_3)$	$\equiv$	$(G_1 \wedge G_2) \wedge G_3$
Identität:	$G \wedge \top$	$\equiv$	G
Absorption:	$G \wedge \bot$	$\equiv$	$\bot$

Anmerkung

Die Kongruenz $\equiv$ besagt, daß es gleichgültig ist, in welcher Reihenfolge und Klammerung die Atome in einer Konjunktion auftreten. Darüber hinaus ist $\top$ in einer Konjunktion überflüssig, während $\bot$ alle anderen Konjunkte überflüssig macht. Wir werden für den Rest des Buches diese Kongruenz beibehalten.

LP-Reduktionsregeln

Entfalten:

$$\frac{(B \leftarrow H) \text{ aus } P \qquad \beta \text{ ist allgemeinster Unifikator von } B \text{ und } A\theta}{\langle A \wedge G, \theta \rangle \mapsto_{Entfalten} \langle H \wedge G, \theta\beta \rangle}$$

Scheitern:

$$\frac{\text{Es gibt keine Klausel } (B \leftarrow H) \text{ aus } P, \qquad \text{so daß ein Unifikator von } B \text{ und } A\theta \text{ existiert}}{\langle A \wedge G, \theta \rangle \mapsto_{Scheitern} \langle \bot, \varepsilon \rangle}$$

Anmerkung

In unserem einfachen LP-Kalkül ist die Auswahl der Klausel innerhalb eines Programms P und die Auswahl des Atoms A aus dem Ziel nicht vorbestimmt (Nichtdeterminismus). Während die Klauselselektion über Erfolg oder Scheitern einer Ableitung entscheiden kann, beeinflußt die Selektion des Atoms lediglich die Länge der Ableitung (im schlimmsten Fall unendlich).

LP-Kalkül Beispiel

$\{$kante(a,b) $\leftarrow$ $\top$,
kante(a,c) $\leftarrow$ $\top$,
kante(b,d) $\leftarrow$ $\top$,
kante(c,d) $\leftarrow$ $\top$,
kante(d,e) $\leftarrow$ $\top$,

verbindung(Start,Ziel) $\leftarrow$
kante(Start,Ziel),

verbindung(Start,Ziel) $\leftarrow$
kante(Start,Knoten) $\wedge$
verbindung(Knoten,Ziel)$\}$

Anmerkung

Zur Illustration wollen wir die Erreichbarkeit in einem gerichteten Graphen mithilfe eines Logikprogramms untersuchen.

Die Kanten des Graphen werden durch das Prädikat `kante/2` beschrieben. Die direkten und indirekten Verbindungen zwischen den Knoten des Graphen werden durch das Prädikat `verbindung/2` definiert.

LP-Kalkül Beispiel

$\{\texttt{kante(a,b)} \leftarrow \top,\ \texttt{kante(a,c)} \leftarrow \top,$
$\texttt{kante(b,d)} \leftarrow \top,\ \texttt{kante(c,d)} \leftarrow \top,$
$\texttt{kante(d,e)} \leftarrow \top,$

$\texttt{verbindung(Start,Ziel)} \leftarrow$
$\quad \texttt{kante(Start,Ziel)},$
$\texttt{verbindung(Start,Ziel)} \leftarrow$
$\quad \texttt{kante(Start,Knoten)} \wedge$
$\quad \texttt{verbindung(Knoten,Ziel)}\}$

$$\begin{aligned} & \langle \texttt{verbindung(b, Y)}, \epsilon \rangle \\ \mapsto_{Entfalten} \quad & \langle \texttt{kante(S, Z)}, \{\texttt{S} \leftarrowtail \texttt{b}, \texttt{Z} \leftarrowtail \texttt{Y}\} \rangle \\ \mapsto_{Entfalten} \quad & \langle \top, \{\texttt{S} \leftarrowtail \texttt{b}, \texttt{Z} \leftarrowtail \texttt{d}, \texttt{Y} \leftarrowtail \texttt{d}\} \rangle \end{aligned}$$

Anmerkung

Man kann jetzt alle vom Knoten `b` aus erreichbaren Knoten `Y` bestimmen, durch das Ziel `verbindung(b,Y)`.

Wenn man die erste Regel für das Prädikat `verbindung` und eine passende Kante auswählt, dann erhält man folgende Ableitung:

Eine berechnete Antwort für das obige Programm und das Ziel `verbindung(b,Y)` ist dann $\theta = \{\texttt{Y} \leftarrow\!\shortmid \texttt{d}\}$, d. h. `d` ist ein aus `b` erreichbarer Knoten.

LP-Kalkül Beispiel

$\{\texttt{kante(a,b)} \leftarrow \top,\ \texttt{kante(a,c)} \leftarrow \top,$
$\texttt{kante(b,d)} \leftarrow \top,\ \texttt{kante(c,d)} \leftarrow \top,$
$\texttt{kante(d,e)} \leftarrow \top,$

$\texttt{verbindung(Start,Ziel)} \leftarrow$
$\quad \texttt{kante(Start,Ziel)},$
$\texttt{verbindung(Start,Ziel)} \leftarrow$
$\quad \texttt{kante(Start,Knoten)} \wedge$
$\quad \texttt{verbindung(Knoten,Ziel)}\}$

$$
\begin{aligned}
& \langle \texttt{verbindung(b,Y)}, \epsilon \rangle \\
\mapsto_{E.}\ & \langle \texttt{kante(S,K)} \wedge \texttt{verbindung(K,Z)}, \{\texttt{S} \leftarrowtail \texttt{b}, \texttt{Z} \leftarrowtail \texttt{Y}\} \rangle \\
\mapsto_{E.}\ & \langle \texttt{verbindung(K,Z)}, \{\texttt{S} \leftarrowtail \texttt{b}, \texttt{Z} \leftarrowtail \texttt{Y}, \texttt{K} \leftarrowtail \texttt{d}\} \rangle \\
\mapsto_{E.}\ & \langle \texttt{kante(K,Z)}, \{\texttt{S} \leftarrowtail \texttt{b}, \texttt{Z} \leftarrowtail \texttt{Y}, \texttt{K} \leftarrowtail \texttt{d}\} \rangle \\
\mapsto_{E.}\ & \langle \top, \{\texttt{S} \leftarrowtail \texttt{b}, \texttt{Z} \leftarrowtail \texttt{d}, \texttt{K} \leftarrowtail \texttt{d}, \texttt{Y} \leftarrowtail \texttt{e}\} \rangle
\end{aligned}
$$

Anmerkung

Wenn aber anstatt der ersten Regel für `verbindung` die zweite Regel ausgewählt wird, erhält man folgende Ableitung:

Eine zweite berechnete Antwort ist $\sigma = \{\mathtt{Y} \leftarrowtail \mathtt{e}\}$.

LP-Kalkül Beispiel

$\{\texttt{kante(a,b)} \leftarrow \top,\ \texttt{kante(a,c)} \leftarrow \top,$
$\texttt{kante(b,d)} \leftarrow \top,\ \texttt{kante(c,d)} \leftarrow \top,$
$\texttt{kante(d,e)} \leftarrow \top,$

$\texttt{verbindung(Start,Ziel)} \leftarrow$
$\quad \texttt{kante(Start,Ziel)},$
$\texttt{verbindung(Start,Ziel)} \leftarrow$
$\quad \texttt{kante(Start,Knoten)} \wedge$
$\quad \texttt{verbindung(Knoten,Ziel)}\}$

$$\begin{aligned} & \langle \texttt{verbindung(f,g)}, \epsilon \rangle \\ \mapsto_{E.} \quad & \langle \texttt{kante(S,Z)}, \{\texttt{S} \leftarrowtail \texttt{f}, \texttt{Z} \leftarrowtail \texttt{g}\} \rangle \\ \mapsto_{Sch.} \quad & \langle \bot, \epsilon \rangle \end{aligned}$$

$$\begin{aligned} & \langle \texttt{v(f,g)}, \epsilon \rangle \\ \mapsto_{E.} \quad & \langle \texttt{k(S,K)} \wedge \texttt{v(K,Z)}, \{\texttt{S} \leftarrowtail \texttt{f}, \texttt{Z} \leftarrowtail \texttt{g}\} \rangle \\ \mapsto_{E.} \quad & \langle \texttt{k(S,K)} \wedge \texttt{k(K,K1)} \wedge \texttt{v(K1,Z)}, \{\texttt{S} \leftarrowtail \texttt{f}, \texttt{Z} \leftarrowtail \texttt{g}\} \rangle \end{aligned}$$

Anmerkung

Betrachten wir nun die Frage, ob zwischen den Knoten `f` und `g` eine Verbindungskante existiert, d. h. das Ziel `verbindung(f,g)`. Mit der ersten Regel erhalten wir:

Diese Ableitung ist also erfolglos. Die Auswahl der zweiten Regel führt ebenfalls zu einer erfolglosen Ableitung, wenn im zweiten Ableitungsschritt das Ziel `kante(S,K)` zur Reduktion selektiert wird. Eine andere Möglichkeit ist es, das Ziel `verbindung(K,Z)` und die zweite Regel für `verbindung` auszuwählen:

Es ist leicht ersichtlich, daß sich bei Wiederholung der gleichen Auswahl eine unendliche Ableitung ergibt.

Stabilität

LP-Sprachen besitzen folgende vorteilhafte Eigenschaft, die Stabilität genannt wird:

Wenn $\langle G, \theta \rangle \mapsto_{Entfalten} \langle G', \theta' \rangle$ und α sowie $\theta'\alpha$ sind Substitutionen, dann ist auch $\langle G \wedge H, \theta'\alpha \rangle \mapsto_{Entfalten} \langle G' \wedge H, \theta'\alpha \rangle$ möglich.

Anmerkung

Den Begriff "Monotonie", der dafür auch in der Literatur vorkommt, werden wir für eine stärkere Form der Stabilität bei CCLP-Sprachen verwenden.

Wenn eine Reduktion *Entfalten* zu einem Folgezustand mit Substitution θ' führt, dann kann diese Reduktion auch auf jeden Zustand mit erweiterten Ziel und zusätzlichen Substitutionen zu θ' angewendet werden und es treten im Folgezustand keine neuen Substitutionen auf.

Diese Eigenschaft ist einer der Vorteile von LP gegenüber herkömmlichen Programmiersprachen mit überschreibender Zuweisung (engl. destructive assignment).

Vervollständigung

Beispiel 1 Sei G das Ziel `verbindung(f,g)`. Das Ziel hat keine erfolgreiche Ableitung. Es gil:

$P^{\rightarrow} \not\models$ `verbindung(f,g)` und

$P^{\rightarrow} \not\models \neg$`verbindung(f,g)`

Wenn man $P^{\rightarrow}$ aber so „vervollständigt“, daß es nicht nur die aus den Klauseln resultierenden notwendigen Bedingungen (Implikationen), sondern auch die entsprechenden hinreichenden Bedingungen enthält (Implikation in die andere Richtung), dann folgt daraus $\neg$`verbindung(f,g)`. $\diamond$

Anmerkung

Eine L-Klausel $A \leftarrow G$ kann als Implikation $G \rightarrow A$ verstanden werden. G ist die Konjunktion $A_1 \wedge \ldots \wedge A_n$. Die Implikation entspricht der Klausel $A \vee \neg A_1 \vee \ldots \vee \neg A_n$, was den Namen L-Klausel erklärt. Die logische Leseweise eines Programms P ist dann der universelle Abschluß der Konjunktion der Klauseln von P, wir schreiben dafür $P^{\rightarrow}$.

Mit dieser einfachen deklarativen Semantik können allerdings nur positive Informationen erschlossen werden, nicht aber negative.

Vervollständigung

Definition 7 *Vervollständigung* von P, $P^{\leftrightarrow}$:
Für n-stellige Prädikat p aus P mit

$$\begin{array}{lcl} p(\bar{t}_1) & \leftarrow & G_1 \\ \vdots & & \vdots \\ p(\bar{t}_m) & \leftarrow & G_m \end{array}$$

wird in $P^{\leftrightarrow}$ eingefügt

$$\begin{array}{lcll} p(\bar{x}) & \leftrightarrow & \exists \bar{y}_1 \ (\bar{t}_1 \doteq \bar{x} \wedge G_1) & \vee \\ & & \vdots & \vee \\ & & \exists \bar{y}_m \ (\bar{t}_m \doteq \bar{x} \wedge G_m) & \end{array}$$

$\bar{x}$ ist eine Folge der Länge n von paarweise verschiedenen, neuen Variablen, und $\bar{y}_i$ die Folge der Variablen in G_i und $\bar{t}_i$.
Für jedes andere Prädikatssymbol p, wird in $P^{\leftrightarrow}$ eingefügt:

$$\neg p(\bar{x})$$

Anmerkung

Clarksche Gleichheitstheorie CET

Definition 8 Sei Σ eine Signatur, die unendlich viele Funktionssymbole, davon mindestens eine Konstante, enthält. Die *Clarksche Gleichheitstheorie CET* ist:

Reflexivität:
$\forall(\top \rightarrow x \doteq x)$
Symmetrie:
$\forall(x \doteq y \rightarrow y \doteq x)$
Transitivität:
$\forall(x \doteq y \wedge y \doteq z \rightarrow x \doteq z)$
Verträglichkeit:
$\forall(x_1 \doteq y_1 \wedge \ldots \wedge x_n \doteq y_n \rightarrow f(x_1, \ldots, x_n) \doteq f(y_1, \ldots, y_n))$
Zerlegung:
$\forall(f(x_1, \ldots, x_n) \doteq f(y_1, \ldots, y_n) \rightarrow x_1 \doteq y_1 \wedge \ldots \wedge x_n \doteq y_n)$
Widerspruch:
$\forall(f(x_1, \ldots, x_n) \doteq g(y_1, \ldots, y_m) \rightarrow \bot)$ falls $f \neq g$ oder $n \neq m$
Azyklizität:
$\forall(x \doteq t \rightarrow \bot)$ falls t keine Variable ist und x in t vorkommt

Anmerkung

Damit das Gleichheitszeichen $\doteq$ richtig behandelt wird und nicht wie ein beliebiges binäres Prädikat, werden die Sätze, die die Gleichheit beschreiben (Clark'sche Gleichheitstheorie), zu $P^{\leftrightarrow}$ hinzugenommen.

Die ersten drei Sätze sagen aus, daß $\doteq$ eine Äquivalenzrelation ist.

Clark'sche Vervollständigung

Definition 9 Sei P ein Logikprogramm. Die *Clark'sche Vervollständigung* $\mathcal{P}$ von P ist die Formelmenge:

$$\mathcal{P} := CET \cup P^{\leftrightarrow}$$

$\lhd$

Anmerkung

Deklarative und Operationale Semantik

Theorem 1 Korrektheit und Vollständigkeit von erfolgreichen Ableitungen Sei P ein Logikprogramm, G ein Ziel und θ eine Substitution.

- Wenn θ eine berechnete Antwort von G ist, dann gilt
$$\mathcal{P} \models \forall G\theta$$
- Wenn $\mathcal{P} \models \forall G\theta$ gilt, dann gibt es eine berechnete Antwort σ von G, sodaß $\theta = \sigma\alpha$ gilt (d. h. die berechnete Antwort kann allgemeiner ausfallen).

Anmerkung

Die deklarative und die operationale Semantik einer Sprache sollen einander möglichst gleichen. Einerseits soll alles, was wir ableiten können, auch logisch aus dem Programm folgen (Korrektheit), andererseits soll alles, was folgt, auch ableitbar sein (Vollständigkeit).

Beweise siehe Buch.

Deklarative und Operationale Semantik

Definition 10 Eine Ableitung ist *fair*, wenn sie entweder scheitert oder wenn jedes Atom, das in der Ableitung vorkommt, nach endlich vielen Reduktionen ausgewählt wird. $\lhd$

Theorem 2 Korrektheit und Vollständigkeit von erfolglosen Ableitungen Sei P ein Logikprogramm und G ein Ziel. $\mathcal{P} \models \neg\exists G$ genau dann, wenn jede faire Ableitung beginnend mit $\langle G, \epsilon \rangle$ endlich scheitert.

Anmerkung

Beweise siehe Buch.

Negation durch Scheitern

Beispiel 2 Gegeben sei das Programm {`p(a)` $\leftarrow \top$} . Dann scheitert das Ziel `p(b)`. Daher ist das negierte Ziel $\neg$`p(b)` erfolgreich. Fügt man aber nun dem Programm das Faktum `p(b)` $\leftarrow \top$ hinzu, dann ist das Ziel `p(b)` erfolgreich und daher scheitert nun $\neg$`p(b)`. Es gibt also eine erfolgreiche Antwort weniger. $\diamond$

Anmerkung

Damit entfernt man sich jedoch beträchtlich von der Negation der klassischen Logik. Insbesondere zerstört Negation durch Scheitern die Eigenschaft des LP-Kalküls ohne Negation, daß erfolgreiche Ableitungen erhalten bleiben, wenn man dem Programm Klauseln hinzufügt.

Negation durch Scheitern

Beispiel 3 Gegeben sei das Programm {`p(a)` $\leftarrow \top$, `q(b)` $\leftarrow \top$ } . Das Ziel $\neg$`p(X)` $\wedge$ `q(X)` scheitert, wenn $\neg$`p(X)` zuerst zur Reduktion ausgewählt wird. Hingegen führt die sofortige Reduktion von `q(X)` zur Substitution `X` $\leftarrowtail$ `b` und damit ist das verbleibende Ziel $\neg$`p(X)` durch Negation durch Scheitern erfolgreich. $\diamond$

Anmerkung

Desweiteren kann Negation durch Scheitern zu Ableitungen mit einander widersprechenden Endzuständen führen, wenn das negierte Ziel ungebundene Variablen enthält.

Negation durch Scheitern

Erweiterte Syntax von Zielen

Ziele: $G, H ::= \top \mid \bot \mid A \mid \neg A \mid G \wedge H$

Reduktionen von negierten Atomen

Scheitern NaF:

$$\frac{\langle A, \theta\rangle \mapsto^* \langle \top, \alpha\rangle}{\langle \neg A \wedge G, \theta\rangle \mapsto \langle \bot, \epsilon\rangle}$$

Erfolg NaF:

Jede Ableitung scheitert endlich:

$$\frac{\langle A, \theta\rangle \mapsto^* \langle \bot, \epsilon\rangle}{\langle \neg A \wedge G, \theta\rangle \mapsto \langle G, \theta\rangle}$$

Anmerkung

Negation durch Scheitern

Beispiel 4 Sei $P = \{\mathtt{p} \leftarrow \neg\mathtt{p}\}$ ein Logikprogramm. Die Vervollständigung von P

$$\mathcal{P} = \{p \leftrightarrow \neg p\} \cup CET$$

ist eine inkonsistente Formelmenge. $\diamond$

Anmerkung

Ursprünglich konsistente Formelmengen können durch die Vervollständigung in inkonsistente Mengen überführt werden.

Negation durch Scheitern

Beispiel 5 Um das Ziel ¬`verbindung(X,Y)` zu bearbeiten, müssen wir `verbindung(X,Y)` berechnen. Dieses Ziel ist erfolgreich, eine Antwort ist z.B. $\{\texttt{X} \leftarrow\!\shortmid \texttt{a}, \texttt{Y} \leftarrow\!\shortmid \texttt{b}\}$. Daher scheitert das Ziel ¬`verbindung(X,Y)` endlich, d.h. es ist nicht der Fall, daß es keine Verbindung zwischen zwei beliebigen Knoten gibt. Es gibt aber z.B. keine Verbindung von `b` nach `a`. ◇

Anmerkung

Wir zeigen an einem Beispiel, daß Ziele mit Variablen zu einem Widerspruch mit der deklarativen Semantik führen können.

	Entwicklung von Prolog
um 1964	J. A. Robinson, Resolutionsprinzip
um 1972	A. Colmerauer, Marseille, spezialisierter Thereombeweiser für natürlichsprachiges Verstehen, genannt Prolog
um 1972	R. A. Kowalski, Edingburgh, prozedurale Interpretation von Hornklauseln
1975	D. H. D. Warren, London, erste effiziente Implementierung erster Compiler, fast vollständig in Prolog, WAM (Warren Abstract Machine)
1982-1994	Fifth Generation Computing Project, Japan

Anmerkung

Colermauer, Kowalski, FCGS Projekt

Künstliche Intelligenz, Expertenssysteme

Microsoft Windows NT Netzwerkkonfiguration

Knowledgewares Application Development Workbench (60.000 Lizenezen 1995) 250.000 Zeilen prolog, in C 10* soviel.

FCGS scheitert, CLP, Prolog Konkurse

	Entwicklung von Prolog, Fortsetzung
1984-1996	LP am European Computer-Industry Research Center (ECRC), München, mit Schwerpunkt Logikprogrammierung ECL^{i}PS^e Prolog mit etwa 300 Lizenzen weltweit
seit 1985	LP am Swedish Insitute of Computer Science (SICS) Sicstus Prolog mit etwa 500 Lizenzen weltweit
80er Jahre	EG Esprit-Projekte im Bereich LP
80er Jahre	Borlands Turbo-Prolog, Pascalähnliche, Prologartige Sprache
1996	ISO Prolog Standard.

Anmerkung

Die konkrete Syntax „:-\, „,\'.'

SLDNF-Resolution („**L**inear resolution with **S**election function for **D**efinite clauses with **N**egation as **F**ailure“)

Nichtdeterminismus (Auswahl von Klauseln und Atomen) nach einem festen Schema auflöst und Negation durch Scheitern nur für (zur Laufzeit) variablenfreie Ziele.

Das am weitesten links stehende Literal eines Zieles selektiert. Klauseln in textueller Reihenfolge ausgewählt: Scheitert die Ableitung mit einer bestimmten Klausel, versucht man durch Rücksetzen (engl. backtracking) (chronologisches Rücksetzen).

Die Selektionsregel für das Literal zusammen mit dem chronologischen Rücksetzen führt dazu, daß eine Ableitung vollständig berechnet wird, bevor Alternativen untersucht werden. Tiefensuche. Unvollständig: erste berechnete Ableitung unendlich, sodaß andere (erfolgreiche) Ableitungen nicht mehr gefunden werden können.

Programmierer durch die Reihenfolge der Klauseln und Literale den Programmablauf (die Steuerung) beeinflussen kann.

Multidirektionalität, Invertierbarkeit vs. Built-Ins, Funktionen