# Die Behandlung graphischer Seiteneffekte beim Backtracking in Prolog[1]

Rainer Hammwöhner Ulrich Thiel

Universität Konstanz

Informationswissenschaft

## 1. Einleitung

Im Rahmen des Projekts TOPOGRAPHIC wird zur Zeit am Lehrstuhl für Informationswissenschaft der Universität Konstanz ein Prototyp für ein wissensbasiertes Informationssystem mit graphischem Interface entwickelt. Der experimentelle Charakter dieses Systems erfordert den Einsatz einer Spezifikationssprache, die eine inkrementelle Entwicklung im Sinne des Rapid Prototyping ermöglicht. Die Entscheidung, dieser Spezifikationssprache eine um Graphik- und Wissensbasiszugriffsfunktionen erweiterte Prolog-Version zugrunde zu legen, gründet sich auf folgende Gesichtspunkte:

- Prolog eignet sich sehr gut als Datenbankabfragesprache (z.B. Futo et al. 1978)
- Der Aufbau von Bildern läßt sich auf Datenbankoperationen zurückfuhren, mit denen die Repräsentation des Bildes in der Datenbank manipuliert wird. (Pereira 1986)

In diesem Beitrag wollen wir uns mit einigen Aspekten der Einbettung von Graphik in Prolog auseinandersetzen. Standard-Prolog (Clocksin/Mellish 1984) hat bezüglich der Ein-/ Ausgabe nur einen minimalen Funktionsumfang, der sich trotzdem nur recht unglücklich in das Konzept von Prolog als logischer Programmiersprache einfügt (Clocksin 1984). Im ersten Abschnitt dieses Beitrags werden wir deshalb Verbesserungen für einige Konstrukte in Standard-Prolog vorschlagen, die das Verhalten beim Backtracking betreffen. Das dabei angewandte Verfahren wird dann auf das Problem der Einbettung von Graphik in Prolog übertragen. Grundannahme bei diesen Überlegungen ist, daß eine Erweiterung des Funktionsumfangs von Prolog angestrebt wird, jedoch nicht eine Veränderung an den Resolutions- oder Unifikationsalgorithmen. Ein graphikfähiges Prolog, wie es hier im Bezug auf das Verhalten beim Backtracking diskutiert wird, kann also mit allen Prolog-Versionen, die über eine Schnittstelle zu einer prozeduralen Programmiersprache verfugen, erstellt werden. Anschließend werden die bis dahin pragmatisch begründeten Vorschläge abgesichert, indem das graphikfähige Prolog - zunächst nur informal - als ein modallogisches System reinterpretiert wird. Ein kurzer Ausblick auf weitergehende Fragestellungen wird zum Abschluß gegeben.

---

[1] Dieser Text ist erschienen in: Hommel, G. / Schindler, S. (ed): Proc. GI- 16. Jahrestagung I, Berlin, Heidelberg, 1986, pp. 313-321.

## 2. Seiteneffekte in Standardprolog

Als außerlogische Operatoren werden wir diejenigen Standard-"Prädikate" bezeichnen, deren Auswertung zu nicht monotonem Verhalten führt. Dies sind insbesondere die Operatoren zur Manipulation der Datenbasis (asserta, assertz, retract) und zur Ein-/Ausgabe (tell,told, read, write, etc.). Die von diesen Operatoren bewirkten Zustandsänderungen der Datenbasis bezeichnen wir als Seiteneffekte.

Hierzu ein Beispiel :

```
prädikat_1:-  bedingung_1,
              asserta( fakt_1 (x)),
              bedingung_2.

prädikat_2:-  fakt_1(x).

test :- prädikat_1; prädikat_2.
```

Das Ergebnis von 'test' ist nicht nur davon abhängig, ob 'prädikat_1' bewiesen werden kann, sondern auch davon, ob wenigstens das 'asserta' erreicht wird, da die hier spezifizierte Änderung der Datenbasis von potentiellem Backtracking nicht berührt wird. Operatoren, die im Falle des Backtrackings ihre Seiteneffekte zurücksetzen, lassen sich in Prolog folgendermaßen definieren:

```
asserta*(TERM)     :- asserta(TERM); retract (TERM), fail.

assertz*(TERM)     :- assertz(TERM); retract(TERM), fail.

retract*(TERM)     :- retract(TERM); asserta(TERM), fail.
```

Das heißt, im Fall des Backtracking wird der 'oder-Zweig' der Regel durchlaufen und damit der ursprünglich gesetzte Seiteneffekt annulliert. Das 'fail' verhindert, daß dieser Zweig der Regel einen neuen Lösungsweg für den Beweis bereitstellt. Während der Zustand der Datenbasis im Fall von asserta* und assertz* durch Backtracking exakt wieder hergestellt wird, wird durch retract* nur ein deklarativ äquivalenter Zustand erreicht, da die Reihenfolge der in der Datenbasis befindlichen Formeln verändert sein kann. Gravierende Änderungen im Systemverhalten können dann auftreten, wenn das Argument von retract* eine Regel ist. Deshalb sollte dieser Operator entweder nur für atomare Formeln zugelassen werden, oder so erweitert werden, daß der Ursprungszustand auch prozedural äquivalent wiederhergestellt wird.

Ein weiteres Problem tritt auf, wenn die Argumentterme der außer logischen Operatoren nicht unifizierte Variablen enthalten, denn dann wird durch den Seiteneffekt eine implizite Änderung der Variablenquantifizierung bewirkt.

Beispiel:

```
goal(X) :-asserta(fact(X)) , X='abc'.
```

das entspricht: $\forall$X (goal(X) $\leftarrow$ asserta(fact(X)), X='abc'

und damit: $\forall$ X (fact(X))

Dieses Problem werden wir im Verlauf des Vertrags wieder aufgreifen, fuhren aber zunächst Graphik-Prolog als eine Erweiterung des Standard-Prolog (Clocksin/Melish 1984) ein.

## 3. Graphik-Prolog

Graphikfähiges Prolog entsteht aus Standard-Prolog durch die Definition einer Menge von Graphik-Prädikaten und einer dualen Menge von Graphik-Operatoren. Sowohl die Prädikate, wie auch die Operatoren beziehen sich auf graphische Objekte, wobei komplexe Objekte aus elementaren zusammengesetzt werden können, die durchaus nicht mit den atomaren Bausteinen des Darstellungsmediums (z.B. Bildpunkte) identisch sein müssen, sondern Polygonzüge, Textfragmente etc. darstellen können. Aussagen über die Merkmale bereits existierender Objekte und damit über das aktuell dargestellte Bild können mit Hilfe graphischer Prädikate formuliert werden, während die dualen Operatoren als Seiteneffekt Objekte mit vorgegebenen Merkmalen erzeugen. Diese Operatoren werden durch Unterprogramme einer prozeduralen Programmiersprache implementiert und als eingebaute Prädikate in Prolog eingebettet. Sie sind in ihrem Verhalten während des Beweisvorgangs analog zu den Prädikaten zur Datenbasismanipulation zu verstehen, nur daß sie sich nicht auf Formeln der Datenbasis beziehen, sondern auf graphische Objekte. Welche Auswirkungen die unterschiedliche Behandlung von Seiteneffekten auf die Formulierung von Regeln in Graphik-Prolog hat, soll nun an einem kleinen Beispiel erläutert werden, dem die in (Clocksin/Mellish 1984) angeführte rekursive Regel zum Durchwandern eines Netzes zugrunde liegt. Dabei ist zu beachten, daß die inversen Operatoren, die beim Backtracking die Seiteneffekte beseitigen, in der -Regel mehr Statusinformationen benötigen als aus der Argumentenliste ersichtlich ist. Ein Operator zum Löschen graphischer Objekte z.B. muß den kompletten Zustand eines Objekts mit allen seinen Teilobjekten sichern, damit daß Objekt beim Backtracking wiederhergestellt werden kann. Zunächst wird eine minimale Graphik-Erweiterung von Prolog definiert. Es seien drei Typen graphischer Objekte gegeben: Netze, Knoten und Kanten, wobei sich Netze aus durch Kanten verbundenen Knoten zusammensetzen. Für die elementaren Objekte sind folgende Prädikate/Operatoren definiert:

| Prädikat | Operator | Bedeutung |
|---|---|---|
| node(Id,X,Y,Color) | create_node(Id,X,Y,C) | Der Knoten Id befindet sich an der Position X,Y und hat die Farbe C |
| connects(Id,N1,N2) | connect(Id,N1 ,N2) | Die Kante Id verbidet die Knoten N1 und N2 |
| color(Id,C) | paint(Id,C) | Der Knoten Id hat die Farbe C |

Nun soll folgende Aufgabe gelöst werden: Ein Netz sei auf dem Bildschirm dargestellt. Es gilt nun einen Weg zwischen zwei Knoten zu finden, wobei die Bewegung entlang der Kanten erfolgt. Der gefundene Weg soll durch Knoten mit verändertem Darstellungsmodus gekennzeichnet werden. Der Lösungsansatz beruht darauf, daß von einem vorgegebenen Knoten ausgehend ein erreichbarer Nachbarknoten aufgesucht wird. War dieser Knoten zuvor noch nicht erreicht worden - das wird mit Hilfe einer Liste festgestellt -, so wird er als neuer Ausgangspunkt betrachtet.

Variante 1 : Seiteneffekte werden beim Backtracking nicht beseitigt, deshalb kann der gesuchte Weg erst nach vollständiger Abarbeitung der Rekursion in die Graphik eingetragen werden, es müssen also zwei Rekursionen abgearbeitet werden.

```
go(Node,Node, Liste) :-    update_graphics(Liste).

go(Node1 ,Node2,Liste) :- is_related(Node1,NodeX),
                          not (member(NodeX,Liste)),
                          go(NodeX,Nöde2, IJStodeX|Liste]).
```

```
update_graphics ([ ]).

update_graphics([Node | Liste]):- paint(Node,black),
                                  update_graphics (Liste).
```

Variante 2 : Die Seiteneffekte werden beim Backtracking rückgängig gemacht, deshalb kann die graphische Markierung gefundener Knoten direkt in die Regel eingebracht werden, wodurch man zu einer vereinfachten Formulierung kommt. Überdies ist diese Regelversion in der Lage durch Backtracking weitere Lösungsvarianten zu erzeugen, was bei Variante 1 nicht ohne weitere Verkomplizierung möglich ist.

```
paint*(Ob;j,Color):-    node(Obj,X,Y,C),
                        (paint(Obj,Color);
                        paint(Obj,C),fail).

go(Node, Node, Liste).

go(Node1 ,Node2,Liste):- is_related(Node1,NodeX),
                         not(member(NodeX,Liste)),
                         paint* (NodeX, inverted),
                         go(NodeX,Nbde2, [NodeX|Liste]).
```

Für beide Varianten gilt:

```
is_related(Node1 ,Node2):- connects(X,Node1 ,Node2).
```

Die beiden Abbildungen auf der folgenden Seite zeigen eine beispielhafte Anwendung der Regelversion 2 auf ein einfaches Netz. Gesucht wurde ein Weg zwischen den Knoten 2 und 7- Durch anschließendes resatisfy wurde eine zweite Lösung gefunden.

## 4. Graphische Operatoren als Modaloperatoren

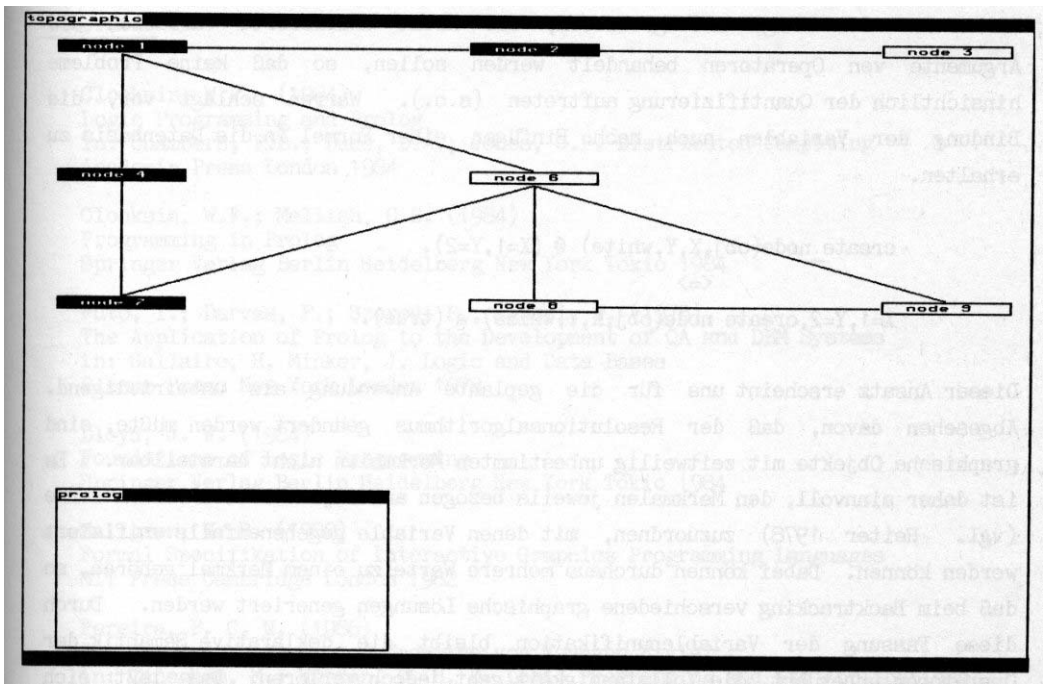
In (Warren 1984) wird eine modallogische Interpretation von Änderungen der Datenbasis gegeben, die allerdings auf das Hinzufügen oder Löschen atomarer Formeln beschränkt ist. Da die Repräsentation von Graphik extensional durch die Menge der dargestellten Objekte erfolgen kann, erscheint diese Einschränkung als nicht gravierend, so daß dieser Ansatz - abgesehen von der Behandlung nicht unifizierter Variablen (s.u.) - unverändert übernommen werden kann. Eine exakte Definition von Syntax und Semantik der erweiterten Prolog-Version wird an dieser Stelle nicht gegeben, da sie aufgrund der logischen Äquivalenz - nur die Seiteneffekte unterscheiden sich - nur zu einer Rekonstruktion fuhren würde. Deshalb soll eine informelle Einführung anhand einiger Beispiele erfolgen. Außerlogische Operatoren werden nicht mehr in Analogie zu Prädikaten gesehen, sondern als Modaloperatoren, die auf die in einer Klause rechts von ihnen stehenden Prädikate angewandt werden. Um diese mehrdeutige Verwendung des ','-Junktors zu vermeiden, wurde eine modifizierte Schreibweise eingeführt, die für die Anwendung eines Operators auf eine Formel ein eigenes Symbol vorsieht

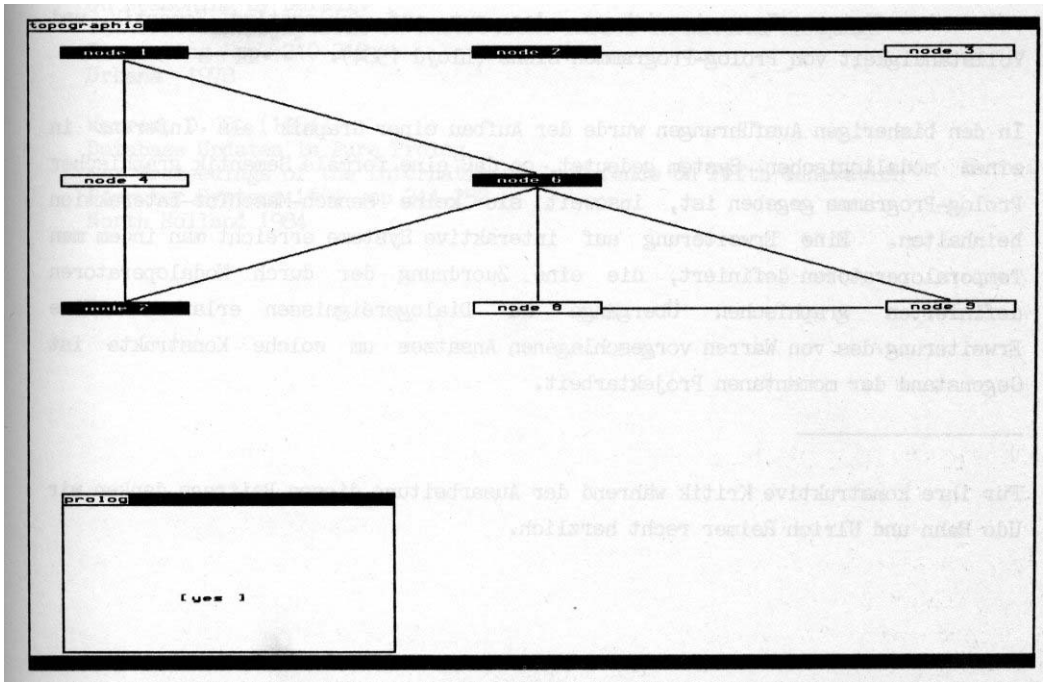
```
create_node(obj, 1,1,white),true. ≡ create_node (obj, 1,1, white) @ (true).
```

Diese Formel ist erfüllt, wenn das aktuelle Bild um den Knoten 'obj' erweitert werden kann.

```
create_and_connect(Obj,X,Y,C):- create_node(Obj,X,Y,C) @
                                (node(Obj2,X2,Y2,C2),
                                connect(rel,0bj,0bj2) @ (C2==white)).
```

Durch diese Formel wird ein neuer Operator definiert, der einen Knoten erzeugt und ihn durch eine Kante mit einem beliebigen weißen Knoten verbindet.




Bisher war die Frage offengeblieben, wie nicht unifizierte Variablen als Argumente von Operatoren behandelt werden sollen, so daß keine Probleme hinsichtlich der Quantifizierung auftreten (s.o.). Warren schlägt vor, die Bindung der Variablen auch nach Einfügen einer Formel in die Datenbasis zu erhalten.

create_node(obj,X,Y,white) @ (X=1,Y=2).

⇔

X= 1,Y=2, create_node(obj,X,Y,white) @ (true).

Dieser Ansatz erscheint uns für die geplante Anwendung als unbefriedigend. Abgesehen davon, daß der Resolutionsalgorithmus geändert werden müßte, sind graphische Objekte mit zeitweilig unbestimmten Merkmalen nicht darstellbar. Es ist daher sinnvoll, den Merkmalen

jeweils bezogen auf Objektklassen Defaultwerte (vgl. Reiter 1978) zuzuordnen, mit denen Variable gegebenenfalls unifiziert werden können. Dabei können durchaus mehrere Werte zu einem Merkmal gehören, so daß beim Backtracking verschiedene graphische Lösungen generiert werden. Durch diese Fassung der Variablenunifikation bleibt die deklarative Semantik der Operatoren unberührt, die Vollständigkeit geht jedoch verloren. Das läßt sich darlegen, indem man sich die möglichen Lösungen durch Fakten dargestellt denkt. Die Defaultwerte seien die ersten n dieser Fakten und beim n-ten Fakt sei ein cut eingefügt. Zur Auswirkung des cut auf deklarative Semantik und Vollständigkeit von Prolog-Programmen siehe (Lloyd 1984).

In den bisherigen Ausführungen wurde der Aufbau einer Graphik als Inferenz in einem modallogischen System gedeutet, so daß eine formale Semantik graphischer Prolog-Programme gegeben ist, insoweit sie keine Mensch-Maschine-Interaktion beinhalten. Eine Erweiterung auf interaktive Systeme erreicht man, indem man Temporaloperatoren definiert, die eine Zuordnung der durch Modaloperator definierten graphischen Übergänge zu Dialogereignissen erlauben. Eine Erweiterung des von Warren vorgeschlagenen Ansatzes um solche Konstrukte ist Gegenstand der momentanen Projektarbeit.

Für ihre konstruktive Kritik während der Ausarbeitung dieses Beitrags danken wir Udo Hahn und Ulrich Reimer recht herzlich.

## Literatur

Clocksin, W.P. (1984) Logic Programming and Prolog. In: Chambers, F.B.; Duce, D.A.; Jones, G.P. Distributed Computing, Academic Press, London.

Clocksin, W.F.; Mellish, C.S. (1984) Programming. In Prolog, Springer Verlag, Berlin, Heidelberg, New York, Tokio.

Futo, I.; Darvas, F.; Szeredi P.; Igüszi, N. (1978) The Application of Prolog to the Developnent of QA and DBM Systems. In: Gallaire, H. Minker, J. Logic and Data Bases, Plenum Press New York London.

Lloyd, J. W. (1984) Foundations of Logic Programming, Springer Verlag, Berlin, Heidelberg, New York, Tokio.

Mallgren, W. R. (1982) Formal Specification of Interactive Graphics Programming Languages, MIT Press, Cambridge, London.

Pereira, F. C. N. (1986) Can Drawing be Liberated from the von Neumann Style? In: Caneghem, M.; Warren, D. H. D. Logic Programming and its Applications, Ablex Publishing Corporation, Norwood, New Jersey, pp 175-187.

Reiter, R. (1978) On Reasoning by Default. In: Proc. 2nd Symposium on Theoretical Issues in Natural Language Processing, Urbana, pp. 210-218.

Warren, D. S. (1984) Database Updates in Pure Prolog. In: Proceedings of the International Conference on Fifth Generation Computer Systems, North Holland, pp 244-252.