View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Digitale Bibliothek Thüringen

ThILLM
Thüringer Institut für Lehrerfortbildung, Lehrplanentwicklung und Medien

LOGIK-ORIENTIERTES PROGRAMMIEREN MIT PROLOG – UNTERRICHT, ÜBUNGEN UND ABITUR

Otto Thiele, Jena
Manfred Kämmerer, Jena
Jürgen Helbig, Heiligenstadt
Bernd Merten, Ohrdruf

2001

Heft 63 MATERIALIEN

Die Reihe „Materialien" wird vom Thüringer Institut für Lehrerfortbildung, Lehrplanentwicklung und Medien im Auftrag des Thüringer Kultusministeriums herausgegeben, sie stellt jedoch keine verbindliche, amtliche Verlautbarung des Kultusministeriums dar.

2001
ISSN: 0944-8705
Herausgeber:
Thüringer Institut für Lehrerfortbildung,
Lehrplanentwicklung und Medien, ThILLM Bad Berka
Heinrich-Heine-Allee 2-4
99438 Bad Berka
Telefon: 03 64 58/56-0
Telefax: 03 64 58/56-300
institut@thillm.thueringen.de
http://www.thillm.de

Redaktion:	Dr. Wolfgang Moldenhauer, ThILLM
Inhalt:	Otto Thiele, Jena, Manfred Kämmerer, Jena, Jürgen Helbig, Heiligenstadt, Bernd Merten, Ohrdruf
Grafiken:	Manfred Kämmerer, Jena
Titelei:	Satzstudio Nußbaum, Alach
Druck:	gb • druckerei, Arnstadt

Dem Freistaat Thüringen, vertreten durch das ThILLM, sind alle Rechte der Veröffentlichung, Verbreitung, Übersetzung und auch die Einspeicherung und Ausgabe in Datenbanken vorbehalten. Die Herstellung von Kopien in Auszügen zur Verwendung an Thüringer Bildungseinrichtungen, insbesondere für Unterrichtszwecke, ist gestattet.

Diese Publikation wurde im Juli 2006 orthografisch an die amtliche Neuregelung der deutschen Rechtschreibung nach dem Stand vom März 2006 angepasst.

Inhaltsverzeichnis

Vorwort 5

1 Einblick in das logikorientierte Programmieren

1.1. Fakten, Regeln und Prädikate 7
1.2. Relationale Datenbanken in Prolog 14
1.3 Rekursion 17
1.4 Listen 21
1.5 Wegsuche in Graphen 24
1.6 Auskunftssystem 31
1.7 Manipulation symbolischer Ausdrücke 35
1.8 Dynamisches Verändern der Wissensbasis 37
1.9 Parser und Interpreter 39
1.9.1 Parser 39
1.9.2 Interpreter 43
1.10 Ein kleines Expertensystem 49
1.11 Resolutionsprinzip, Unifikation und Backtracking 53
1.11.1 Resolutionsprinzip 53
1.11.2 Unifikation 57
1.11.3 Backtracking 58

2. Aufgaben aus dem Unterricht

2.1. Familienstammbäume 62
2.1.1 Karls Stammbaum 62
2.1.2 Auszug aus dem Stammbaum der Musikerfamilie Bach 62
2.2 Fakten Regeln Prädikate 63
2.3 Hinter sieben Bergen 63
2.4 Weimarer Dichterkreis 65
2.5 Thüringens Geographie 66
2.6 Expertensystem 67
2.7 Wanderungen 69
2.8 Ampelmännchen 69
2.9 Sehenswürdigkeiten von Erfurt 70
2.10 Gartenzwerge 71
2.11 Rekursion 72
2.12 Das ist das Haus vom Nikolaus 72
2.13 Rundreise 73
2.14 Brücken von Paris 74
2.15 Kryptogramm 74
2.16 Bibliothek 76
2.17 Sehen oder nicht sehen 77
2.18 Binärer Baum 77

3. Aufgaben aus Abiturprüfungen

Grundfach 1994 79
Leistungsfach 1994 80
Grundfach 1995 81
Leistungsfach 1995 82
Grundfach 1996 82
Leistungsfach 1996 83
Grundfach 1997 84
Leistungsfach 1997 84
Grundfach 1998 85
Leistungsfach 1998 86
Nachtermin Leistungsfach 1998 87
Grundfach 1999 88
Leistungsfach 1999 89
Grundfach 2000 89
Leistungsfach 2000 90
Leistungsfach 2001 91

4. Lösungsvorschläge zu ausgewählten Abituraufgaben

Grundfach 1994 92
Grundfach 1995 92
Grundfach 1996 93
Leistungsfach 1996 94
Grundfach 1997 94
Grundfach 1998 95
Nachtermin Leistungsfach 1998 96
Grundfach 1999 97
Grundfach 2000 97
Leistungsfach 2000 97

Anhang 1

Kopiervorlagen 99

Anhang 2

Literaturverzeichnis und Quellenangaben 118

Vorwort

Die vorliegende Publikation gibt einen Einblick in die Entwicklung des Faches Informatik an den Thüringer Gymnasien, Kollegs und Gesamtschulen. Diese Entwicklung hat mit der Einführung des Wahlfachs und Wahlunterrichts Informatik einen ersten Abschluss gefunden (www.medienkunde.de).

Die Autoren, Herr Jürgen Helbig, Herr Manfred Kämmerer, Herr Bernd Merten und Herr Otto Thiele, waren seit Anfang der 90-er Jahre an dieser Entwicklung beteiligt, so unter anderem als Mitglieder der Landesfachkommission Informatik (www.th.schule.de/th/lfk-informatik, www.lfk-informatik). Mit dieser Veröffentlichung leisten die Autoren einen weiteren wichtigen Beitrag zur unterrichtlichen Weiterentwicklung.

Nach dem Thüringer Informatiklehrplan (1999) ist das Programmieren mit PROLOG ein Wahlthema im Grundfach und ein Pflichtthema im Leistungsfach. Damit lernen die Schülerinnen und Schüler ein weiteres Paradigma, das logikorientierte Programmieren, kennen.
Diese Publikation soll Lehrerinnen und Lehrer unterstützen, die PROLOG im Unterricht als Werkzeug einsetzen. Dazu werden zunächst die nötigen Grundlagen zusammengestellt. Die Autoren berücksichtigen dabei, dass PROLOG seit 1991 unterrichtet wird und daher Erfahrungen vorliegen. Aufgaben aus und damit für den Unterricht konkretisieren die Grundlagen. Der Entwicklung sprachverarbeitender Systeme kommt nach dem Lehrplan eine besondere Bedeutung zu. Hierzu werden erste Beispiele angeführt, die noch erweitert werden müssen.
Die Thüringer Abituraufgaben der Jahre 1994 bis 2001 zu PROLOG wurden zusammengestellt und exemplarisch "gelöst". Kopiervorlagen helfen, die praktische Arbeit der Lehrerinnen und Lehrer zu unterstützen.
Die Beispiele und Aufgaben besitzen häufig Bezüge zum Thüringer Land und zu anderen Unterrichtsfächern und regen damit zum fächerübergreifenden Arbeiten an.

Teilen Sie uns mit, welche Erfahrungen Sie mit den Materialien gesammelt haben, damit wir Ihre Hinweise bei der weiteren Entwicklung berücksichtigen können.

Bernd Schreier
Direktor des ThILLM

Dr. Wolfgang Moldenhauer
Referent für Informatik

1 Einblick in das logikorientierte Programmieren

Das Lösen von Problemen mithilfe der logikorientierten Programmiersprache PROLOG, entwickelt 1973 von ALAIN COMERAUER, besitzt im Grund- und Leistungsfach Informatik der gymnasialen Oberstufe in Thüringen eine gute Tradition. Im Themenbereich »Einblick in das logikorientierte Programmieren« werden die Schülerinnen und Schüler der Klassenstufe 12 mit der bislang im Informatikunterricht nicht gewohnten deskriptiven Denkweise vertraut gemacht. Zum Lösen eines Problems ist Wissen derart zu spezifizieren, dass es in Form von Fakten und Regeln in der Wissensbasis eines PROLOG-Programms implementiert werden kann. Nach dem Einlesen korrekter Anfragen gibt das PROLOG-System Antworten aus, die durch die Suche in der Wissensbasis gefunden werden.

Die Erfahrung der Autoren zeigt, dass die Schülerinnen und Schüler, die bisher die algorithmische Denkweise beim Lösen von Problemen gewohnt sind, unkompliziert an die neue Denkweise herangeführt werden können. In den folgenden Kapiteln werden aus dieser Sicht erprobte, didaktisch aufbereitete Inhalte für die Gestaltung des Unterrichts im Grund- und Leistungsfach dargelegt.

1.1 Fakten, Regeln und Prädikate

Der folgende Text enthält Fakten zur Familie des Dichters JOHANN WOLFGANG VON GOETHE (1749-1832):

> Die männlichen Mitglieder der Familie Goethe sind Johann Kaspar, Johann Wolfgang, August, Wolfgang und Walter. Katharina Elisabeth, Kornelia, Christiane, Ottilie und Alma sind die weiblichen Familienmitglieder. Johann Kaspar und Katharina Elisabeth, Johann Wolfgang und Christiane sowie August und Ottilie sind miteinander verheiratet. Johann Wolfgang und Kornelia sind die Kinder von Johann Kaspar und Katharina Elisabeth. August ist das Kind von Johann Wolfgang und Christiane. Wolfgang, Walter und Alma sind die Kinder von August und Ottilie.

Alle Fakten aus dem Text können in ein PROLOG-Programm übertragen werden.

```
% Programm Familie Goethe

ist_maennlich(johann_kaspar).
ist_maennlich(johann_wolfgang).
ist_maennlich(august).
ist_maennlich(wolfgang).
ist_maennlich(walter).

ist_weiblich(katharina_elisabeth).
ist_weiblich(kornelia).
ist_weiblich(christiane).
ist_weiblich(ottilie).
ist_weiblich(alma).
```

```
ist_verheiratet_mit(johann_kaspar,katharina_elisabeth).
ist_verheiratet_mit(johann_wolfgang,christiane).
ist_verheiratet_mit(august,ottilie).

ist_kind_von(johann_wolfgang,johann_kaspar).
ist_kind_von(kornelia,johann_kaspar).
ist_kind_von(johann_wolfgang,katharina_elisabeth).
ist_kind_von(kornelia,katharina_elisabeth).
ist_kind_von(august,johann_wolfgang).
ist_kind_von(august,christiane).
ist_kind_von(wolfgang,august).
ist_kind_von(walter,august).
ist_kind_von(alma,august).
ist_kind_von(wolfgang,ottilie).
ist_kind_von(walter,ottilie).
ist_kind_von(alma,ottilie).
```

Das Programm besteht aus vier Prädikaten:

Alle Mitglieder der Familie Goethe, die die Eigenschaft besitzen männlich zu sein, werden durch das Prädikat *ist_maennlich/1* angegeben. Dieses Prädikat ist einstellig und besteht aus fünf Fakten. Die weiblichen Familienmitglieder werden durch das Prädikat *ist_weiblich/1* repräsentiert. Es ist ebenfalls einstellig und besteht aus fünf Fakten.

Das Prädikat *ist_verheiratet_mit/2* beschreibt, besser induziert oder schließt ein, welche männliche Person mit welcher weiblichen Person verheiratet ist. Es ist zweistellig und besteht aus drei Fakten. Beispielsweise wird das Faktum

```
ist_verheiratet_mit(johann_kaspar,katharina_elisabeth).
```

derart gelesen: »Johann Kaspar ist mit Katharina Elisabeth verheiratet«. Es bleibt zu klären, wie im PROLOG-Programm zum Ausdruck gebracht wird, dass natürlich Katharina Elisabeth ebenfalls mit Johann Kaspar verheiratet ist.

Das zweistellige Prädikat *ist_kind_von/2*, schließt die Beziehung ein, wer in der Familie Goethe Kind von wem ist und besteht aus zwölf Fakten. Das Faktum

```
ist_kind_von(wolfgang,ottilie).
```

wird »Wolfgang, ist Kind von Ottilie« gelesen.

An das Programm können zum Beispiel folgende Anfragen gestellt werden:

```
?- ist_maennlich(johann_wolfgang).
?- ist_maennlich(friedrich).
?- ist_weiblich(Wer).
?- ist_verheiratet_mit(johann_wolfgang,Wem).
?- ist_verheiratet_mit(ottilie,Wem).
?- ist_kind_von(Wer,Wem).
```

Auf die erste Anfrage, ob Johann Wolfgang männlich ist, antwortet das PROLOG-System `Yes`. `No` antwortet das PROLOG-System auf die zweite Anfrage, da im Prädikat ist_maennlich/1 kein Faktum Friedrich ist männlich enthalten ist.

In der dritten Anfrage steht die Variable Wer. Es ist zu beachten, dass in PROLOG jeder Variablenbezeichner mit einem Großbuchstaben beginnen muss. Die Antworten des PROLOG-Systems auf die dritte Anfrage lauten:

```
Wer = katharina_elisabeth;
Wer = kornelia;
Wer = christiane;
Wer = ottilie;
Wer = alma;
No
```

Findet das PROLOG-System keine weiteren Antworten, wird `No` ausgegeben.

Das PROLOG-System antwortet auf die vierte Anfrage:

```
Wem = christiane;
No
```

Die fünfte Anfrage wird mit `No` beantwortet, weil im Prädikat ist_verheiratet_mit/2 zuerst der männliche und danach der weiblichen Ehepartner genannt wird (fehlerhafte Anfrage).

Auf die sechste Anfrage, die die Variablen Wer und Wem enthält, gibt das PROLOG-System als erstes Antwortpaar

```
Wer = august, Wem = johann_wolfgang
```

elf weitere Antwortpaare und `No` aus.

Der Programmierer trägt eine besondere Verantwortung für die inhaltliche Richtigkeit eines PROLOG-Programms. Angenommen das Programm Familie würde um das folgende Faktum

```
ist_maennlich(kornelia).
```

erweitert, würde das PROLOG-System auf die Anfrage, ob Kornelia männlich ist, mit `Yes` antworten.

In mehrstelligen Prädikaten ist die Reihenfolge der Argumente von Bedeutung. So wäre beispielsweise das Faktum

```
ist_kind_von(christiane,august).
```

syntaktisch korrekt, inhaltlich jedoch falsch, da Christiane nicht das Kind von August ist.

Aus den im Text enthaltenen Fakten zur Familie Goethe lassen sich weitere Beziehungen folgern. Beispielsweise ist Johann Kaspar der Vater von Johann Wolfgang und Kornelia. Johann Wolfgang ist Vater von August. August ist Vater von Wolfgang, Walter und Alma. Diese Beziehung lässt sich aus den im PROLOG-Programm stehenden Fakten folgern, wenn das Programm um das Prädikat ist_vater_von/2 erweitert wird.

```
% Erweiterung des Programms Familie Goethe

  ist_vater_von(V,K):-  ist_maennlich(V),
                        ist_kind_von(K,V).
```

Dieses zweistellige Prädikat besteht aus einer Regel zum Folgern der Beziehung, wer in der Familie Goethe Vater von wem ist. V und K sind die Variablen.

Die Regel wird wie folgt gelesen: V ist der Vater von K, falls V männlich ist und K das Kind von V ist.

Das Prädikat ist_mutter_von/2 besteht ebenfalls aus einer Regel.

```
ist_mutter_von(M,K):- ist_weiblich(M),
                      ist_kind_von(K,M).
```

Komplexer ist das Prädikat ist_bruder_von/2, das zweistellig ist und die Beziehung angibt, wer in der Familie Goethe Bruder von wem ist.
Es besteht aus einer Regel.

```
ist_bruder_von(B,G):- ist_maennlich(B),
                      ist_vater_von(V,B),
                      ist_vater_von(V,G),
                      ist_mutter_von(M,B),
                      ist_mutter_von(M,G),
                      B\==G.
```

Zu beachten ist, dass B ungleich G sein muss, da niemand sein eigener Bruder ist.

Dem Programm kann das Prädikat sind_verheiratet/2

```
sind_verheiratet(X,Y):- ist_verheiratet_mit(X,Y);
                        ist_verheiratet_mit(Y,X).
```

hinzugefügt werden, das wie folgt gelesen wird: »X und Y sind verheiratet«.
Auf die Anfragen

```
?- sind_verheiratet(ottilie,august).
?- sind_verheiratet(august,otthilie).
```

antwortet das PROLOG-System jeweils `Yes`.

Auch folgende Prädikate stellen Erweiterungen des Programms dar:

ist_schwester_von/2, *ist_grossmutter_von/2*, *ist_grossvater_von/2* und *ist_enkel_von/2*.

Interessant ist, wie das PROLOG-System Antworten auf eine Anfrage sucht. Exemplarisch sei dazu die Anfrage

```
?- ist_vater_von(Wer,Wem).
```

gewählt. In Abbildung 1.1.1 ist der Baum dargestellt, den das PROLOG-System durchsuchen muss, um alle Antworten auf diese Anfrage zu finden.

Aus Gründen der Übersichtlichkeit werden in dieser Abbildung die Namen der Prädikate verkürzt geschrieben, z.B. steht statt »ist_vater_von(Wer,Wem)« nur »vater(Wer,Wem)«.

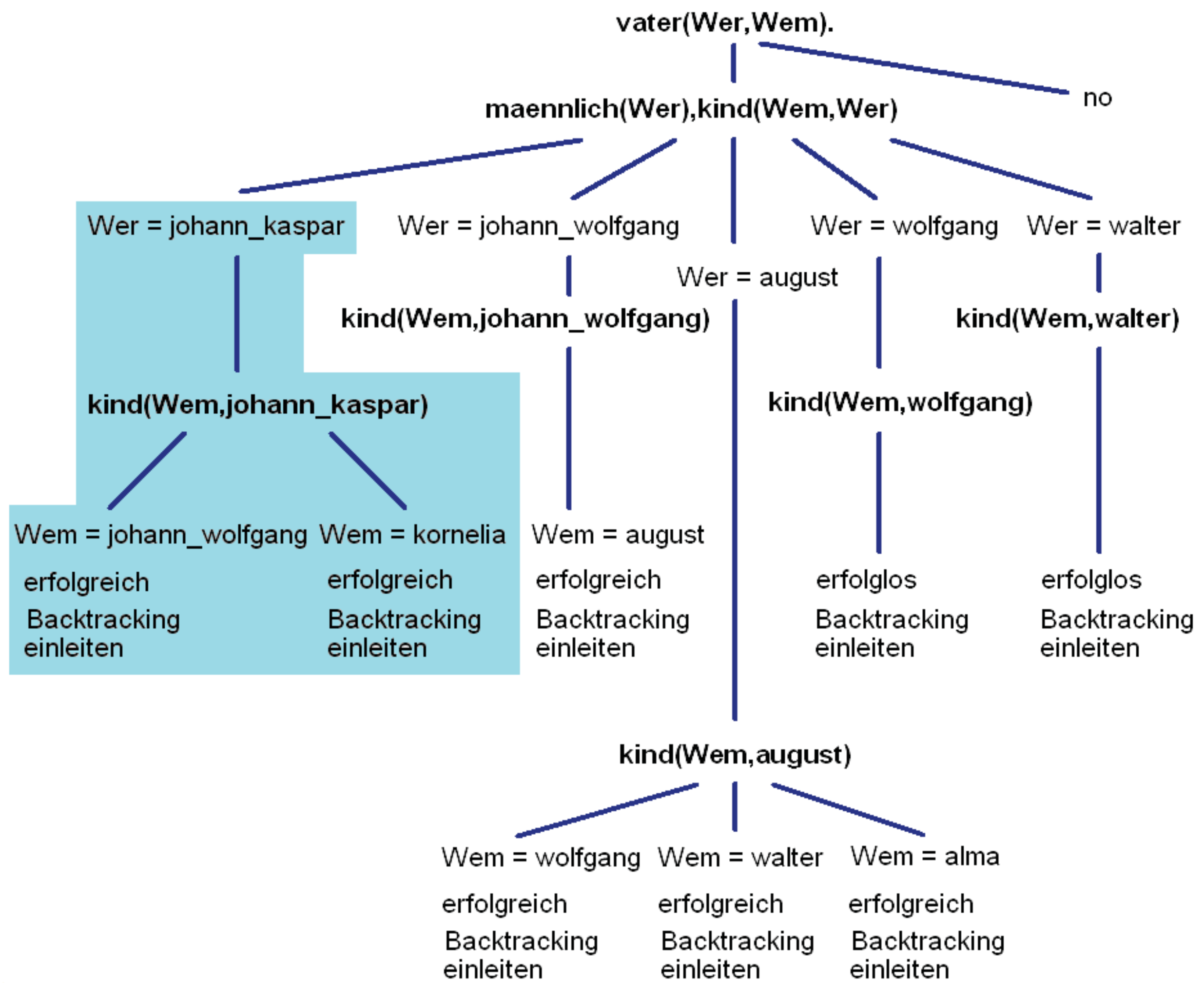


Abbildung 1.1.1 Grafische Darstellung der Suche des PROLOG-Systems nach Antworten auf die Anfrage `?- ist_vater_von(Wer,Wem).` im Programm Familie Goethe

Exkurs:

Das PROLOG-Programm

```
p(a).
p(b).
q(1).
q(2).
q(3).
s(X,Y):- p(X),q(Y).
s(c,4).
```

enthält eine Regel. In der Regel ist `s(X,Y)` der Regelkopf und `p(X),q(Y)` der Regelrumpf. Sowohl der Regelkopf als auch die Bestandteile des Regelrumpfs sind Ziel bzw. Teilziele der Suche des PROLOG-Systems nach Antworten auf die Anfrage (Ziel) `?- s(U,V).`

Die Antworten des PROLOG-System sind:

```
U = a, V = 1;
U = a, V = 2;
U = a, V = 3;
U = b, V = 1;
U = b, V = 2;
U = b, V = 3;
U = c, V = 4;
No
```

Die Suche des PROLOG-Systems nach Antworten auf obige Anfrage kann eingeschränkt werden, indem das Standard-Prädikat *cut/0* (in PROLOG als Ausrufezeichen geschrieben) in die Regel `s(X,Y):- p(X),q(Y).` eingefügt wird.

Dabei sind zwei Fälle zu unterscheiden.

Fall 1: Das Standard-Prädikat *cut/0* wird am Ende des Regelrumpfs eingefügt.

```
s(X,Y):- p(X),q(Y),!.
```

Auf die Anfrage `?- s(U,V).` liefert das PROLOG-System die Antworten:

```
U = a, V = 1;
No
```

Begründung: Das Standard-Prädikat *cut/0* bewirkt, dass für die Teilziele `p(X)` und `q(Y)`, die innerhalb des Regelrumpfs links vom Cut stehen, nach Auffinden der ersten Antwort keine weitere gesucht wird. Danach wird für das Ziel `s(U,V)` ebenfalls keine weitere Antwort gesucht.

Fall 2: Das Standard-Prädikat *cut/0* wird in den Regelrumpf, jedoch nicht am Ende, eingefügt.

```
(a)  s(X,Y) :- p(X),!,q(Y).
```

Auf die Anfrage `?- s(U,V).` liefert das PROLOG-System die Antworten:

```
U = a, V = 1;
U = a, V = 2;

U = a, V = 3;
No
```

Begründung: Das Standard-Prädikat *cut/0* bewirkt, dass für das Teilziel `p(X)`, das innerhalb des Regelrumpfs links vom Cut steht, nach Auffinden der ersten Antwort

keine weitere Antwort gesucht wird. Für das Teilziel `q(Y)`, das rechts vom Cut steht, werden alle Antworten gesucht. Danach wird für das Ziel `s(U,V)` keine weitere Antwort gesucht.

```
(b) s(X,Y):- !,p(X),q(Y).
```

Auf die Anfrage `?- s(U,V).` liefert das PROLOG-System die Antworten:

```
U = a, V = 1;
U = a, V = 2;
U = a, V = 3;
U = b, V = 1;
U = b, V = 2;
U = b, V = 3;
No
```

Begründung: Das Standard-Prädikat *cut/0* bewirkt, dass für die Teilziele `p(X)` und `q(Y)`, die rechts vom Cut stehen, alle Antworten gesucht werden. Danach wird für das Ziel `s(U,V)` keine weitere Antwort gesucht.

Möchte man nur die Antworten des in der Abbildung 1.1.1 markierten Teilbaums anzeigen lassen, dann muss in den Rumpf der Regel des Prädikats *ist_vater_von/2* das Standard-Prädikat *cut/0* folgendermaßen eingefügt werden:

```
ist_vater_von(X,Y):- ist_maennlich(X),!,
                     ist_kind_von(Y,X).
```

Das Prädikat *cut/0* bewirkt, dass das PROLOG-System bei der Anfrage

```
?- ist_vater_von(Wer,Wem).
```

nur im markierten Teilbaum nach Antworten sucht, alle anderen Teilbäume bleiben unbeachtet.

Wird die Anordnung der Fakten im Prädikat *ist_maennlich/1* geändert

```
ist_maennlich(wolfgang).
ist_maennlich(august).
ist_maennlich(walter).
ist_maennlich(johann_kaspar).
ist_maennlich(johann_wolfgang).,
```

ergibt sich bei der Suche nach Antworten auf die Anfrage

```
?- ist_vater_von(Wer,Wem).
```

ein neuer Baum. Das heißt, das PROLOG-System liefert die Antworten entsprechend der veränderten Baumstruktur.

1.2 Relationale Datenbanken in PROLOG

In einem Gymnasium werden die Daten von Facharbeiten der Schüler tabellarisch erfasst. Diese Tabellen bilden die Grundlage für eine relationale Datenbank. Jede Tabelle beschreibt genau e i n e Relation.

Tabelle Schüler:

Schülernummer	Name	Vorname
1	Bauer	Renate
2	Baumbach	Ilona
3	Franke	Frieder
7	Kunze	Ines
12	Maier	Peter
18	Schmidt	Sabine
19	Schulze	Thomas
24	Zimmermann	Uwe

Tabelle Unterrichtsfach:

Fachnummer	Unterrichtsfach
1	Deutsch
2	Mathematik
3	Englisch
7	Chemie
8	Physik
11	Informatik

Tabelle Facharbeit:

Facharbeitsnummer	Thema	Fachnummer
33	Kirchhoffsche Gesetze	8
35	Lösen linearer Gleichungssysteme	2
56	The life of Shakespeare	3
74	Goethes Schaffen in seiner Weimarer Zeit	1
106	Problemlösen mit Oberon	11
134	Elektronenkonfigurationen der Elemente	7

Tabelle Zuordnung:

Facharbeitsnummer	Schülernummer
33	19
33	7
35	19
56	3
74	24
106	2
106	18
134	1
134	12

Alle Relationen können in ein PROLOG-Programm übertragen werden. Die Relationen werden durch die Prädikate *schueler/3*, *unterrichtsfach/2*, *facharbeit/3* und *zuordnung/2* beschrieben. Jedes Prädikat besteht ausschließlich aus F a k t e n .

```
/* Programm Datenbank */
/* Tabelle Schueler */
schueler(1,'Bauer','Renate').
schueler(2,'Baumbach','Ilona').
schueler(3,'Franke','Frieder').
schueler(7,'Kunze','Ines').
schueler(12,'Maier','Peter').
schueler(18,'Schmidt','Sabine').
schueler(19,'Schulze','Thomas').
schueler(24,'Zimmermann','Uwe').

/* Tabelle Unterrichtsfach */
unterrichtsfach(1,'Deutsch').
unterrichtsfach(2,'Mathematik').
unterrichtsfach(3,'Englisch').
unterrichtsfach(7,'Chemie').
unterrichtsfach(8,'Physik').
unterrichtsfach(11,'Informatik').

/* Tabelle Facharbeit */
facharbeit(33,'Kirchhoffsche Gesetze',8).
facharbeit(35,'Loesen linearer Gleichungssysteme',2).
facharbeit(56,'The life of Shakespeare',3).
facharbeit(74,'Goethes Schaffen in seiner Weimarer Zeit',1).
facharbeit(106,'Problemloesen mit Oberon',11).
facharbeit(134,'Elektronenkonfigurationen der Elemente',7).

/* Tabelle Zuordnung */
zuordnung(33,19).
zuordnung(33,7).
zuordnung(35,19).
zuordnung(56,3).
zuordnung(74,24).
zuordnung(106,2).
zuordnung(106,18).
zuordnung(134,1).
zuordnung(134,12).
```

An die Datenbank können beispielsweise folgende Fragen gestellt werden:

- Wie heißen die Schüler, die im Unterrichtsfach Informatik eine Facharbeit schreiben?
- Wie heißen die Schüler, die eine Facharbeit zum Thema 'Elektronenkonfigurationen der Elemente' anfertigen?
- In welchem Unterrichtsfach schreibt Uwe Zimmermann eine Facharbeit?

In PROLOG werden diese Fragen als Anfragen formuliert:

```
?- unterrichtsfach(FNr,'Informatik'),
   facharbeit(FANr,_,FNr),
   zuordnung(FANr,SNr),
   schueler(SNr,N,Vn).

?- facharbeit(FANr,'Elektronenkonfigurationen der Elemente',_),
   zuordnung(FANr,SNr),
   schueler(SNr,N,Vn).

?- schueler(SNr,'Zimmermann','Uwe'),
   zuordnung(FANr,SNr),
   facharbeit(FANr,_,FNr),
   unterrichtsfach(FNr,UF).
```

Die Antworten des PROLOG-Systems auf die erste Anfrage lauten:

```
FNr = 11, FANr = 106, SNr = 2, N = Baumbach, Vn = Ilona;
FNr = 11, FANr = 106, SNr = 18, N = Schmidt, Vn = Sabine;
No
```

Alle Antworten auf die zweite Anfrage lauten:

```
FANr = 134, SNr = 1; N = Bauer, Vn = Renate;
FANr = 134, SNr = 12, N = Maier, Vn = Peter;
No
```

Auf die dritte Anfrage antwortet das PROLOG-System:

```
SNr = 24, FANr = 74, FNr = 1, UF = Deutsch;
No
```

Da PROLOG keine Datenbanksprache ist, müssen alle für eine relationale Datenbanksprache benötigten Anweisungen selbst entworfen und implementiert werden.

1.3 Rekursion

VEIT BACH (1550-1619) zählt zu den ältesten musizierenden Mitgliedern der Familie Bach. JOHANN SEBASTIAN BACH (1685-1785) ist ein Nachkömmling von Veit Bach. Der folgende Text enthält Fakten aus dem Stammbaum der Familie Bach.

> Veit ist der Vater von Johannes. Johannes ist der Vater von Christoph und Heinrich. Christoph ist der Vater von Johann Ambrosius. Heinrich ist der Vater von Johann Michael. Johann Ambrosius ist der Vater von Johann Christoph und Johann Sebastian. Johann Sebastian ist der Vater von Wilhelm Friedemann, Carl Phillip Emanuel, Johann Christoph Friedrich und Johann Christian. Johann Christoph Friedrich ist der Vater von Wilhelm Friedrich Ernst.

Die Fakten werden in ein PROLOG-Programm übertragen.

```
/* Programm Familie Bach */
ist_kind_von(johannes,veit).
ist_kind_von(christoph,johannes). ist_kind_von(heinrich,johannes).
ist_kind_von(johann_ambrosius,christoph).
ist_kind_von(johann_michael,heinrich).
ist_kind_von(johann_christoph,johann_ambrosius).
ist_kind_von(johann_sebastian,johann_ambrosius).
ist_kind_von(wilhelm_friedemann,johann_sebastian).
ist_kind_von(carl_philipp_emanuel,johann_sebastian).
ist_kind_von(johann_christoph_friedrich,johann_sebastian).
ist_kind_von(johann_christian,johann_sebastian).
ist_kind_von(wilhelm_friedrich_ernst,johann_christoph_friedrich).
```

Die Aufgabe besteht darin, das PROLOG-Programm um ein Prädikat zu erweitern, das bei Anfrage darüber Auskunft gibt, wer in der Familie Bach Nachkömmling von wem ist. Ein Nachkömmling eines im Text genannten Familienmitgliedes ist entweder das Kind oder der Enkel, der Urenkel, der Ururenkel usw. Johann Sebastian Bach ist Nachkömmling von Veit, weil er dessen Ururenkel ist.

Die Nachkömmling-Beziehung mithilfe der Kind-, Enkel-, Urenkel- ... Beziehung zu formulieren ist recht aufwendig.

Ein Beispiel zeigt die Lösung des Problems:

> Es ist die Frage zu beantworteten, ob Johann Sebastian ein Nachkömmling von Veit ist.

Johann Sebastian ist ein Nachkömmling von Veit, wenn er ein Kind von Veit oder ein Kind eines Nachkömmlings von Veit ist.

(1) Annahme: Johann Sebastian ist Kind von Veit.
    Da er nicht Kind von Veit ist, muss nach dem Vater von Johann Sebastian gesucht werden, gefunden wird Johann Ambrosius.

(2) Annahme: Johann Ambrosius ist Kind von Veit.
Da er nicht Kind von Veit ist, muss nach dem Vater von Johann Ambrosius gesucht werden, gefunden wird Christoph.

(3) Annahme: Christoph ist Kind von Veit.
Da er nicht Kind von Veit ist, muss nach dem Vater von Christoph gesucht werden, gefunden wird Johannes.

(4) Annahme: Johannes ist Kind von Veit.
Das ist der Fall.

Die Antwort auf obige Frage lautet:

Johann Sebastian ist ein Nachkömmling von Veit.

PROLOG ist eine Programmiersprache, in der Wiederholungen, wie eben beschrieben, nur rekursiv, durch endständige Rekursion realisiert werden können. Die Lösung des Problems gibt das rekursive Prädikat *ist_nachkoemmling_von/2* an:

```
ist_nachkoemmling_von(X,Y):- ist_kind_von(X,Y).
ist_nachkoemmling_von(X,Y):- ist_kind_von(X,Z),
                             ist_nachkoemmling_von(Z,Y).
```

Auf die Anfrage

```
?- ist_nachkoemmling_von(johann_sebastian,veit).
```

antwortet das PROLOG-System `Yes` und auf die Anfrage

```
?- ist_nachkoemmling_von(veit,johann_sebastian).
```

antwortet das PROLOG-System `No`.

CARL FRIEDRICH GAUSS (1777-1855) erhielt mit neun Jahren von seinem Lehrer die Aufgabe, die Zahlen 1 bis 60 zu addieren. Nach kurzer Zeit schrieb Carl Friedrich die Zahl 1830 auf seine Schiefertafel. Wie hat er das gemacht, dachte der Lehrer. Gauß erläuterte, er habe im Kopf gerechnet:

1,	2,	3,	4,	5,	...,	30
+ 60,	59,	58,	57,	56,	...,	31
61,	61,	61,	61,	61,	...,	61

und anschließend durch Multiplikation von 30 * 61 die Summe 1830 erhalten.

Der von Carl Friedrichs Lehrer erwartete Rechenweg zum Lösen der Aufgabenstellung kann in PROLOG mit dem rekursiven Prädikat *summe/2* realisiert werden. Das Prädikat soll bei Anfrage die Summe der natürlichen Zahlen von 1 bis n berechnen.

Die rekursive Definition für das Berechnen der Summe

Summe(1) = 1,
Summe(n) = Summe(n - 1) + n, n > 1

wird in PROLOG wie folgt implementiert:

```
/* Programm Summe */

summe(1,1).
summe(N,S):- N > 1,M is N - 1,summe(M,ZS),S is ZS + N.
```

Das Prädikat *summe/2* besteht aus einem Faktum und einer Regel. Zu beachten ist, dass der Variablen N der Wert N - 1 und der Variable S der Wert S + N nicht zugewiesen (unifiziert) werden kann. Deshalb werden im Prädikat *summe/2* zusätzlich die Variablen M und ZS verwendet.

Um die Summe der natürlichen Zahlen von 1 bis 60 zu berechnen, wird die Anfrage gestellt:

```
?- summe(60,Summe).
```

Dem Entwurf rekursiver Prädikate kommt besondere Bedeutung zu. Angenommen in der Regel des Prädikats *summe/2* wird der Ausdruck N > 1 nicht angegeben, dann würde das PROLOG -System auf die Anfrage

```
?- summe(60,Summe).
```

zwar die Antwort Summe = 1830 ausgeben, jedoch beim gewillkürten Backtracking[1] einen Fehler, Stacküberlauf, verursachen.

Der Franzose Édouard Lucas (1842-1891) stellte auf der Pariser Weltausstellung 1889 eines seiner kombinatorischen Spiele aus, das bis heute unter dem Namen »Turm von Hanoi« bekannt ist.

Ein Turm aus n Scheiben (n >= 1) ist ab n = 2 so gebaut, dass stets eine kleinere Scheibe auf einer größeren zu liegen kommt. Die n Scheiben sollen Scheibe für Scheibe von einem Startort über eine Ablage zu einem Zielort transportiert werden. Bedingung ist, dass zu keiner Zeit eine größere Scheibe auf einer kleineren liegt.

Es ist folgender Algorithmus auszuführen:

- Transport von n - 1 Scheiben vom Start- über den Zielort zur Ablage
- Transport einer Scheibe vom Start- zum Zielort
- Transport von n - 1 Scheiben von der Ablage über den Start- zum Zielort

Zu beachten ist, dass sich die Startort-, Ablage- und Zielortsposition fortlaufend ändern.

```
/* Programm Turm von Hanoi */

transportiere(0,_,_,_).
transportiere(N,S,A,Z):- N > 0,M is N - 1,
                         transportiere (M,S,Z,A),
                         write(S),write(' ⟶ '),write(Z),nl,
                         transportiere (M,A,S,Z).
```

[1] Durch den Anwender erzwungenes Backtracking.

Eine Anfrage an das Programm kann lauten:

```
?- transportiere(3,start,ablage,ziel).
```

Das PROLOG-Systems liefert folgende Antworten:

```
start ⟶ ziel
start ⟶ ablage
ziel ⟶ ablage
start ⟶ ziel
ablage ⟶ start
ablage ⟶ ziel
start ⟶ ziel
Yes
```

Die rekursive Definition zum Berechnen der Anzahl der Scheibentransporte

Anzahl(0) = 0,
Anzahl(n) = 2 * Anzahl(n - 1) + 1, n > 0

wird in PROLOG wie folgt implementiert:

```
/* Erweiterung des Programms Tuerme von Hanoi  */

anzahl(0,0).
anzahl(N,A):- N > 0,M is N - 1,anzahl(M,ZA),A is 2 * ZA + 1.
```

Das PROLOG-Systems liefert auf die Anfrage

```
?- anzahl(3,Anzahl).
```

die Antworten:

```
Anzahl = 7;
No.
```

Im Prädikat anzahl/2 darf der Ausdruck N > 0 nicht weggelassen werden, da sonst bei gewillkürtem Backtracking ein Stacküberlauf auftritt.

1.4 Listen

Die Stadt Mühlhausen liegt im Nordwesten Thüringens. In der Stadt weilten u. a. der Minnesänger WALTHER VON DER VOGELWEIDE (um 1170-1228), der Theologe und Bauernführer THOMAS MÜNTZER (1486-1525), der Mönch HEINRICH PFEIFFER (?-1525), der Komponist JOHANN SEBASTIAN BACH (1685-1750) und der Ingenieur JOHANN AUGUST RÖBLING (1806-1869).

Die Familiennamen der genannten Persönlichkeiten der Stadt Mühlhausen sollen in einer Liste erfasst werden.

Einer PROLOG-Liste liegt die folgende rekursive Definition zugrunde:

Eine Liste ist entweder die leere Liste [ ] o d e r
eine Liste mit einem Kopfelement und einer Restliste [Kopf|Rest].

Soll ein Element in eine PROLOG-Liste aufgenommen werden, kann das mit dem dreistelligen Prädikats *hinein/3* erfolgen.

```
hinein(K,R,[K|R]).
```

Das Prädikat besteht aus einem Faktum.

Die Anfrage

```
?- hinein(von_der_vogelweide,[],L).
```

bewirkt, dass der Familienname von der Vogelweide als Element in die leere Liste eingefügt wird.

```
L = [von_der_vogelweide].
```

Die Anfrage

```
?- hinein(muentzer,[von_der_vogelweide],L).
```

bewirkt, dass der Familienname Müntzer als Kopfelement in die Liste eingefügt wird.

```
L = [muentzer,von_der_vogelweide].
```

Auf diese Weise können die Familiennamen aller oben genannten Persönlichkeiten in die PROLOG-Liste eingefügt werden.

Soll das erste Element aus einer PROLOG-Liste herausgenommen werden, kann das mit dem dreistelligen Prädikats *heraus/3* erfolgen.

```
heraus([K|R],K,R).
```

Dieses Prädikat besteht ebenfalls aus einem Faktum.

Die Anfrage

```
?- heraus([pfeiffer,muentzer,von_der_vogelweide],E,L).
```

bewirkt, dass das Kopfelement `pfeiffer` aus der Liste herausgenommen wird.

```
E = pfeiffer.
```

Es bleibt die Restliste:

```
L = [muentzer,von_der_vogelweide]
```

Soll geprüft werden ob ein bestimmtes Element in einer Liste enthalten ist, kann das mit dem rekursiven Prädikat *enthalten/2* erfolgen.

```
enthalten(E,L):- heraus(L,E,_).
enthalten(E,L):- heraus(L,_,NL),enthalten(E,NL).
```

Ist das gesuchte Element Kopfelement der Liste, war die Suche erfolgreich. Ist das gesuchte Element nicht Kopfelement der Liste, wird das Kopfelement aus der Liste herausgenommen und die Suche wird in der neuen Liste fortgesetzt. Ist die neue Liste leer, bleibt die Suche erfolglos. In jeder Regel des Prädikats *enthalten/2* steht eine anonyme Variable »_«. Eine anonyme Variable wird verwendet, wenn der Wert der Variablen nicht benötigt wird. Das Prädikat *enthalten/2* kann ohne das Prädikat *heraus/3* implementiert werden.

```
enthalten(E,[E,_]).
enthalten(E,[_|L]):- enthalten(E,L).
```

Es soll darauf hingewiesen werden, dass eine derartige Darstellung eines Prädikats vielen Schülern schwer zugänglich ist.

Auf die Anfrage, ob der Familienname Müntzer in der Liste enthalten ist

```
?- enthalten(muentzer,[pfeiffer,muentzer,von_der_vogelweide]).
```

antwortet das PROLOG-System `Yes`. Auf die Anfrage, ob der Familienname Bach in der Liste enthalten ist

```
?- enthalten(bach,[pfeiffer,muentzer,von_der_vogelweide]).
```

antwortet das PROLOG-Systems `No`. Auf die Anfrage

```
?- enthalten(Name,[pfeiffer,muentzer,von_der_vogelweide]).
```

kann das PROLOG-System jeden in der Liste enthaltenen Familiennamen ausgeben.

Sollen die Elemente einer Liste in der umgekehrten Reihenfolge in einer neuen Liste ablegt werden, kann das mit dem Prädikats *umdrehen/3* erfolgen.

```
umdrehen([],NL,NL).
umdrehen(L,HL,NL):- heraus(L,E,L1),
                    hinein(E,HL,L2),
                    umdrehen(L1,L2,NL).
```

Dieses dreistellige Prädikat besteht aus einem Faktum und einer Regel.

Die Anfrage zum Umdrehen der Liste `[l,a,g,e,r]` lautet:

```
?- umdrehen([l,a,g,e,r],[],L).
```

und liefert die Antwort:

```
L = [r,e,g,a,l].
```

Die Arbeitsweise des Prädikats *umdrehen/3* kann mit zwei Stapeln veranschaulicht werden. Dem Prädikat werden die Liste `[l,a,g,e,r]` und die leere Liste `[]` übergeben.

```
      l
      a
      g
      e
      r
Stapel_1: [l,a,g,e,r]          Stapel_2: []
```

Das Kopfelement `l` wird aus der Liste `[l,a,g,e,r]` herausgenommen und in die leere Liste `[]` als Kopfelement aufgenommen.

```
      a
      g
      e
      r                             l
Stapel_1: [a,g,e,r]            Stapel_2: [l]
```

Das Kopfelement `a` wird aus der Liste `[a,g,e,r]` herausgenommen und in die Liste `[l]` als Kopfelement aufgenommen.

```
      g
      e                             a
      r                             l
Stapel_1: [g,e,r]              Stapel_2: [a,l]
```

Das Kopfelement `g` wird aus der Liste `[g,e,r]` herausgenommen und in die Liste `[a,l]` als Kopfelement aufgenommen.

```
      g
      e                             a
      r                             l
Stapel_1: [g,e,r]              Stapel_2: [g,a,l]
```

Das Kopfelement `e` wird aus der Liste `[e,r]` herausgenommen und in die Liste `[g,a,l]` als Kopfelement aufgenommen.

```
                                    e
                                    g
                                    a
      r                             l
Stapel_1: [r]                  Stapel_2: [e,g,a,l]
```

Das Kopfelement `r` wird aus der Liste `[r]` herausgenommen und in die Liste `[e,g,a,l]` als Kopfelement aufgenommen.

```
                                    r
                                    e
                                    g
                                    a
                                    l
Stapel_1: []                   Stapel_2: [r,e,g,a,l]
```

Die Liste ist leer `[]` und die neue Liste lautet `[r,e,g,a,l]`.

Hinweis: Das Prädikat *umdrehen/3* kann ohne die Prädikate *hinein/3* und *heraus/3* implementiert werden.

```
umdrehen([],NL,NL).
umdrehen([E|L],HL,NL):- umdrehen(L,[E|HL],NL).
```

1.5 Wegsuche in Graphen

Ein Tourist kommt in Jena am Paradiesbahnhof an. Das Ziel seines Besuchs ist das Stadtmuseum Göhre am Marktplatz. Auf dem Weg dorthin möchte er weitere Sehenswürdigkeiten betrachten. Er fragt Passanten nach Wegen, die zum Marktplatz führen, und erhält folgende Auskünfte:

> »Vom Paradiesbahnhof (a) können Sie zur Hauptpost (b) oder zum Gasthaus Roter Hirsch (d) gehen. Von der Hautpost kommen Sie zur Goethe Galerie (c). Vom Gasthaus Roter Hirsch (d) können Sie zur Hauptpost (b), zur Goethe Galerie (c) oder zum Holzmarkt (e) gehen. Von der Goethe Galerie (c) gelangen Sie direkt zum Marktplatz (f) oder Sie gehen zum Holzmarkt (e) und von dort zum Marktplatz (f).«

Abbildung 1.5.1 veranschaulicht die dem Touristen genannten Wege. Der Graph enthält sechs Knoten, die jeweils eine Sehenswürdigkeit repräsentieren, und neun gerichteten Kanten. Im Graphen treten keine Zyklen auf.

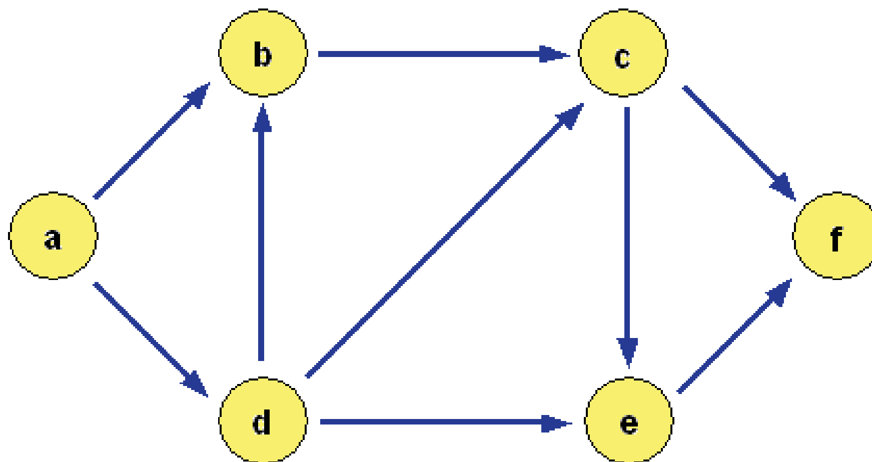


Abbildung 1.5.1 Gerichteter Graph ohne Zyklen

Dem Prädikat *gehe/2* liegt der oben abgedruckte Text zugrunde:

```
gehe(paradiesbahnhof,hauptpost).
gehe(paradiesbahnhof,gasthaus).
gehe(hauptpost,goethe_galerie).
gehe(goethe_galerie,holzmarkt).
gehe(goethe_galerie,marktplatz).
gehe(gasthaus,hauptpost).
gehe(gasthaus,goethe_galerie).
gehe(gasthaus,holzmarkt).
gehe(holzmarkt,marktplatz).
```

Dem Prädikat *gerichtete_kante/2* liegt die Abbildung 1.5.1 zugrunde:

```
gerichtete_kante(a,b).
gerichtete_kante(a,d).
gerichtete_kante(b,c).
gerichtete_kante(c,e).
gerichtete_kante(c,f).
gerichtete_kante(d,b).
gerichtete_kante(d,c).
gerichtete_kante(d,e).
gerichtete_kante(e,f).
```

Das folgende Programm ermöglicht bei Anfrage die Wegsuche von einem Start- zu einem Zielknoten im dargestellten Graphen.

```
/* Programm Graph */

gerichtete_kante(a,b).
gerichtete_kante(a,d).
gerichtete_kante(b,c).
gerichtete_kante(c,e).
gerichtete_kante(c,f).
gerichtete_kante(d,b).
gerichtete_kante(d,c).
gerichtete_kante(d,e).
gerichtete_kante(e,f).

weg_gg(X,X).
weg_gg(X,Y):- gerichtete_kante(X,Z),
              weg_gg(Z,Y).
```

Jede gerichtete Kante zwischen zwei Knoten ist als Faktum im Programm enthalten. In jedem Faktum ist die Reihenfolge der Knoten relevant, weil dadurch die gerichtete Kante bestimmt wird. Beispielsweise beschreibt das Faktum

```
gerichtete_kante(a,b).
```

die Kante vom Knoten a zum Knoten b. Neun Fakten bilden das Prädikat *gerichtete_kante/2*. Das rekursive Prädikat *weg_gg/2* dient der Wegsuche im gerichteten Graph.
Bei der Anfrage

```
?- weg_gg(a,f).
```

erfolgt eine Wegsuche wie in der Abbildung 1.5.2 dargestellt:

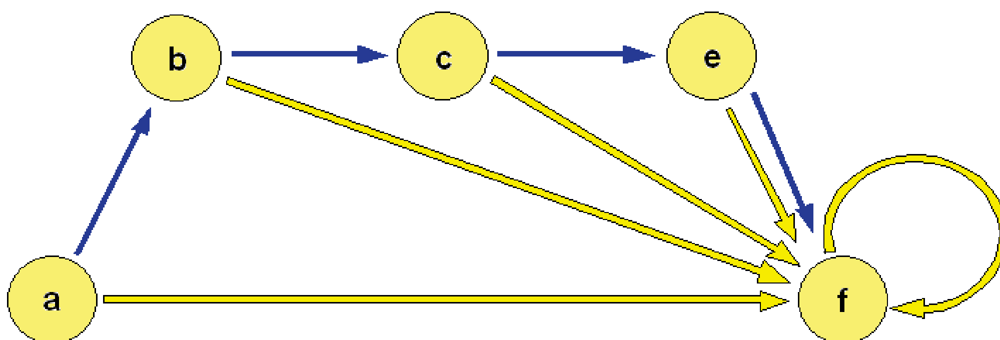


Abbildung 1.5.2 Wegsuche in einem gerichteten Graphen ohne Zyklen

Aus der Abbildung 1.5.2 wird ersichtlich, dass:

- auf der Suche nach einem Weg vom Startknoten a zum Zielknoten f die Kante vom Knoten a zum Knoten b durchgangen und vom Knoten b ein Weg zum Knoten f gesucht wird,
- auf der Suche nach einem Weg vom Knoten b zum Knoten f die Kante vom Knoten b zum Knoten c durchgangen und ein Weg vom Knoten c zum Knoten f gesucht wird,
- auf der Suche nach einem Weg vom Knoten c zum Knoten f die Kante vom Knoten c zum Knoten e durchgangen und ein Weg vom Knoten e zum Knoten f gesucht wird,

- auf der Suche nach einem Weg vom Knoten e zum Knoten f die Kante vom Knoten e zum Knoten f durchgangen und ein Weg vom Knoten f zum Knoten f gesucht wird,
- der Zielknoten f erreicht wird.

Die Wegsuche war erfolgreich und das PROLOG-System antwortet `Yes`.

Die Eigenschaft von a, b, c, d, e und f, Knoten des gerichteten Graphen zu sein, wurde nicht als Prädikat, beispielsweise *knoten/1*, in das Programm übertragen, weil die Beziehung der gerichteten Kante zwischen zwei Knoten alle Knoten des Graphen erfasst.

Sollen Wege vom Start- zum Zielknoten gesucht und als Listen ausgegeben werden, kann das Programm wie folgt erweitert werden:

```
/* Erweiterung des Programms Graph */

jeder_weg_gg(X,X,[X]).
jeder_weg_gg(X,Y,L):- gerichtete_kante(X,Z),
                      jeder_weg_gg(Z,Y,NL),
                      hinein(X,NL,L).
```

In die Erweiterung des Programms Graph wurde das Hilfsprädikat *hinein/3* eingefügt.
Bei Anfrage

```
?- jeder_weg_gg(a,f,L).
```

wird der erste Weg gesucht und `L = [a,b,c,e,f]` ausgegeben. Erfolgt durch Interaktion des Nutzers ein Backtracking, gewillkürtes Backtracking, wird der nächste Weg gesucht und in einer Liste ausgegeben. In der Abbildung 1.5.3 ist die Suche nach allen Wegen vom Startknoten a zum Zielknoten f dargestellt.

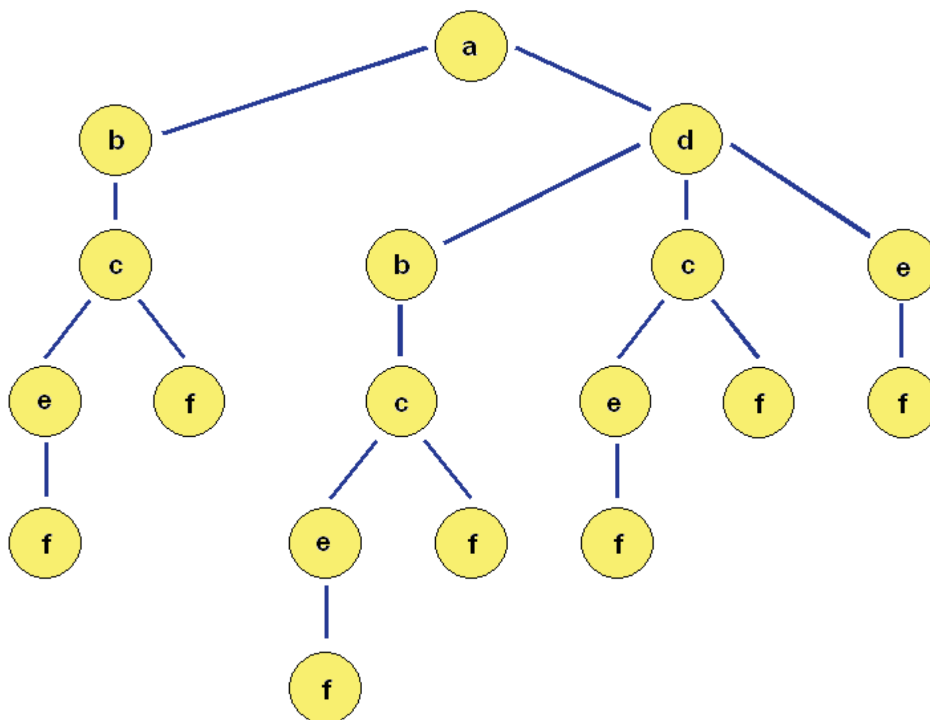


Abbildung 1.5.3 Darstellung aller Wege vom Startknoten a zum Zielknoten f

Das Prädikat *jeder_weg_gg/3* kann ohne das Prädikat *hinein/3* implementiert werden:

```
jeder_weg_gg(X,X,[X]).

jeder_weg_gg(X,Y,[X|L]):- gerichtete_kante(X,Z),
                          jeder_weg_gg(Z,Y,L).
```

Bei einer Wegsuche in einem ungerichteten Graphen muss beachtet werden, dass in jedem Fall Zyklen auftreten. Sie ergeben sich dadurch, dass jede ungerichtete Kante zwischen zwei Knoten sowohl in die eine als auch in die andere Richtung durchgangen werden kann.

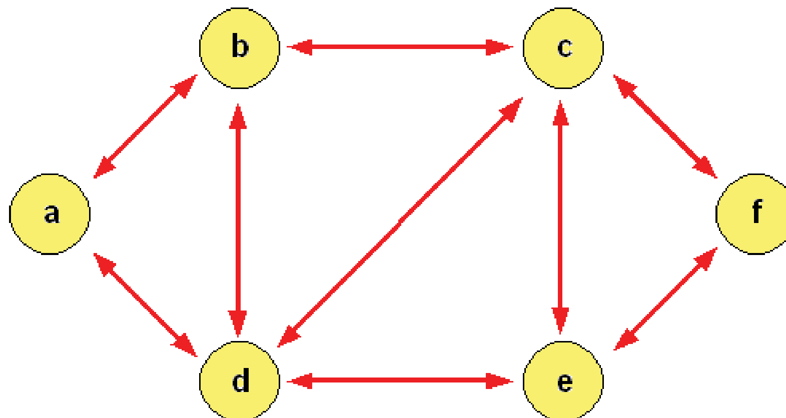


Abbildung 1.5.4 Ungerichteter Graph

Die ungerichteten Kanten des Graphen aus Abbildung 1.5.4 werden durch das Prädikat *ungerichtete_kante/2* beschrieben.

```
/* Erweiterung des Programms Graph */

ungerichtete_kante(X,Y):- gerichtete_kante(X,Y).
ungerichtete_kante(X,Y):- gerichtete_kante(Y,X).
```

Das Prädikat besteht aus zwei Regeln, könnte aber auch aus der Regel

```
ungerichtete_kante(X,Y):- gerichtete_kante(X,Y);
                          gerichtete_kante(Y,X).
```

bestehen.

In die Erweiterung des Programms Graph sind die Hilfsprädikate *heraus/3*, *hinein/3*, *enthalten/2* und *umdrehen/3* einzufügen.

```
/* Erweiterung des Programms Graph */

weg_ug(X,X,_).
weg_ug(X,Y,L):- ungerichtete_kante(X,Z),
                not enthalten(Z,L),
                hinein(Z,L,NL),
                weg_ug(Z,Y,NL).
```

Das Prädikat *weg_ug/3* ermöglicht bei Anfrage die Wegsuche von einem Start- zu einem Zielknoten.

Bei der Suche wird eine Liste angelegt, die alle Knoten enthält, die genau einmal besucht wurden. Wird bei der vom Startknoten ausgehenden Wegsuche ein Knoten erreicht, der be-

reits in der Liste enthalten ist, wird die Suche gestoppt, zum vorhergehenden Knoten zurückgekehrt und die Wegsuche zum Zielknoten über einen a n d e r e n Knoten fortgesetzt.
Auf die Anfrage, ob es einen Weg vom Startknoten f zum Zielknoten a gibt

```
?- weg_ug(f,a,[f]).
```

antwortet das PROLOG-System `Yes`.
Falls alle möglichen Wege von einem Start- zu einem Zielknoten gesucht und als Listen ausgegeben werden sollen, muss das Programm folgendermaßen modifiziert werden:

```
/* Modifizierung des Programms Graph */

jeder_weg_ug(X,X,L,L2):- umdrehen(L,[],L2).
jeder_weg_ug(X,Y,L,L2):- ungerichtete_kante(X,Z),
                         not enthalten(Z,L),
                         hinein(Z,L,NL),
                         jeder_weg_ug(Z,Y,NL,L2).
```

Bei der Anfrage

```
?- jeder_weg_ug(a,f,[a],L).
```

wird der erste Weg gesucht und die Liste `L = [a,b,c,e,f]` ausgegeben. Erfolgt ein gewillkürtes Backtracking, wird der nächste Weg gesucht und in Form einer Liste ausgegeben.

Um bei der Anfrage:

```
?- jeder_weg_ug(a,f,[a],L).
```

die Listen aller Wege ohne gewillkürtes Backtracking zu erhalten, wird das Standardprädikat *fail/0* in das Prädikat *jeder_weg_ug/4* implementiert:

```
jeder_weg_ug(X,X,L):- umdrehen(L,[],L2),
                      write(L2),nl.

jeder_weg_ug(X,Y,L):- ungerichtete_kante(X,Z),
                      not enthalten(Z,L),
                      hinein(Z,L,NL),
                      jeder_weg_ug(Z,Y,NL),
                      fail.
```

Das Prädikat *jeder_weg_ug/3* kann ohne das Prädikat *hinein/3* implementiert werden.

```
jeder_weg_ug(X,X,L):- umdrehen(L,[],L2),
                      write(L2),nl.

jeder_weg_ug(X,Y,L):- ungerichtete_kante(X,Z),
                      not enthalten(Z,L),
                      jeder_weg_ug(Z,Y,[Z|L]),
                      fail.
```

In Südwestthüringen liegt die Stadt Meiningen, die von Süden nach Norden von der Werra durchflossen wird. Ein Stadtteil von Meiningen befindet sich auf einer Insel im Fluss. Das Westufer und die Insel verbindet eine, das Ostufer und die Insel verbinden zwei Brücken. Nördlich und südlich der Insel verbindet jeweils eine Brücke das West- und das Ostufer miteinander. Es ist vom Westufer und von der Insel aus möglich, Wege so zu beschreiten, dass bei jedem der Wege jede der fünf Brücken genau einmal überquert wird.

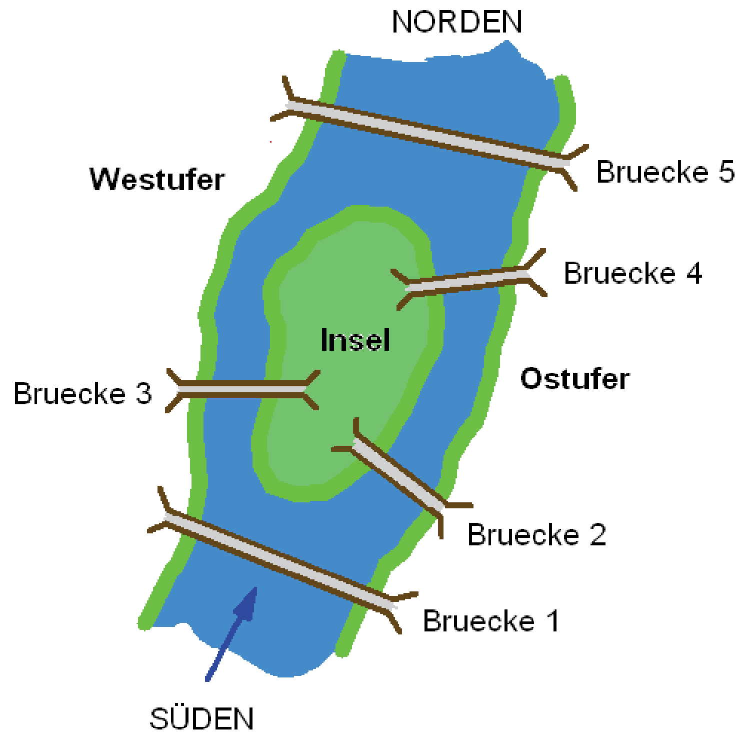


Abbildung 1.5.5 Meininger Brückenproblem

Hinter dem "Meininger Brückenproblem" verbirgt sich der in Abbildung 1.5.6 dargestellte ungerichtete Graph.

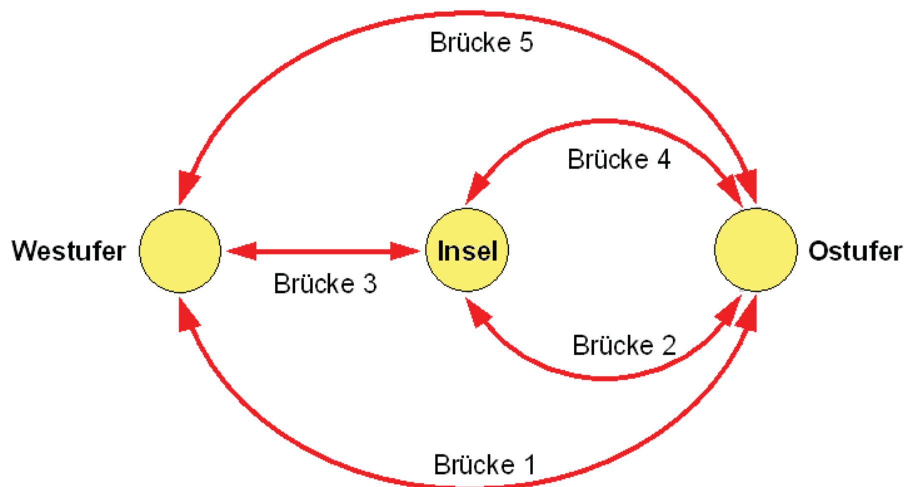


Abbildung 1.5.6 Graph zum Meininger Brückenproblem

Das Problem besteht darin, einen Weg zu suchen, bei dem jede Kante des Graphen genau einmal durchgangen wird. Wie oft dabei die Knoten besucht werden, ist uninteressant.

```
/* Programm Meininger Brueckenproblem */

ueberqueren(bruecke1,westufer,ostufer).
ueberqueren(bruecke2,ostufer,insel).
ueberqueren(bruecke3,westufer,insel).
ueberqueren(bruecke4,ostufer,insel).
ueberqueren(bruecke5,westufer,ostufer).
```

```
bruecke(BR,X,Y):- ueberqueren(BR,X,Y);
                  ueberqueren(BR,Y,X).

/* Hilfspraedikate */

heraus([K|R],K,R).
hinein(K,R,[K|R]).

enthalten(E,L):- heraus(L,E,_).
enthalten(E,L):- heraus(L,_,NL),
                 enthalten(E,NL).

umdrehen([],NL,NL).
umdrehen(L,HL,NL):- heraus(L,E,L1),
                    hinein(E,HL,L2),
                    umdrehen(L1,L2,NL).

/* Wegsuche */

weg(X,X,L,0):- umdrehen(L,[],NL),
               write(NL),nl.

weg(X,Y,L,N):- bruecke(BR,X,Z),
               not enthalten(BR,L),
               M is N - 1,
               weg(Z,Y,[BR|L],M),
               fail.

weg_beschreiten(Von):- weg(Von,_,[],5).
```

Damit das PROLOG-System alle Wege, die am Westufer beginnen, sucht und ausgibt, ist die Anfrage:

```
?- weg_beschreiten(westufer).
```

zu stellen.

Bei Anfrage:

```
?- weg_beschreiten(insel).
```

gibt das PROLOG-System alle Wege aus, die auf der Insel beginnen. Vom Ostufer aus gibt es keinen Weg. Deshalb antwortet das Prolog-System bei der Anfrage:

```
?- weg_beschreiten(ostufer).

No.
```

Das Prädikat *weg/4* kann auch ohne die Hilfsprädikate *heraus/3* und *hinein/3* implementiert werden.

```
weg(X,X,L,0):- umdrehen(L,[],NL),write(NL),nl.
weg(X,Y,L,N):- bruecke(BR,X,Z),
               not enthalten(BR,L),
               M is N - 1,
               weg(Z,Y,[BR|L],M),
               fail.
```

Im Unterschied zum "Meininger Brückenproblem" findet man im Graphen zum "Königsberger Brückenproblem" (Abbildung 1.5.7) keinen Weg.

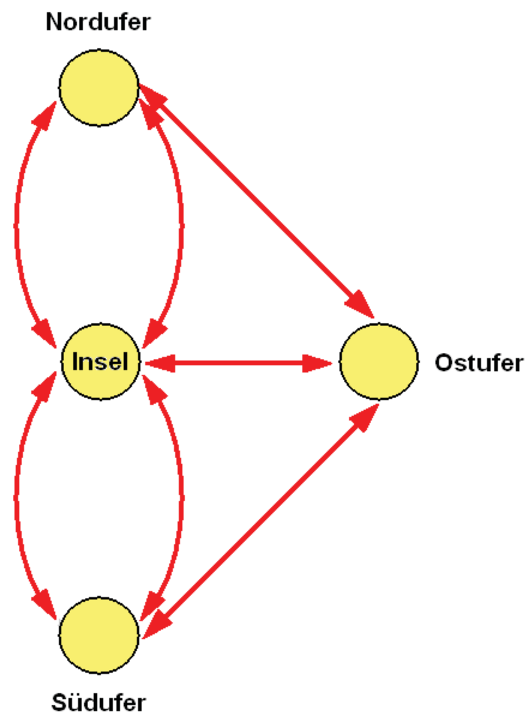


Abbildung 1.5.7 Graph zum Königsberger Brückenproblem (vgl. Schüler-Duden, Die Informatik)

1.6 Auskunftssystem

Der ungerichtete Graph, Abbildung 1.6.1, stellt ein fiktives Streckennetz einer Bahngesellschaft dar. Das Besondere an diesem Graphen besteht darin, dass die Kanten gewichtet sind. Das heißt, zwischen je zwei Knoten ist deren Entfernung angegeben.

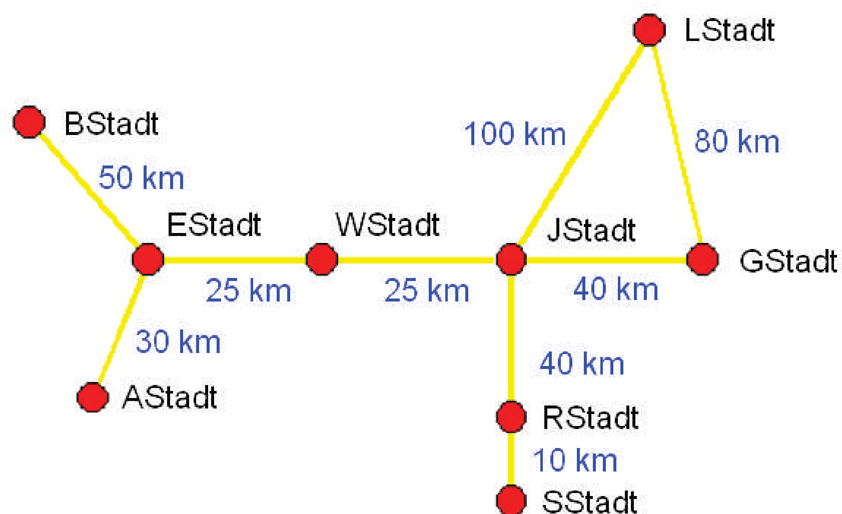


Abbildung 1.6.1 Ungerichteter Graph (fiktives Streckennetz, nicht maßstabgerecht)

Das Prädikat *strecken_von_nach/3*, das alle Kanten des ungerichteten Graphen und deren Wichtungen enthält, wird in der externen Datei `strecken.pro` gespeichert. Diese Datei kann jederzeit geändert werden, indem Strecken hinzugefügt oder gelöscht werden. Der Vor-

teil der externen Datei besteht darin, dass das Bahnauskunftssystem selbst nicht verändert werden muss.

```
/* Datei strecken.pro */

strecke_von_nach(astadt,estadt,30).
strecke_von_nach(bstadt,estadt,50).
strecke_von_nach(estadt,wstadt,25).
strecke_von_nach(wstadt,jstadt,25).
strecke_von_nach(jstadt,gstadt,40).
strecke_von_nach(jstadt,rstadt,40).
strecke_von_nach(rstadt,sstadt,10).
strecke_von_nach(jstadt,lstadt,100).
strecke_von_nach(lstadt,gstadt,80).
```

Das Programm Bahnauskunftssystem realisiert die Wegsuche in dem in der Abbildung 1.6.1 dargestellten Graphen.

```
/* Programm Bahnauskunftssystem */

start:- laden,begruessung,anleitung,bahnauskunft.

begruessung:- nl,nl,
              write('                Bahnauskunftssystem'),nl,
              write('               °°°°°°°°°°°°°°°°°°°°'),
              nl.
anleitung:- write(' Geben Sie eine Ortsangabe gefolgt von '),
            write('einem Punkt ein. '),nl,
            write(' Betätigen Sie anschließend die '),
            write('ENTER-Taste! '),nl.

laden:- reconsult('strecken.pro').   /* aktuelles Verzeichnis */

strecke(X,Y,Z):- strecke_von_nach (X,Y,Z);
                 strecke_von_nach (Y,X,Z).

/* Hilfspraedikate */

heraus([K|R],K,R).
hinein(K,R,[K|R]).

enthalten(E,L):- heraus (L,E,_).
enthalten(E,L):- heraus (L,_,NL),
                 enthalten(E,NL).

umdrehen([],NL,NL).
umdrehen(L,HL,NL):- heraus (L,E,L1),
                    hinein (E,HL,L2),
                    umdrehen(L1,L2,NL).

ausgeben([]).
ausgeben(L):- heraus(L,E,NL),
              write(' '),write(E),write(', '),
              ausgeben(NL).

* Suche nach Verbindungen */

verbindung(X,X,L,EN):- umdrehen(L,[],NL),
                       ausgeben(NL),
                       nl,write(' '),write(EN),write(' km'),nl,nl.

verbindung(X,Y,L,EN):- strecke(X,Z,W),
                       S is EN + W,
```

```
                            not enthalten(Z,L),
                            hinein(Z,L,NL),
                            verbindung(Z,Y,NL,S),fail.
bahnauskunft:- nl,write(' Startort: '),read(X),write(X),
               nl,write(' Zielort:  '),read(Y),write(Y),
               nl,nl,
               verbindung(X,Y,[X],0).
```

Das Programm wird mit der Anfrage

```
?- start.
```

gestartet. Danach erfolgen die Eingaben des Start- und des Zielorts, die jeweils mit einem Punkt abzuschließen sind. Ausgegeben werden entweder alle Verbindungen zwischen Start- und Zielort, deren jeweilige Entfernung und `No` oder nur `No`, falls keine Verbindung existiert.

Das Standardprädikat *fail/0* bewirkt, dass das PROLOG-System während der Abarbeitung des Prädikats *verbindung/4* alle Verbindungen von einem eingegebenen Start- zu einem eingegebenen Zielort sucht. Das geschieht folgendermaßen:

Findet das PROLOG-System eine Verbindung vom geforderten Start- zum Zielort, ist die Suche damit vorerst beendet und es werden alle Orte dieser Verbindung und die Entfernung zwischen Start- und Zielort in Kilometern ausgegeben. Obwohl das PROLOG-System erfolgreich war, erklärt das Standardprädikat *fail/0* die Wegsuche als erfolglos und erzwingt damit ein Backtracking. Das heißt, das PROLOG-System sucht nach einer weiteren Verbindung vom geforderten Start- zum Zielort. Durch das Standardprädikat *fail/0* wird im Prädikat *verbindung/4* das Backtracking sooft erzwungen, bis das PROLOG-System alle Verbindungen von einem eingegebenen Start- zu einem eingegebenen Zielort gefunden hat.

Das PROLOG-System findet für die Eingaben des Startortes `estadt` und des Zielortes `lstadt` zwei Verbindungen und ermittelt die dazugehörigen Entfernungen:

Bildausschnitt 1:	Bildausschnitt 2:
estadt wstadt jstadt gstadt lstadt 170 km	estadt wstadt jstadt lstadt 150 km

In der Abbildung 1.6.2 (siehe folgende Seite) ist die Suche nach allen Verbindungen von `estadt` nach `lstadt` grafisch dargestellt.

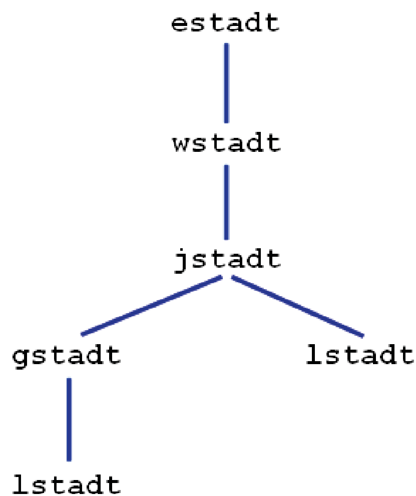


Abbildung 1.6.2 Darstellung der Verbindungen von `estadt` nach `lstadt`

Das PROLOG-Programm Bahnauskunftssystem kann erweitert werden, indem neue Strecken mithilfe des Prädikats *erweitern/0* hinzugefügt bzw. stillgelegte Strecken mithilfe des Prädikats *entfernen/0* gelöscht werden. Die Veränderungen im Prädikat *strecke_von_nach/3* müssen unbedingt mithilfe des Prädikats *sichern/0* in der Datei `strecken.pro` gesichert werden. Die Dateien `strecken.pro` und `bahnausk.pro` müssen im gleichen, aktuellen Verzeichnis stehen, um absolute Pfadangaben zu vermeiden.

```
/* Erweiterung des Programms Bahnauskunftssystem */

/* Hilfsprädikate, um retract/1 zu deklarieren */
/* (entnommen aus: autofix.pro)                 */

repeat.
repeat:- repeat.

retract(X):- repeat,
             try_retract(X,NextGoal),
             call(NextGoal).
try_retract(X,true):- retractone(X),!.
try_retract(_,(!,fail)).

erweitern:- write('Von:  '),read(V),nl,
            write('Nach: '),read(N),nl,
            write('Entfernung: '),read(E),nl,
            asserta(strecke_von_nach(V,N,E)).

entfernen:- write('Von:  '),read(V),nl,
            write('Nach: '),read(N),nl,
            write('Entfernung: '),read(E),nl,
            retract(strecke_von_nach(V,N,E)).

sichern:-   tell('strecken.pro'),
            listing(strecke_von_nach),
            told.
```

1.7 Manipulation symbolischer Ausdrücke

Im Programm Differenzialrechnung sind die Ableitungen der elementaren Funktionen

$f(x) = c$	$f'(x) = 0$	
$f(x) = x^n$	$f'(x) = n * x^{(n-1)}$	(n > 0, n Element der ganzen Zahlen)
$f(x) = e^x$	$f'(x) = e^x$	
$f(x) = \sin(x)$	$f'(x) = \cos(x)$	
$f(x) = \cos(x)$	$f'(x) = -\sin(x)$	

und die Differenziationsregeln

$f(x) = c * u(x)$	$f'(x) = c * u'(x)$
$f(x) = u(x) + v(x)$	$f'(x) = u'(x) + v'(x)$
$f(x) = u(x) - v(x)$	$f'(x) = u'(x) - v'(x)$
$f(x) = u(x) * v(x)$	$f'(x) = u'(x) * v(x) + u(x) * v'(x)$
$f(x) = u(x) / v(x)$	$f'(x) = (u'(x) * v(x) - u(x) * v'(x)) / v(x)^2$

als Fakten bzw. Regeln implementiert.

```
/* Programm Differenzialrechnung */

diff:- op(10,yfx,^).     /* Definition des Operatorsymbols ^ */

/* Ableitungen */

abl(C,0):- number(C).
abl(x^N,N * x^M):- N>0,M is N - 1.
abl(e^x,e^x).
abl(sin(x),cos(x)).
abl(cos(x),-sin(x)).

/* Differentiationsregeln */

abl(C*U,C*DU):- number(C),
                abl(U,DU).

abl(U+V,DU+DV):-  abl(U,DU),
                  abl(V,DV).

abl(U-V,DU-DV):-  abl(U,DU),
                  abl(V,DV).

abl(U*V,DU*V+U*DV):-  abl(U,DU),
                      abl(V,DV).

abl(U/V,(DU*V-U*DV)/V^2):- abl(U,DU),
                           abl(V,DV).
```

Da das Prädikat *number/1* in Fix-PROLOG als Standardprädikat nicht verfügbar ist, muss es als Hilfsprädikat implementiert werden.

```
/* Hilfsprädikat */

number(Z):- integer(Z).
```

Das Prädikat *integer/1* wird verwendet, weil in Fix-PROLOG Zahlen nur als ganze Zahlen verarbeitet werden können.

Um mit dem Programm symbolisch differenzieren zu können, muss einmalig die Anfrage

```
?- diff.
```

gestellt werden.

Danach erhält man beispielsweise die Ableitung der Funktion f(x) = x^3 mittels Anfrage

```
?- abl(x^3,A).
```

Die Antworten des Prolog-Systems lauten

```
A = 3 * x ^ 2;
No
```

Die Ableitung der Funktion f(x) = sin(x) * cos(x) erhält man, wenn die Anfrage

```
?- abl(sin(x)*cos(x),D).
```

gestellt wird. Das Prolog-System antwortet:

```
A = cos(x) * cos(x) + sin(x) * (-sin(x));
No
```

Die Kettenregel lässt sich nicht in allgemeiner Form in ein PROLOG-Programm implementieren. Es muss stets die äußere Funktion vorgegeben sein.

```
/* Erweiterung des Programms Differenzialrechnung */

abl(U^N,N*U^M*DU):- N>0,M is N - 1,
                    abl(U,DU).

abl(e^U,e^U*DU):- abl(U,DU).
abl(sin(U),cos(U)*DU):- abl(U,DU).
abl(cos(U),-sin(U)*DU):- abl(U,DU).
```

Um die Ableitung der verketteten Funktion f(x) = sin(x^3) zu bilden, ist die Anfrage

```
?- abl(sin(x^3),A).
```

zu stellen. Als Antworten gibt das Prolog-System

```
A = cos(x^3) * 3 * x^2;
No
```

aus.

1.8 Dynamisches Verändern der Wissensbasis

Die Wissensbasis, d.h. alle Fakten und Regeln, kann während der Laufzeit eines PROLOG-Programms verändert werden. Dieser Vorgang soll an einem in PROLOG programmierten Stack (Stapel) erläutert werden.

Ein Stack ist ein LIFO-Speicher (last in first out), das heißt, dass jeweils nur das zuletzt in den Stack eingefügte Element aus dem Stack entfernt werden kann.

Dem PROLOG-Programm Stack liegt folgende Spezifikation zugrunde:

Die Elemente dieses Stacks sind Zahlen. Die Operationen des Stacks sind:

make empty	erzeugt den leeren Stack, Abbildung 1.8.1
push(E)	fügt die Zahl E als letztes (oberstes) Element in den Stacks ein, Abbildung 1.8.2
pop	entfernt das letzte (oberstes) Element aus dem Stack, Abbildung 1.8.3
top(E)	liefert die Zahl E, die als letztes (oberstes) Element in den Stack eingefügt wurde, Abbildung 1.8.4
empty	ist wahr, wenn der Stack leer ist, sonst falsch, Abbildung 1.8.5

Abbildung 1.8.1

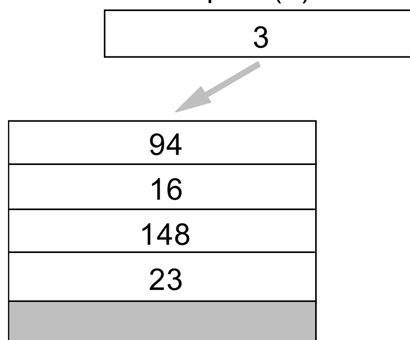


Abbildung 1.8.2

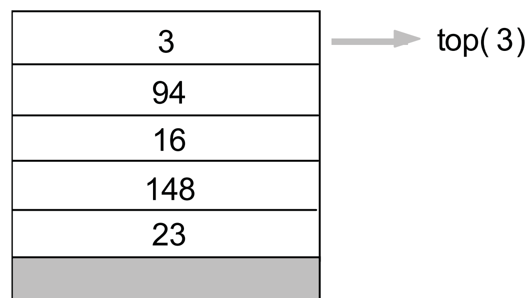


Abbildung 1.8.4

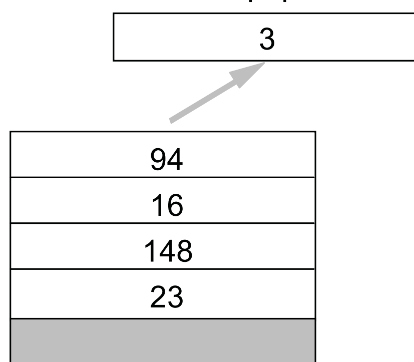


Abbildung 1.8.3

Abbildung 1.8.5

```
/* Programm Stack */

makeempty:- abolish(stackelement/1).
push(E):-   number(E),
            asserta(stackelement(E)).
top(E):-    stackelement(E),!.
pop:-       retract(stackelement(_)),!.
empty:-     not stackelement(_).
```

Da Fix-PROLOG die Standardprädikate *abolish/1* und *number/1* nicht enthält, muss das Programm Stack durch Hilfsprädikate ergänzt werden.

```
/* Hilfspraedikate */
abolish(stackelement/1):- retractall(stackelement(0));true.
number(X):- integer(X).
```

Durch das Standardprädikat *true/0* ist das Hilfsprädikat *abolish/1* bei jeder Anfrage erfolgreich.

Das Prädikat *makeempty/0* dient dem Zweck, das Prädikat *stackelement/1* aus der Wissensbasis zu löschen.

Das Prädikat *push/1* bewirkt, dass in der Wissensbasis ein neues Faktum an den Anfang der Folge der Fakten des Prädikats *stackelement/1* eingefügt wird.

Das Prädikat *pop/0* bewirkt, dass das erste Faktum aus der Folge der Fakten des Prädikats *stackelement/1* aus der Wissensbasis entfernt wird.

Das Prädikat *top/0* liefert die Zahl, die im ersten Faktum der Folge der Fakten des Prädikats *stackelement/1* enthalten ist oder `No`. Das Standardprädikat *cut/0* verhindert, dass mehr als ein Element vom Stack geholt wird.

Das Prädikat *empty/0* liefert `Yes`, falls das Prädikat *stackelement/1* nicht in der Wissensbasis enthalten ist, ansonsten `No`.

Am Beispiel der folgenden Anfragen soll die Arbeitsweise des Programms Stack erläutert werden:

Anfragen und Antworten	Auszug aus der Wissensbasis
`?- makeempty.` `Yes`	`/* Prädikat stackelement/1 nicht enthalten */`
`?- push(1).` `Yes`	`stackelement(1).`
`?- push(2).` `Yes`	`stackelement(2).` `stackelement(1).`

```
?- push(3).                   stackelement(3).
Yes                           stackelement(2).
                              stackelement(1).

?- not empty,top(E),pop.      stackelement(2).
E = 3                         stackelement(1).

?- not empty,top(E),pop.      stackelement(1).
E = 2

?- push(4).                   stackelement(4).
Yes                           stackelement(1).

?- not empty,top(E),pop.      stackelement(1).
E = 4

?- not empty,top(E),pop.      /* Prädikat stackelement/1 nicht enthalten */
E = 1

?- not empty,top(E),pop.      /* Prädikat stackelement/1 nicht enthalten */
No
```

1.9 Parser und Interpreter

1.9.1 Parser

Gegeben ist die Grammatik einer Sprache L mit der Menge der Nichtterminalsymbole

N = {BEZEICHNER, GROSSBUCHSTABE, KLEINBUCHSTABE, ZIFFER},

der Menge der Terminalsymbole

T = {A, B, C, a, b, c, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9},

der Menge der Produktionsregeln

P = { BEZEICHNER =
GROSSBUCHSTABE {GROSSBUCHSTABE | KLEINBUCHSTABE | ZIFFER},
GROSSBUCHSTABE = 'A' | 'B' | 'C',
KLEINBUCHSTABE = 'a' | 'b' | 'c',
ZIFFER = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'}

und

S = BEZEICHNER

als Startsymbol.

Mithilfe dieser Grammatik lässt sich jedes Wort der Sprache L ableiten. Das Ableiten eines Wortes beginnt mit dem Startsymbol, das ein Nichtterminalsymbol ist. Im weiteren Verlauf des Ableitens werden mithilfe der Produktionsregeln Nichtterminalsymbole durch Nichtterminalsymbole oder Terminalsymbole ersetzt. Das Ableiten endet, wenn alle Nichtterminalsymbole durch Terminalsymbole ersetzt sind. Beispielsweise sind Bc3 und Acbac92b Wörter der Sprache L.

Die Produktionsregeln der Sprache L können grafisch durch Syntaxdiagramme dargestellt werden.

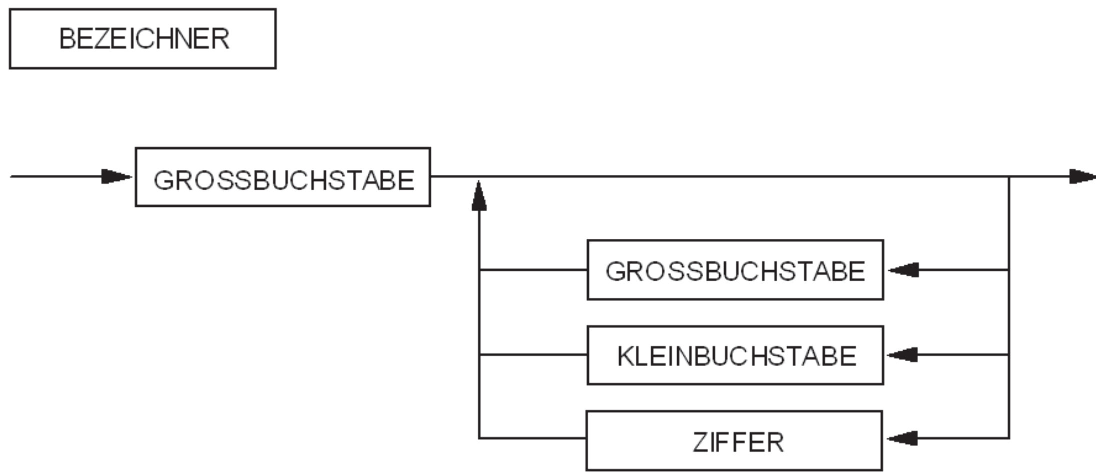


Abbildung 1.9.1.1 Syntaxdiagramme der Sprache L

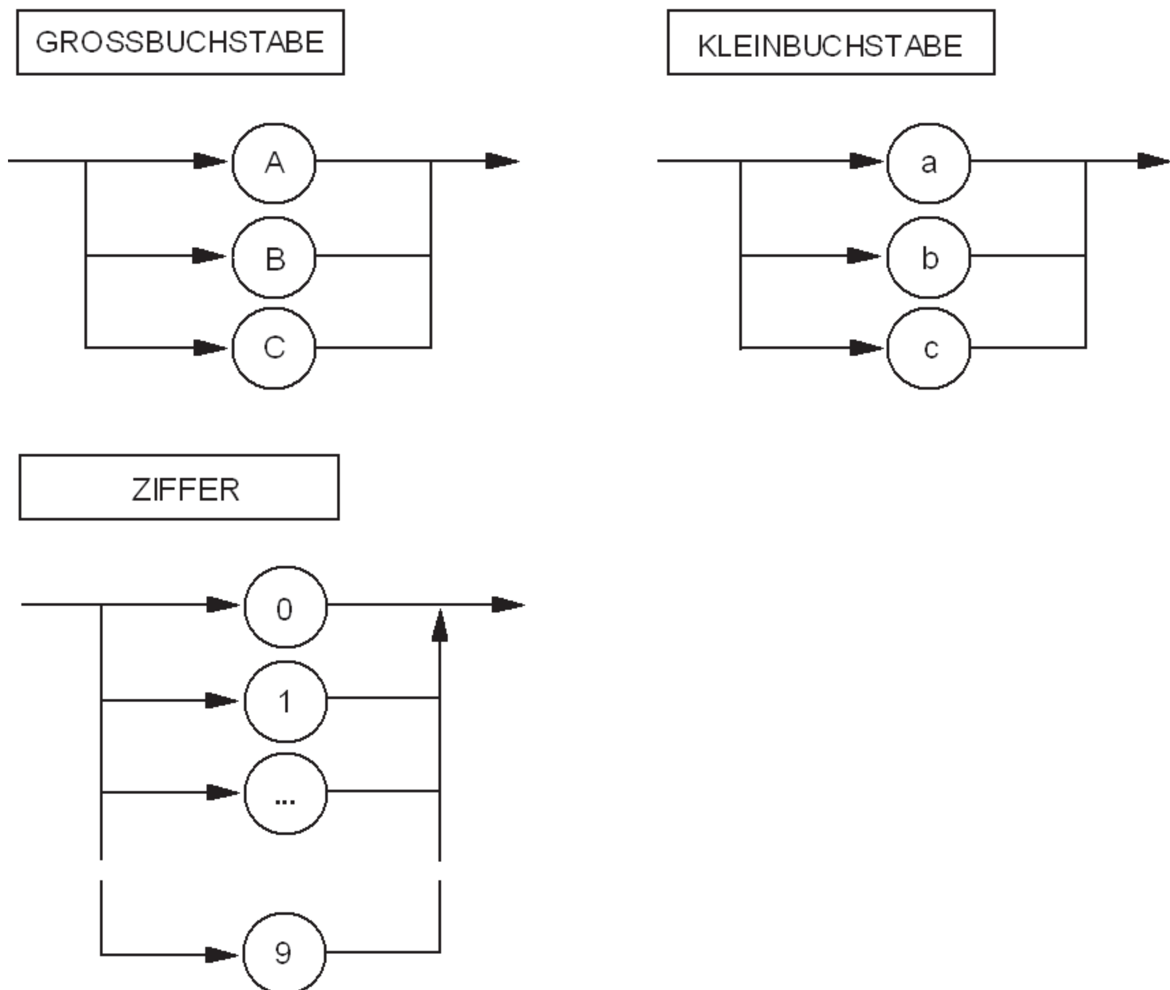


Abbildung 1.9.1.1 Syntaxdiagramme der Sprache L (Fortsetzung)

Jedes Wort der Sprache L ist ein in PROLOG gültiger Bezeichner, jedoch nicht jeder PROLOG-Bezeichner ist ein Wort dieser Sprache.

Das Programm `Parser` findet heraus, ob der bei einer Anfrage angegebene Bezeichner syntaktisch korrekt ist.

```
/* Programm Parser */

grossbuchstabe('A').
grossbuchstabe('B').
grossbuchstabe('C').

kleinbuchstabe('a').
kleinbuchstabe('b').
kleinbuchstabe('c').

ziffer('0').
ziffer('1').
ziffer('2').
ziffer('3').
ziffer('4').
ziffer('5').
ziffer('6').
ziffer('7').
ziffer('8').
ziffer('9').

/* Hilfspraedikat */

heraus([K|R],K,R).
bezeichner(L):- heraus(L,K,R),
                grossbuchstabe(K),
                wiederhole(R).

wiederhole([]).
wiederhole(L):- heraus(L,K,R),
                grossbuchstabe(K),
                wiederhole(R).

wiederhole(L):- heraus(L,K,R),
                kleinbuchstabe(K),
                wiederhole(R).

wiederhole(L):- heraus(L,K,R),
                ziffer(K),
                wiederhole(R).
```

In einer Anfrage ist der Bezeichner als Prolog-Liste anzugeben.

```
?- bezeichner(['B','c','3']).
```

Soll das Hilfsprädikat *heraus/3* im Programm nicht verwendet werden, so können die Prädikate *bezeichner/1* und *wiederholen/1* wie folgt formuliert werden:

```
bezeichner([K|R]):- grossbuchstabe(K),
                    wiederhole(R).

wiederhole([ ]).
wiederhole([K|R]):- grossbuchstabe(K),
                    wiederhole(R).

wiederhole([K|R]):- kleinbuchstabe(K),
                    wiederhole(R).

wiederhole([K|R]):- ziffer(K),
                    wiederhole(R).
```

Im Buch »Compilerbau« nutzt NIKLAUS WIRTH als einführendes Beispiel eine Sprache, der die folgende Grammatik zugrunde liegt:

Die Menge der Nichtterminalsymbole ist

N = {SATZ, SUBJEKT, PRÄDIKAT},

die Menge der Terminalsymbole ist

T = {Katze, Hund, essen, schlafen },

die Menge der Produktionsregeln ist

P = { SATZ = SUBJEKT PRÄDIKAT,
SUBJEKT = 'Katze' | 'Hund',
PRÄDIKAT = 'essen' | 'schlafen'}

und

S = SATZ

ist das Startsymbol.

In dieser Sprache können maximal vier Sätze gebildet werden.

Katzen essen

Katzen schlafen

Hunde essen

Hunde schlafen

Das Prolog-Programm `Sprache` prüft Sätze auf syntaktische Korrektheit.

```
/* Programm Sprache */

subjekt(katze).
subjekt(hund).
praedikat(essen).
praedikat(schlafen).

satz([S,P|_]):- subjekt(S),
                praedikat(P).
```

In den Anfragen sind die Sätze in PROLOG-Listen anzugeben.

```
?- parser([katze,essen]).
?- parser([katzen,schlafen]).
?- parser([hunde,essen]).
?- parser([hunde,schlafen]).
```

Auf die Anfrage

```
?- parser([schlafen,hunde]).
```

antwortet das Prolog-System `No`.

1.9.2 Interpreter

Gegeben ist eine Syntax zum Darstellen von Zahlen im Dualsystem.

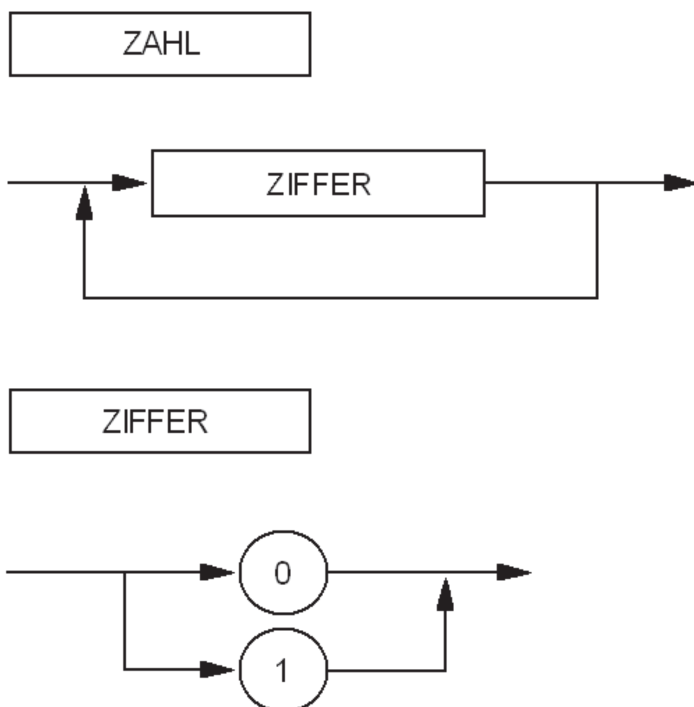


Abbildung 1.9.2.1 Syntaxdiagramme zum Darstellen von vorzeichenlosen ganzen Dualzahlen

Dualzahlen können in Dezimalzahlen konvertiert werden. Im nachstehenden Beispiel wird gezeigt, wie nach HORNER die Konvertierung erfolgen kann:

$$11011_{dual} = 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0$$
$$= (((1*2 + 1)*2 + 0)*2 + 1)*2 + 1$$
$$= 26_{dez}$$

Das Programm `Dual-Dezimal-Interpreter` konvertiert bei Anfrage eine Zahl aus dem Dual- in das Dezimalsystem. Die Zahl wird auf ihre syntaktische Korrektheit geprüft. Ist die Zahl korrekt, wird sie konvertiert.

```
/* Programm Dual-Dezimal-Interpreter */

/* Syntax der Dualzahlen */

ziffer(0).
ziffer(1).

zahl([]).
zahl([K|R]):- ziffer(K),
              zahl(R).

/* Parser */

parser(Z):- Z\==[],
            zahl(Z).
```

```
/* Interpreter */

konvertieren([K|[]],Z,E):- E is Z+K.
konvertieren([K|R],Z,E) :- H is (Z+K)*2,
                           konvertieren(R,H,E).

interpreter_dude(Du,De):-  parser(Du),
                           konvertieren(Du,0,De).
```

Auf die Anfrage

```
?- interpeter_dude([1,1,0,1,1],E).
```

antwortet das Prolog-System

```
E = 26
```

und auf

```
?- interpeter_dude([1,2,1,1,0],E).
```

antwortet das Prolog-System

```
No
```

Das komplexere Programm `Baum` ermöglicht das preorder Traversieren eines binären Baums. Dem Interpreter liegt die Sprache *PreTrav* mit der in Abbildung 1.9.2.2 dargestellten Syntax zugrunde. Mit der Anweisung `start` beginnt und mit der Anweisung `stopp` endet das Traversieren. Mithilfe der Anweisungen `abwickeln` und `aufwickeln` wird Knoten für Knoten preorder traversiert. Beginnend beim Wurzelknoten wird zum linken Nachbarknoten gegangen und dabei ein "Faden" abgewickelt.

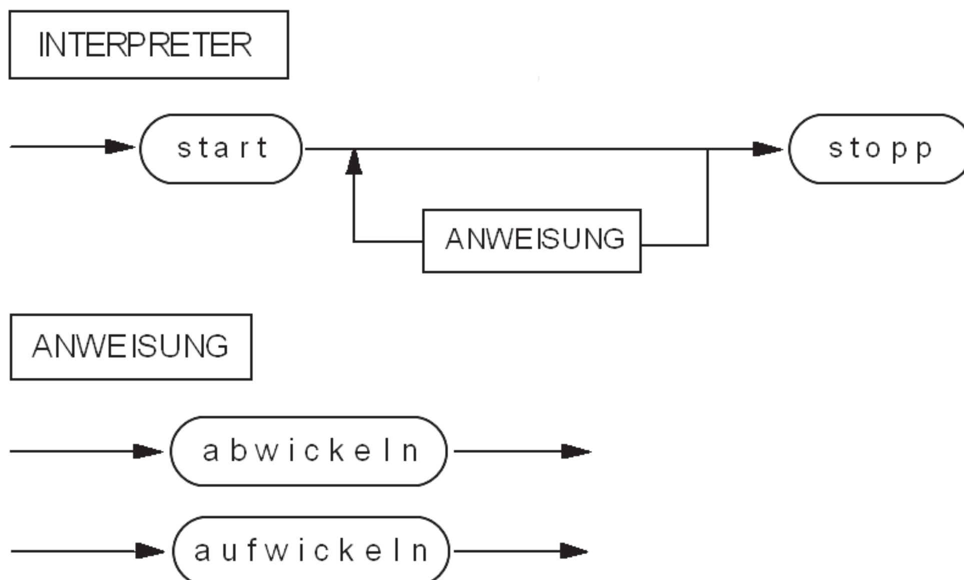


Abbildung 1.9.2.2 Syntaxdiagramme der Sprache PreTrav

Besitzt dieser Knoten wiederum einen linken Nachbarknoten, so wird dieser besucht und dabei der "Faden" weiter abgewickelt. Es wird solange zu den linken Nachbarknoten weitergegangen und dabei der "Faden" abgewickelt, bis der Knoten erreicht wird, der keinen linken Nachbarknoten hat. Besitzt dieser Knoten einen rechten Nachbarknoten, wird der "Faden"

weiter abgewickelt und dieser Knoten besucht. Besitzt ein Knoten weder einen linken noch einen rechten Nachbarknoten, so muss unter Aufwickeln des "Fadens" zum vorhergehenden Knoten zurückgegangen werden. Für den Fall, dass der linke Teilbaum eines Knotens traversiert wurde und der Knoten keinen rechten Nachbarknoten besitzt, muss ebenfalls unter Aufwickeln des "Fadens" zum vorhergehenden Knoten zurückgegangen werden. Der Baum ist komplett traversiert, wenn jeder Knoten mindestens einmal besucht und wieder der Wurzelknoten erreicht ist.

Soll der Baum nicht komplett traversiert werden, kann die Arbeit des Interpreters durch die Anweisung `stopp` an jeder beliebigen Stelle beendet werden.

Die Abbildung 1.9.2.3 stellt den binären Baum dar, der im Programm `BinBaum` implementiert ist. Das Programm ermöglicht bei Anfrage

```
?- interpreter.
```

das schrittweise preorder Traversieren eines binären Baums unter Verwendung der Sprache PreTrav. Der in der Datei `bbaum.pro` angegebene binäre Baum ist exemplarisch.
In das PROLOG-Programm `BinBaum` können andere binären Bäume aus entsprechenden Dateien geladen werden.

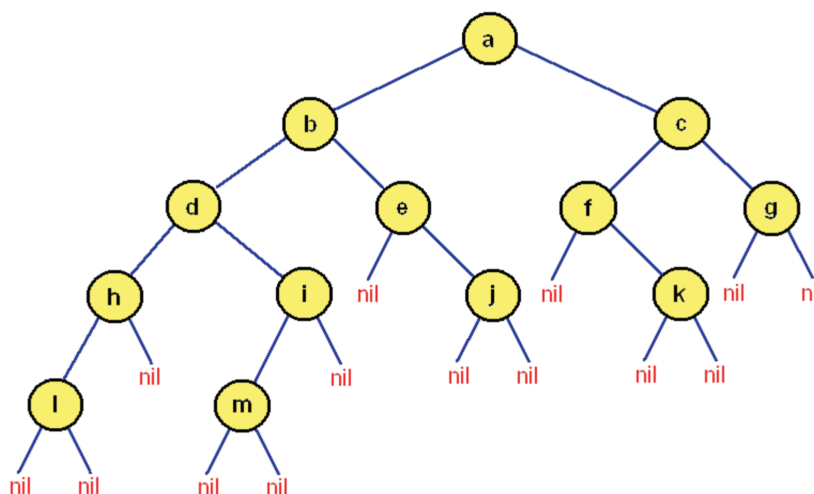


Abbildung 1.9.2.3 Binärer Baum des Programms Interpreter

```
/* BinBaum.pro - ein exemplarischer binaerer Baum */

wurzel(a).
baum(a,b,c).
baum(b,d,e).
baum(d,h,i).
baum(h,l,nil).
baum(l,nil,nil).
baum(i,m,nil).
baum(m,nil,nil).
baum(e,nil,j).
baum(j,nil,nil).
baum(c,f,g).
```

```
baum(f,nil,k).
baum(k,nil,nil).
baum(g,nil,nil).

/* Programm Interpreter */

/* Programm wird mit ?- interpreter. gestartet */

interpreter:- consult('bbaum.pro'),
              read(A1),
              A1==start,
              interpretiere(A1),
              repeat,
              read(A),
              interpretiere(A),
              /* until */
              (A==start;A==stopp),
              interpretiere(stopp),
              !,fail.

/* Beginn des Traversieren */

interpretiere(start):- wurzel(K),
                       assertz(knoten(K)),
                       asserta(besucht(K)).
```

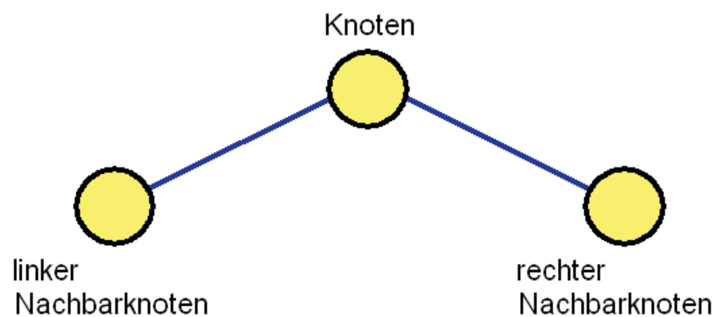


Abbildung 1.9.2.4 Knoten mit Nachbarknoten

```
/* Besuchen des linken Nachbarknotens */

interpretiere(abwickeln):- /* Knoten → linker Nachbarknoten */
                           ((knoten(K),baum(K,LNK,_),
                           LNK\==nil,
                           not besucht(LNK),
                           write(K),write(' → '),write(LNK),nl,
                           asserta(besucht(LNK)),
                           retract(knoten(K)),
                           assertz(knoten(LNK)));
                           /* Fall: nil links */
                           (knoten(K),baum(K,LNK,_),
                           LNK==nil,
                           not leer(K,links),
                           write(LNK),write(' links'),nl,
                           asserta(leer(K,links)))),!.
```

```
/* Besuchen des rechten Nachbarknotens */

interpretiere(abwickeln):- /* Knoten → rechter Nachbarknoten */
                           ((knoten(K),baum(K,_,RNK),
                           RNK\==nil,
                           not besucht(RNK),
                           write(K),write(' → '),write(RNK),nl,
                           asserta(besucht(RNK)),
                           retract(knoten(K)),
                           assertz(knoten(RNK)));
                           /* Fall: nil rechts */
                           (knoten(K),baum(K,_,RNK),
                           RNK==nil,
                           not leer(K,rechts),
                           write(RNK),write(' rechts'),nl,
                           asserta(leer(K,rechts)))),!.

/* Zurueckkehren vom Nachbarnoten zum Knoten */

interpretiere(aufwickeln):- /* linker Nachbarknoten → Knoten */
                            ((knoten(LNK),baum(K,LNK,_),
                            write(K),write(' ← '),write(LNK),nl,
                            retract(knoten(LNK)),
                            assertz(knoten(K)));
                            /* rechter Nachbarknoten → Knoten */
                            (knoten(RNK),baum(K,_,RNK),
                            write(K),write(' ← '),write(RNK),nl,
                            retract(knoten(RNK)),
                            assertz(knoten(K)))),!.

/* Beenden des Traversierens */

interpretiere(stopp):- abolish(knoten/1),
                       abolish(besucht/1),
                       abolish(leer/2).
```

Fix-PROLOG besitzt kein Standardprädikat *abolish/1*. Deshalb wird das nachstehende Hilfsprädikat benötigt.

```
/* Hilfspraedikat */

abolish(knoten/1):-  retractall(knoten(0));
                     true.
abolish(besucht/1):- retractall(besucht(0));
                     true.
abolish(leer/2):-    retractall(leer(0,0));
                     true.
```

Das Standardprädikat *true/0* wird verwendet, damit das Hilfsprädikat *abolish/1* bei j e d e r Anfrage erfolgreich ist.

Zum preorder Traversieren des Baums aus Abbildung 1.9.2.3 müssen nach dem Programmstart folgende Eingaben ausgeführt werden (siehe Kopiervorlage auf der folgenden Seite):

start.
abwickeln.
a → b
abwickeln.
b → d
abwickeln.
d → h
abwickeln.
h → l
abwickeln.
nil links
abwickeln.
nil rechts
abwickeln.

aufwickeln.
h ← l
abwickeln.
nil rechts
abwickeln.

aufwickeln.
d ← h
abwickeln.
d → i
abwickeln.
i → m
abwickeln.
nil links
abwickeln.
nil rechts
abwickeln.

aufwickeln.
i ← m
abwickeln.
nil rechts
abwickeln.

aufwickeln.
d ← i
abwickeln.

aufwickeln.
b ← d
abwickeln.
b → e
abwickeln.
nil links
abwickeln.
e → j
abwickeln.
nil links
abwickeln.
nil rechts
abwickeln.

aufwickeln.
e ← j
abwickeln.

aufwickeln.
b ← e
abwickeln.

aufwickeln.
a ← b
abwickeln.
a → c
abwickeln.
c → f
abwickeln.
nil links
abwickeln.
f → k
abwickeln.
nil links
abwickeln.
nil rechts
abwickeln.

aufwickeln.
f ← k
abwickeln.

aufwickeln.
c ← f
abwickeln.
c → g
abwickeln.
nil links
abwickeln.
nil rechts
abwickeln.

aufwickeln.
c ← g
abwickeln.

aufwickeln.
a ← c
abwickeln.

aufwickeln.

stopp.

Die Knoten des binären Baums wurden in der Reihenfolge

a → b → d → h → l → i → m → e → j → c → f → k → g besucht.

1.10 Ein kleines Expertensystem

Die folgenden Ausführungen sind als Grundlage für das Lösen einer komplexeren Aufgabenstellung gedacht.

Aufgabenstellung:

Entwerfen und Implementieren eines Auskunfts- und Beratungssystems in PROLOG, das Schülerinnen und Schüler der Klassenstufe 10 der gymnasialen Oberstufe bei der Wahl der Leistungs- und Grundfächer für das Kurssystem an ihrem Gymnasium unterstützt.

Teilaufgaben:

(1) Das Expertenwissen wird aus den Verwaltungsvorschriften entnommen und durch das Wissen und die Erfahrungen des Oberstufenkoordinators ergänzt.

1. Leistungsfach:	entweder Deutsch oder Mathematik
2. Leistungsfach:	…
1. Grundfach:	je nach Wahl des ersten Leistungsfachs entweder Mathematik oder Deutsch
2. Grundfach:	…
…	…
9. Grundfach:	…
Prüfungsbereich:	…

(2) Das Expertenwissen wird so weit wie möglich in Form von Fakten und Regeln in die Wissensbasis übertragen. Dabei wird unter anderem auch eine Datenbank angelegt.

Tabelle Schüler:

Schülernr.	Familienname	Vorname
12	Maier	Peter
7	Kunze	Ines
18	Schmidt	Sabine
Xx	…	…

```
schueler(12,'Maier','Peter').
schueler(7,'Kunze','Ines').
...
```

Tabelle erstes Leistungsfach:

Fachnr.	Unterrichtsfach
1	Deutsch
2	Mathematik

```
leistungsfach1(1,'Deutsch').
leistungsfach1(2,'Mathematik').
...
```

Tabelle zweites Leistungsfach:

Fachnr.	Unterrichtsfach
8	Physik
14	Geschichte
...	...

```
leistungsfach2(8,'Physik').
leistungsfach2(14,'Geschichte').
...
```

Tabelle erstes Grundfach:

Fachnr.	Unterrichtsfach
1	Deutsch
2	Mathematik

```
grundfach1(1,'Mathematik').
grundfach1(2,'Deutsch').
...
```

Tabelle neuntes Grundfach:

Fachnr.	Unterrichtsfach
16	Sport

```
grundfach9(16,'Sport').
...
wahl_leistungsfach1(SNr,Ufach):- ...
wahl_leistungsfach2(SNr,Ufach):- ...
wahl_grundfach1(SNr,Ufach):- ...
...
wahl_grundfach9(SNr,Ufach):- ...
```

(3) Dem Expertensystem wird ein Erklärungsteil mit Hilfestellungen hinzugefügt. In diesen Hilfestellungen werden dem Schüler wesentliche Folgerungen, die das Expertensystem trifft, verständlich erklärt. Das Expertensystem hat lediglich eine beratende Funktion. Der Schüler kann das Expertensystem hilfreich zur sicheren Entscheidungsfindung in den verschiedenen Fällen zu Rate ziehen. Die Entscheidungen muss der Schüler letztendlich jedoch selbst treffen.

Exkurs:

Das folgende Programm mit Erläuterungen soll die Arbeitsweise des Erklärungsteils eines Expertensystems veranschaulichen.

```
/* Auszug aus der Wissensbasis eines Expertensystems */

koerper(holz).
leichter_als_wasser(holz).
schwimmt_im_wasser(X):- koerper(X),
                        leichter_als_wasser(X).
```

Auf die Anfrage, ob Holz schwimmt, wird `Yes` ausgegeben.

Die Antwort kommt dadurch zustande, dass das Prolog-System eine Regel rückwärtsverkettet abarbeitet (engl. backward chaining):

Das Prolog-System geht von der Annahme aus, dass Holz im Wasser schwimmt. Durch die Regel ergeben sich die Annahmen, dass Holz ein Körper und leichter als Wasser ist. Diese Annahmen werden durch Fakten der Wissensbasis bestätigt. Da die Annahmen bestätigt wurden und Prolog-Regeln an sich erfüllbar sind, wird die Annahme, dass Holz im Wasser schwimmt, bestätigt.

$a \wedge b,\ a \wedge b \Rightarrow c \models c$ (modus ponens).

Das PROLOG-System antwortet mit `Yes`.

Mit Hilfe des Standardprädikats clause/2 kann ein Programm geschrieben werden, das beim Abarbeiten von Regeln alle bestätigten Annahmen des Prolog-Systems ausgibt. Das folgende Prädikat bestaetigt/1 ist für Regeln der Form a:- b, c anwendbar.

```
/* Veranschaulichen des Backward Chainings */

bestaetigt(X):- clause(X,Y),
                Y =.[ _,B,C|_ ],
                clause(B,U),
                write(B),write(' = '),write(U),nl,
                clause(C,W),
                write(C), write(' = '),write(W),nl.

?- bestaetigt(schwimmt_im_wasser(holz)).
```

Die Antworten lauten:

```
koerper(holz)= true;
leichter_als_wasser(holz)= true;
Yes
```

Hinweis: `Y` erhält den Wert `leichter_als_wasser(holz)` und `Y = .L` bewirkt, dass `L` den Wert `[,,koerper(holz),leichter_als_wasser(holz)]` erhält.

Auf die Anfrage

```
?- bestaetigt(schwimmt_im_wasser(stahl)).
```

wird `No` geantwortet.

(4) Das Expertensystem erhält einen D i a l o g t e i l für umgangssprachliche Kommunikation mit dem Schüler.

Zum Beispiel kann folgende Frage gestellt werden:

Kann Musik als Leistungsfach gewählt werden?

Ein S c a n n e r überträgt die Wörter der Frage in eine PROLOG-Liste:

```
['Kann','Musik','als','Leistungsfach','ausgewählt','werden','?']
```

Ein F i l t e r entfernt aus der Liste alle unnötigen Wörter und Zeichen:

```
['Musik','Leistungsfach']
```

Ein P a r s e r bringt die Frage in ein Schema. Der Parser erkennt dabei Synonyme und grammatische Formen. Mit Hilfe dieses Schemas kann dann das PROLOG-System die Frage beantworten. Das Schema für die oben gestellte Frage wäre beispielsweise:

```
frage:- schema(leistungsfach,'Musik').

schema(leistungsfach,LF):-
                          leistungsfach1(_,LF);leistungsfach2(_,LF).
```

(5) Dem Expertensystem wird ein W i s s e n s e r w e r b s t e i l hinzugefügt. Ändern sich zum Beispiel die Kriterien zur Einwahl in das Kurssystem, muss der Anwender die Möglichkeiten haben, die Änderungen dem Expertensystem mitteilen zu können. Das Expertensystem muss selbstständig in der Lage sein, diese Änderungen in der Wissensbasis vorzunehmen.

Die Teilaufgaben 2 bis 5 können jeweils von einer S c h ü l e r g r u p p e bearbeitet werden. Während der Laufzeit des Expertensystems besteht die Möglichkeit, Prädikate, beispielsweise die nachstehenden Tabellen, in die Wissensbasis aufzunehmen, zu ändern oder zu löschen (dynamisches Verändern der Wissensbasis).

Tabelle Belegung erstes Leistungsfach:

Schülernr.	Fachnr.
12	1
…	…

Tabelle Belegung neuntes Grundfach:

Schülernr.	Fachnr.
12	14
…	…

Die Veränderungen, die in den Prädikaten vorgenommen werden, müssen gesichert werden.
Beispiel:

```
sichern:- tell('schuedat.pro'),listing(schueler),told.
```

Mit Hilfe der Anfrage

```
?- sichern.
```

wird das Prädikat *schueler/3* in die externe Datei `schuedat.pro` kopiert. Es ist zu beachten, dass in dieser Datei n u r das Prädikat *schueler/3* steht.

1.11 Resolutionsprinzip, Unifikation und Backtracking

1.11.1. Resolutionsprinzip

Gegeben ist das folgende PROLOG-Programm mit den Prädikaten *m/0*, *n/0*, *p/0*, *q/0*

```
m.
n.
p.
q:- m,n,p.
```

an das die Anfrage ?- q. gestellt wird. Für das PROLOG-System ist die Anfrage (Ziel) eine Behauptung, die bewiesen werden muss. Um zu zeigen, wie das PROLOG-System den Beweis führt, wird die HORN-Logik verwendet.

Die HORN-Logik (Alfred HORN) bildet die logische Grundlage der Programmiersprache PROLOG. In der HORN-Logik wird eine Klausel (Faktum, Regel, Anfrage) entweder als Literal oder als eine Disjunktion von Literalen dargestellt. Die Disjunktion von Literalen darf höchstens ein Literal in nicht negierter Form enthalten. Den Fakten des gegebenen PROLOG-Programms entsprechen die Literale m, n, und p, der Regel entspricht die Disjunktion:

$q \lor \neg m \lor \neg n \lor \neg p$

von Literalen und die Anfrage wird negiert und entspricht dem Literal $\neg q$.

PROLOG	HORN-Logik
`m.`	m
`n.`	n
`p.`	p
`q:- p, n, m.`	$q \lor \neg m \lor \neg n \lor \neg p$
`?- q.`	$\neg q$

Tabelle 1.11.1.1 Darstellung der Klauseln in der HORN-Logik

Das PROLOG-System führt einen indirekten Beweis derart, dass ausgehend von der negierten Anfrage in der Klauselmenge M = $\{\neg q, m, n, p, q \lor \neg m \lor \neg n \lor \neg p\}$ ein Widerspruch (die leere Klausel) gefolgert wird. Dem Beweis liegt die Schlussregel (1) zugrunde:

(1) $\neg q \Rightarrow \text{falsch} \vDash q$ (»aus $\neg q \Rightarrow$ falsch folgt q«)

Um zu beweisen, dass q wahr ist, muss das PROLOG-System nachweisen, dass die Implikation $\neg q \Rightarrow$ falsch wahr ist. Den Nachweis führt das PROLOG-System nach dem Resolutionsprinzips:

Es werden in der Klauselmenge M = $\{\neg q, m, n, p, q \lor \neg m \lor \neg n \lor \neg p\}$ zwei Klauseln gesucht. Jede der beiden Klauseln muss mindestens ein Literal enthalten und ein Literal der einen Klausel muss die Negation eines Literals der anderen Klausel sein. Werden zwei solche Klauseln gefunden, wird aus ihnen nach Schlussregel (2) eine Klausel gefolgert:

(2) $a, \neg a \lor b \vDash b$ oder $a, \neg a \vDash \text{falsch}$

Die gefolgerte Klausel ist entweder b oder die leere Klausel , d.h. die Klausel, die keine Literale enthält. Die leere Klausel wird gefolgert, wenn in der Menge M zwei Klauseln gefunden werden, die jeweils genau ein Literal enthalten und das Literal der einen Klausel die Negation des Literals der anderen ist.

Die nach der Schlussregel (2) gefolgerte Klausel wird Resolvente genannt. In der Menge M ersetzt die Resolvente die Klauseln, aus denen sie gefolgert wird.

Wird die leere Klausel gefolgert, ist die Anfrage bewiesen, ansonsten werden in der Menge M erneut, wie oben beschrieben, zwei Klauseln gesucht. Das wird solange ausgeführt, bis entweder die leere Klausel gefolgert wird oder in der Menge M keine zwei Klauseln gefunden werden, die resolvieren.

In den folgenden Ausführungen werden in einigen Klauseln zusätzliche Klammern eingefügt, um den Leserinnen und Lesern das Verständnis für das Folgern zu erleichtern.

Nachstehend der Beweis, den das PROLOG-System führt:
In der Menge M

$M = \{\neg q, m, n, p, q \vee \neg m \vee \neg n \vee \neg p\}$

wird beginnend mit der Abfrageklausel $\neg q$ eine zweite Klausel gesucht, die das Literal q enthält und die Klauseln $q \vee (\neg n \vee \neg p \vee \neg m)$ gefunden.
Aus den Klauseln $\neg q$ und $q \vee (\neg n \vee \neg p \vee \neg m)$ wird nach Schlussregel (2) die Klausel (Resolvente) $\neg m \vee \neg n \vee \neg p$ gefolgert. Es ergibt sich

$M = \{m, n, p, \neg m \vee \neg n \vee \neg p\}$.

Für die weitere Suche des PROLOG-Systems ist entscheidend, dass im Rumpf der Regel

```
q:- m, n, p.
```

erst m, dann n und zuletzt p stehen. Deshalb wird in der Menge M als Nächstes zur Klausel m eine Klausel gesucht, die das Literal $\neg m$ enthält und die Klausel
$\neg m \vee (\neg n \vee \neg p)$ gefunden. Aus den Klauseln m und $\neg m \vee (\neg n \vee \neg p)$ wird nach Schlussregel (2) die Klausel $\neg n \vee \neg p$ gefolgert. Es ergibt sich

$M = \{n, p, \neg n \vee \neg p\}$.

Danach wird in der Menge M zur Klausel n eine Klausel gesucht, die das Literal $\neg n$ enthält und die Klausel $\neg n \vee \neg p$ gefunden. Aus den Klauseln n und $\neg n \vee \neg p$ wird nach Schlussregel (2) die Klausel $\neg p$ gefolgert. Es ergibt sich

$M = \{p, \neg p\}$.

Zuletzt wird in der Menge M zur Klausel p eine Klausel gesucht, die das Literal $\neg p$ enthält und die Klausel $\neg p$ gefunden. Aus den Klauseln p und $\neg p$ wird nach Schlussregel (3) die leere Klausel gefolgert. Es ergibt sich

$M = \{ \ \}$.

Das PROLOG-System hat nachgewiesen, dass die Implikation $\neg q \Rightarrow$ falsch wahr ist. Nach Schlussregel (1) wird q gefolgert, d.h. q ist wahr. Die Anfrage `?- q.` ist bewiesen und das PROLOG-System gibt `Yes` aus.

Es folgen weitere Anfragen, die vom PROLOG-System zu beweisen sind.

An das Programm wird die Anfrage `?- n.` gestellt:
In der Menge

$M = \{\neg n, m, n, p, q \vee \neg m \vee \neg n \vee \neg p\}$

wird beginnend mit der Anfrageklausel $\neg n$ eine zweite Klausel gesucht, die das Literal n enthält und die Klausel n gefunden. Aus den Klauseln $\neg n$ und n wird die leere Klausel   gefolgert. Es ergibt sich

$M = \{\ \ , m, p, q \vee \neg m \vee \neg n \vee \neg p\}$

Das PROLOG-System hat gezeigt, das die Implikation $\neg n \Rightarrow$ falsch wahr ist. Nach der Schlussregel (1) ist n wahr und die Anfrage `?- n.` bewiesen. Es wird `Yes` ausgegeben.

An das Programm wird die Anfrage `?- k.` gestellt:
In der Menge

$M = \{\neg k, m, n, p, q \vee \neg m \vee \neg n \vee \neg p\}$

wird beginnend mit der Anfrage $\neg k$ eine zweite Klausel gesucht, die das Literal k enthält. Eine solche Klausel wird in der Menge M nicht gefunden. Das PROLOG-System kann die leere Klausel nicht folgern. Die Implikation $\neg n \Rightarrow$ falsch ist falsch. Es wird `No` ausgeben.

Exkurs:

a und b sind logische Aussagen. Für die Negation $\neg a$, Disjunktion $a \vee b$ und Implikation $a \Rightarrow b$ gelten per Definition:

a	b	$\neg a$	$a \vee b$	$a \Rightarrow b$
falsch	falsch	wahr	falsch	wahr
falsch	wahr	- - - -	wahr	wahr
wahr	falsch	falsch	wahr	falsch
wahr	wahr	- - - -	wahr	wahr

p und q sind aussagenlogische Variablen.

Der Ausdruck

$\neg p \Rightarrow falsch \vDash p$ (»aus $\neg p \Rightarrow$ falsch folgt p«)

ist eine Schlussregel (indirekter Beweis), weil die Implikation $(\neg p \Rightarrow falsch) \Rightarrow p$ bei jeder Belegung der aussagenlogischen Variable p wahr ist (Tautologie).

p	$\neg p$	$\neg p \Rightarrow$ falsch	$(\neg p \Rightarrow$ falsch$) \Rightarrow p$
falsch	wahr	falsch	wahr
wahr	falsch	wahr	wahr

Für das Schließen nach dieser Regel in einem Beweis ist wichtig:

Ist $\neg q \Rightarrow$ falsch wahr, folgt q ist wahr.

Der Ausdruck

$p, \neg p \vee q \vDash q$ (»aus p und $\neg p \vee q$ folgt q«)

ist eine Schlussregel (modus ponens), weil die Implikation $p \wedge (\neg p \vee q) \Rightarrow q$ bei jeder Belegung der aussagenlogischen Variablen p und q wahr ist (Tautologie).

p	q	$\neg p \vee q$	$p \wedge (\neg p \vee q)$	$p \wedge (\neg p \vee q) \Rightarrow q$
falsch	falsch	wahr	falsch	wahr
falsch	wahr	wahr	falsch	wahr
wahr	falsch	falsch	falsch	wahr
wahr	wahr	wahr	wahr	wahr

Für das Schließen nach dieser Regel in einem Beweis ist wichtig:

Ist $p \wedge (\neg p \vee q)$ wahr, folgt q ist wahr.

Der Ausdruck

$p, \neg p \vDash$ falsch (»aus p und $\neg p$ folgt falsch«)

ist eine Schlussregel, weil die Implikation $p \wedge \neg p \Rightarrow$ falsch bei jeder Belegung der aussagenlogischen Variable p wahr ist (Tautologie).

p	$\neg p$	$p \wedge \neg p$	$p \wedge \neg p \Rightarrow$ falsch
falsch	wahr	falsch	wahr
wahr	falsch	falsch	wahr

1.11.2 Unifikation

Gegeben ist das folgende PROLOG-Programm mit den Prädikaten *p/1*, *q/1* und *r/1*.

```
p(a).
q(a).
r(X):- p(X),q(X).
```

Die Anfrage (Ziel) an das Programm `?- r(E).` ist vom PROLOG-System zu beweisen:

In der Menge M der Klauseln

$M = \{\neg r(E), p(a), q(a), r(X) \vee \neg p(X) \vee \neg q(X)\}$

besitzen einige Literale eine Variable. Die Variablen E und X sind zu Beginn des Beweises frei, d.h. an keinen Wert gebunden. In der Menge M wird, beginnend mit der Anfrageklausel $\neg r(E)$, die Klausel $r(X) \vee (\neg p(X) \vee \neg q(X))$ gefunden.

Die freie Variable E und die freie Variable X unifizieren. Aus den Klauseln $\neg r(E)$ und $r(X) \vee (\neg p(X) \vee \neg q(X))$ wird die Klausel $\neg p(X) \vee \neg q(X)$ als Resolvente gefolgert. Es ergibt sich die Menge

$M = \{p(a), q(a), \neg p(X) \vee \neg q(X)\}$

in der zur Klausel p(a) die Klausel $\neg p(X) \vee \neg q(X)$ gefunden wird. Die Variable X und das Atom a unifizieren, d.h., die Variable X wird an den Wert a gebunden und gleichzeitig die Variable E. Aus den Klauseln p(a) und $\neg p(a) \vee \neg q(a)$ wird als Resolvente die Klausel $\neg q(a)$ gefolgert. Es ergibt sich die Menge

$M = \{q(a), \neg q(a)\}$

in der zur Klausel q(a) die Klausel $\neg q(a)$ gefunden wird. Die beiden Atome a unifizieren, weil beide Atome gleich sind. Aus den Klauseln q(a) und $\neg q(a)$ werden als Resolvente die leere Klausel gefolgert. Es ergibt sich die Menge

$M = \{ \}$.

Das PROLOG-System hat die Anfrage bewiesen und gibt

```
E = a
```

aus.

Exkurs:

In PROLOG ist eine Variable entweder frei oder gebunden. Eine freie Variable ist an keinen, eine gebundene Variable ist an einen Wert gebunden. Eine gebundene Variable darf an keinen anderen Wert gebunden werden. Nur ein Backtracking kann die Bindung der Variable lösen.

Unifikationsregeln:

Eine freie Variable kann mit einer anderen freien Variable unifizieren. Sobald eine der Variablen an einen Wert gebunden ist, ist die andere Variable an den gleichen Wert gebunden.

Eine freie Variable kann mit einer gebundenen Variable unifizieren. Die freie wird an den gleichen Wert wie die gebundene Variable gebunden.

Eine gebundene Variable kann mit einer gebundenen Variablen unifizieren, falls beide Variablen an den gleichen Wert gebunden sind.

Eine freie Variable kann mit einer Liste unifizieren. Die Variable wird an die Liste gebunden.

Eine freie Variable kann mit einem Atom unifizieren. Die Variable wird an das Atom gebunden.

Eine freie Variable kann mit einer Zahl unifizieren. Die Variable wird an die Zahl gebunden.

Eine Liste kann mit einer Liste unifizieren, falls beide Listenköpfe und beide Restlisten unifizieren.

Ein Atom kann mit einem Atom unifizieren, falls die beiden Atome gleich sind.

Eine Zahl kann mit einer Zahl unifizieren, falls die beiden Zahlen gleich sind.

Eine freie Variable kann mit einer Struktur unifizieren. Die Variable wird an die Struktur gebunden.

Beispiel: `X = p(a).` (`p` wird Funktor genannt)

Eine Struktur kann mit einer Struktur unifizieren, falls beiden Funktoren sowie die Stelligkeiten gleich sind und die Argumente unifizieren.

Beispiel: `p(a) = p(X).` (die freie Variable `X` wird an das Atom `a` gebunden)

1.11.3 Backtracking

Gegeben ist das Programm, das aus den Prädikaten *p/1*, *q/1* und *r/1* besteht:

```
p(a).                  /* erste Klausel */
p(b).
p(c).
q(a).
q(c).

r(X):- p(X),q(X).      /* letzte Klausel */
```

An das Programm wird die Anfrage:

```
?- r(E).
```

gestellt.

Um alle Antworten zu finden, muss das PROLOG-System mehrfach die Anfrage (Ziel)

```
?- r(E).
```

beweisen. Mit diesen Beweisen ist eine Tiefensuche (depth first search) verbunden, die durch einen Backtrackingalgorithmus realisiert wird. In das PROLOG-System ist ein derartiger Suchalgorithmus integriert. Der Ablauf der Tiefensuche wird durch die Reihenfolge der Klauseln innerhalb des Programms bestimmt.

Das Ziel `r(E)` enthält das Argument `E`, eine Variable, die zu Beginn der Suche frei, d.h. ungebunden, ist.

(1) Im Programm wird, beginnend mit der ersten Klausel, eine einstellige Klausel mit dem Namen `r` gesucht und die Regel `r(X):- p(X),q(X).` gefunden.
Die freien Variablen `E` und `X` unifizieren. Für die weitere Suche ist `p(X)` das erste und `q(X)` das zweite Teilziel. Die Suche wird mit dem ersten Teilziel `p(X)` fortgesetzt, d.h., im Programm wird eine einstellige Klausel mit dem Namen `p` gesucht. Dabei wird das Faktum `p(a)` gefunden, das die erste einstellige Klausel mit dem Namen `p` ist. Die Variable `X` und das Atom `a` unifizieren, wodurch gleichzeitig die Variable `E` an das Atom `a` gebunden wird. Die Suche wird mit dem zweiten Teilziel `q(a)` fortgesetzt. Dafür wird eine einstellige Klausel mit dem Namen `q` und dem Argument `a` gesucht. Gefunden wird das Faktum `q(a)`, das als einziges Faktum den Namen `q` und das Argument `a` besitzt. Das Atom `a` des zweiten Teilziels und das Atom `a` des Faktums unifizieren. Das PROLOG-System hat das Ziel, die Anfrage zu beweisen, erreicht und gibt `E = a` aus.

(2) Die Suche nach weiteren Antworten kann durch Interaktion des Nutzers erzwungen werden (gewillkürtes Backtracking). Dafür schließt das PROLOG-System das Faktum `q(a)` von der Suche aus, kehrt zum zweiten Teilziel `q(a)` (Wahlpunkt) zurück und setzt die Suche an dieser Stelle fort. Dabei wird keine Klausel mit dem Namen `q` und dem Argument `a` gefunden. Das Ziel, die Anfrage zu beweisen, wird nicht erreicht.

(3) Das PROLOG-System erzwingt die Fortsetzung der Suche. Die Bindungen der Variablen `X` und `E` an das Atom `a` werden gelöst, wobei `X` und `E` aneinander gebunden bleiben. Das PROLOG-System schließt für die Suche das Faktum `p(a)` aus und kehrt zum ersten Teilziel `p(X)` (Wahlpunkt) zurück. Im Programm wird eine einstellige Klausel mit dem Namen `p` gesucht und das Faktum `p(b)` gefunden, weil es die zweite einstellige Klausel mit dem Namen `p` ist. Die Variable `X` und das Atom `b` unifizieren. Gleichzeitig wird die Variable `E` an das Atom `b` gebunden. Die Suche wird mit dem zweiten Teilziel `q(b)` fortgesetzt und keine einstellige Klausel mit dem Namen `q` und dem Argument `b` gefunden. Das Ziel, die Anfrage zu beweisen, wird nicht erreicht.

(4) Das PROLOG-System erzwingt erneut die Fortsetzung der Suche, löst dafür die Bindungen der Variablen `X` und `E` an das Atom `b`, lässt jedoch `X` und `E` aneinander gebunden, schließt das Faktum `p(b)` von der weiteren Suche aus und kehrt zum ersten Teilziel `p(X)` (Wahlpunkt) zurück. Danach wird im Programm eine einstellige Klausel mit dem Namen `p` gesucht und das Faktum `p(c)` gefunden, weil es die dritte einstellige Klausel mit dem Namen `p` ist. Die Variable `X` und das Atom `c` unifizieren und `E` wird gleichzeitig an `c` gebunden. Die Suche wird mit dem zweiten Teilziel `q(c)` fortgesetzt und eine einstellige Klausel mit dem Namen `q` und dem Argument `c` gesucht. Es wird das Faktum `q(c)` gefunden, das als einzige Klausel den Namen `q` und das Argument `c` besitzt. Die beiden Atome `c` unifizieren. Da es kein weiteres Teilziel gibt, hat das PROLOG-System das Ziel, die Anfrage zu beweisen, erreicht und gibt `E = c` aus.

(5) Die Suche nach weiteren Antworten kann wiederum durch Interaktion des Nutzers erzwungen werden. Das bedeutet, das PROLOG-System kehrt zum zweiten Teilziel `q(c)` (Wahlpunkt) zurück, setzt die Suche fort und findet keine einstellige Klausel mit dem Namen `q` und dem Argument `c`. Das Ziel, die Anfrage zu beweisen, wird nicht erreicht.

(6) Das PROLOG-System erzwingt die Fortsetzung der Suche, löst dafür die Bindungen der Variablen `X` und `E` an das Atom `c`, wobei `X` an `E` gebunden bleibt. Das PROLOG-System kehrt zum ersten Teilziel `p(X)` (Wahlpunkt) zurück und führt die Suche fort, wobei keine einstellige Klausel mit dem Namen `p` gefunden wird. Das Ziel, die Anfrage zu beweisen, wird nicht erreicht.

(7) Das PROLOG-System erzwingt nochmals die Fortsetzung der Suche, löst dafür die Bindung der Variable `E` an `X`. Im Programm wird die Regel `r(X):- p(X),q(X).` von der Suche ausgeschlossen. Das PROLOG-System kehrt zum Ziel `r(E)` (Wahlpunkt) zurück, führt die Suche fort und findet keine einstellige Klausel mit dem Namen `r`. Das Ziel, die Anfrage zu beweisen, wird nicht erreicht. `No` wird ausgegeben. Die Suche ist beendet.

Hinweis: Die grafische Darstellung der Tiefensuche befindet sich auf der folgenden Seite.

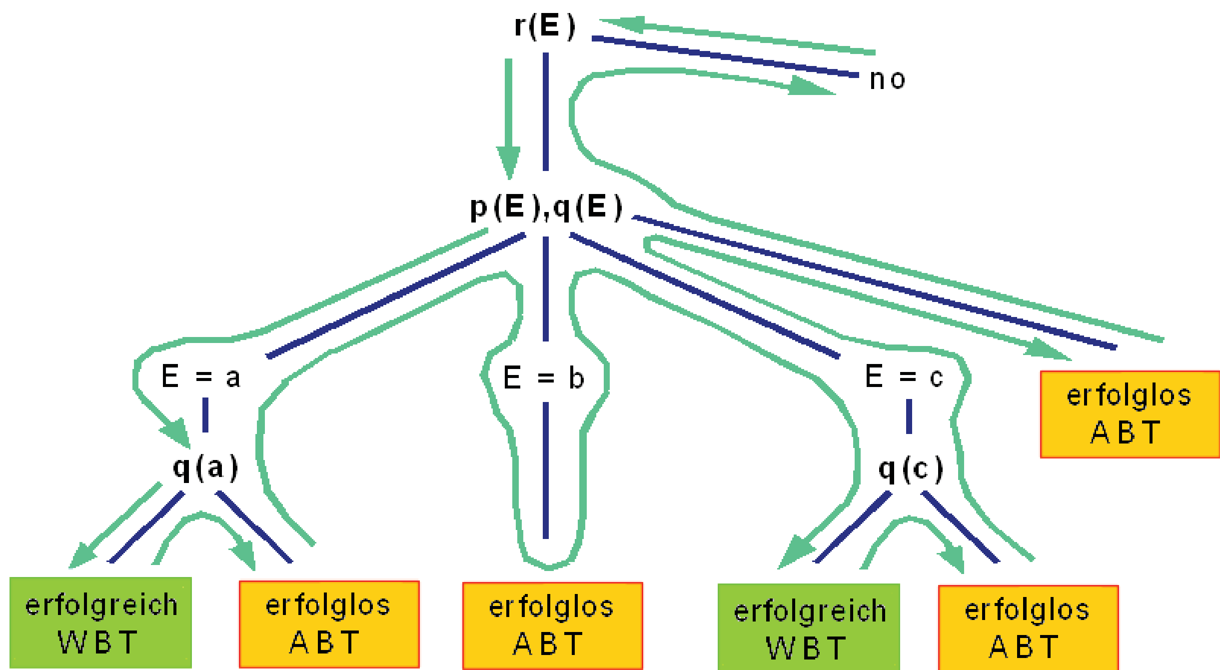


Abbildung 1.11.3 Grafische Darstellung der Tiefensuche des PROLOG-Systems nach Antworten auf die Anfrage `?- r( E ).`

2. Aufgaben aus dem Unterricht

2.1 Familienstammbäume

2.1.1 Karls Stammbaum

Von einem Familienstammbaum ist nur der Teil bekannt, der Auskunft über die Kinder, Enkel und Urenkel von Karl gibt.

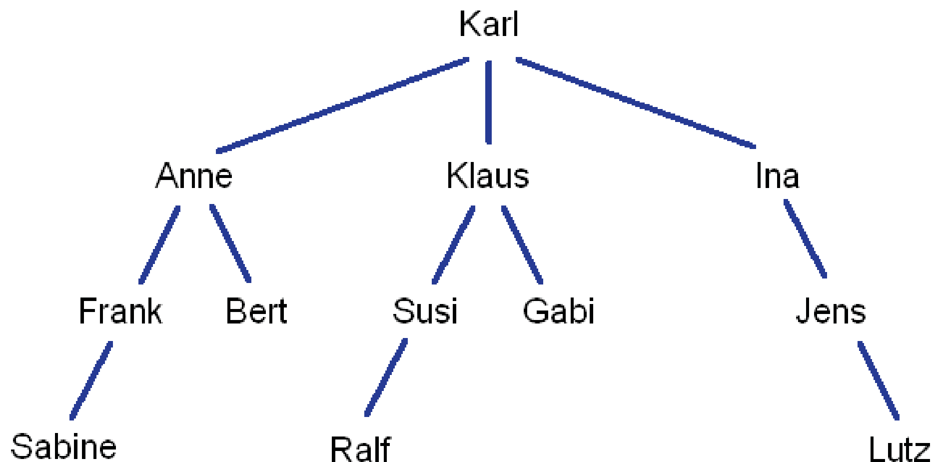


Abbildung 2.1.1 Auszug aus dem Familienstammbaum von Karl

a) Übertragen Sie alle Fakten aus dem Stammbaum unter Verwendung eines Vater-Kind-Prädikats und eines Mutter-Kind-Prädikats in ein PROLOG-Programm!

b) Erweitern Sie das Programm um ein Geschwister-Prädikat, ein Großmutter-Enkel-Prädikat, ein Großvater-Enkel-Prädikat und ein Enkel-Großelternteil-Prädikat!

c) Formulieren Sie die folgenden Anfragen in PROLOG-Notation!

   Wer ist Susis Vater?

   Wer ist Inas Mutter?

   Sind Frank und Bert Geschwister?

   Wer ist Sabines Großmutter?

   Wer ist wessen Enkel?

d) Erläutern Sie, wie das PROLOG-System auf die Anfrage

   Wer ist wessen Großvater?

   alle Antworten sucht!

2.1.2 Auszug aus dem Stammbaum der Musikerfamilie Bach

Christoph ist der Vater von Johann Ambrosius. Johann Ambrosius ist der Vater und Elisabeth die Mutter von Johann Christoph und Johann Sebastian. Maria Barbara ist die Mutter von Wilhelm Friedemann und Carl Philipp Emanuel. Anna Magdalena ist die Mutter von Johann Chris-

toph Friedrich und Johann Christian. Johann Sebastian ist der Vater von Wilhelm Friedemann, Carl Philipp Emanuel, Johann Christoph Friedrich und Johann Christian.

a) Stellen Sie den Auszug aus dem Stammbaum der Musikerfamilie Bach grafisch dar!

b) Übertragen Sie alle Fakten aus dem Stammbaum unter Verwendung der Prädikate *ist_maennlich/1*, *ist_vater_von/2, ist_ehefrau_von/2* in ein PROLOG-Programm!

c) Erweitern Sie das Programm um eine Regel für die Mutter-Kind-Beziehung!

d) Erweitern Sie das Programm um jeweils eine Regel für die Bruder- und die Halbbruder-Beziehung!

e) Formulieren Sie die folgenden Anfragen in PROLOG-Notation!
   - Ist Johann Christoph ein Bruder von Johann Sebastian?
   - Wie heißt der Bruder von Johann Christoph Friedrich?
   - Wie heißen die Halbbrüder von Wilhelm Friedemann?

2.2 Fakten, Regeln, Prädikate

Gegeben ist das folgende PROLOG-Programm:

```
p(a,b).
p(d,e).
p(b,f).

q(c,b).
q(d,b).
q(g,f).
q(a,e).
q(c,f).

r(X,Y,Z):- p(X,Z),q(Y,Z).
```

a) Geben Sie die Anzahl der Fakten und der Regeln des Programms an!

b) Nennen Sie alle Namen und die jeweilige Stelligkeit der Prädikate des Programms!

c) Erläutern Sie, wie das PROLOG-System auf die Anfrage `?- r(L,M,R).` die Antworten sucht!

2.3 Hinter sieben Bergen

Es lebten sieben Zwerge namens Erster, Zweiter, Dritter, Vierter, Fünfter, Sechster und Siebter, hinter den sieben Bergen. Als die Zwerge einmal nach dem Tagwerk in ihr Häuschen kamen, geschah Folgendes:

Erster rief: ”Wer hat auf meinem Stühlchen gesessen?”

Zweiter rief: ”Wer hat von meinem Tellerchen gegessen?”

Dritter rief: ”Wer hat von meinem Gemüschen gekostet?”

Vierter rief: ”Wer hat mit meinem Gäbelchen gestochen?”

Fünfter rief: ”Wer hat mit meinem Messerchen geschnitten?”

Sechster rief: ”Wer hat aus meinem Becherchen getrunken?”

Siebter rief: ”Wer liegt in meinem Bettchen?”

a) Übertragen Sie die Sätze der Geschichte als Fakten in ein PROLOG-Programm!

b) Formulieren Sie zum Programm die folgenden Anfragen:
   Ist vom Tellerchen von Zweiter gegessen worden?
   Ist mit dem Löffelchen von Fünfter gestochen worden?
   Was geschah mit dem Gemüschen von Dritter?
   Was stellte Siebter fest?
   Wessen Gemüschen wurde gekostet?
   Welches Besteckteil von wem wurde zum Schneiden benutzt?
   Wer rief was aus?

Weil Schneewittchen in allen Bettchen Probe lag und in dem von Siebter einschlief, riefen alle Zwerge außer Siebter entsetzt: ”Wer lag in meinem Bettchen?”

Erweitern Sie das Programm um eine Regel, die die Namen aller Zwerge außer Siebter bei Anfrage ausgibt: ”Wer lag in wessen Bettchen?”

Wie bekannt ist, überstand Schneewittchen alle Widrigkeiten der bösen Königin und feierte mit den Zwergen ein Fest. Sie stellten sich im Kreis auf (siehe Abbildung 2.3.1), fassten sich an den Händen und tanzten vor Freude.

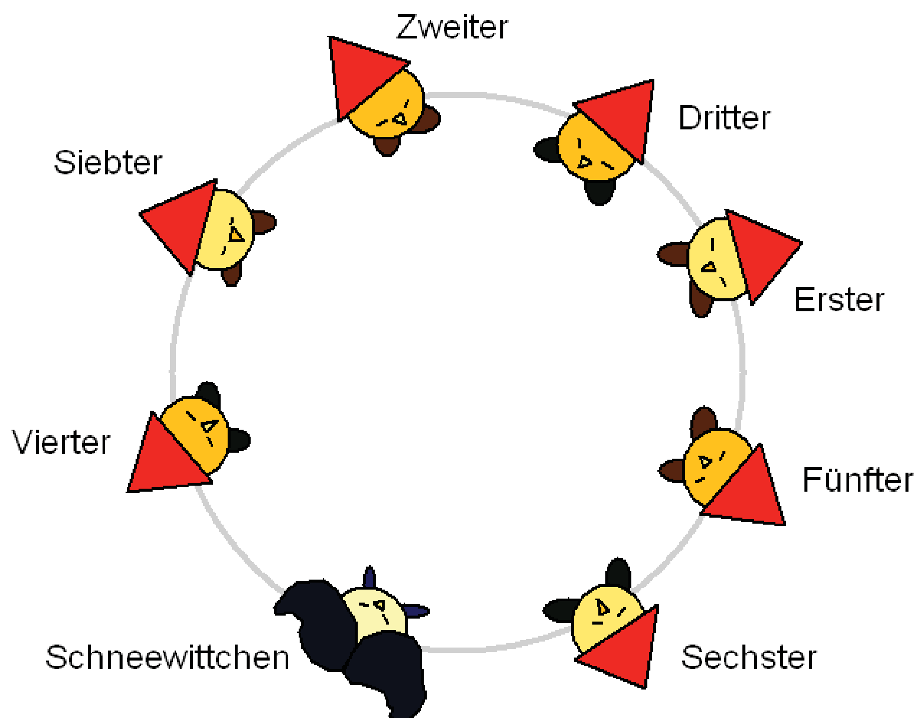


Abbildung 2.3.1 Anordnung der Tänzer im Kreis

c) Beschreiben Sie die Anordnung der Tänzer in einem Prädikat *tanzt_links_von/2* derart, dass zu jedem Tänzer sein rechter Nachbar angegeben wird!

d) Beschreiben Sie die Anordnung der Tänzer im Kreis derart in einem Prädikat *tanzt_rechts_von/2*, dass zu jedem Tänzer sein linker Nachbar angegeben wird. Verwenden Sie dafür das Prädikat *tanzt_links_von/2* aus Teilaufgabe d)!

e) Geben Sie ein Prädikat an, das bei Anfrage den linken und den rechten Nachbarn eines Tänzers ausgibt!

f) Geben Sie ein Prädikat an, das bei Anfrage ausgibt, welche Tänzer sich im Kreis gegenüberstehen!

P.S. Nach dem Fest zog Schneewittchen mit ihrem Prinz von dannen und die Zwerge lebten ungestört weiter.

2.4 Weimarer Dichterkreis

Dem literarischen Kreis von ANNA AMALIA, Herzogin zu Sachsen-Weimar-Eisenach, gehörten unter anderem die Dichter JOHANN WOLFGANG VON GOETHE, FRIEDRICH VON SCHILLER, JOHANN GOTTFRIED VON HERDER und CHRISTIAN MARTIN WIELAND an.

a) Übertragen Sie die folgenden Sätze als Fakten in ein PROLOG-Programm!
Der Dichter Goethe lebte von 1749 bis 1832.
Der Dichter Schiller lebte von 1759 bis 1805.
Der Dichter Herder lebte von 1744 bis 1803.
Der Dichter Wieland lebte von 1733 bis 1813.

b) Formulieren Sie zum Programm die folgenden Anfragen:
Wann wurde der Dichter Schiller geboren?
Wann verstarb der Dichter Wieland?
Welcher Dichter lebte von 1744 bis 1803?
Wann lebte der Dichter Goethe?

c) Der Dichter JOHANN CHRISTIAN FRIEDRICH HÖLDERLIN lebte von 1770 bis 1843.
Begründen Sie, weshalb das PROLOG-System auf die Anfrage, welcher Dichter von 1770 bis 1843 lebte, die Antwort `no` ausgibt!

d) Geben Sie eine PROLOG-Liste an, welche die Namen der Dichter des Weimarer Kreises enthält!

e) Geben Sie ein rekursives Prädikat an, das bei Anfrage die Anzahl der Elemente einer PROLOG-Liste ausgibt!

f) Formulieren Sie zu dem Programm die Anfragen:
   Wie viele Elemente enthält die leere Liste?
   Wie viele Elemente enthält die Liste aus Teilaufgabe e)?

g) Gegeben ist das folgende PROLOG-Programm:

```
ist_weiblich(von_lengefeld).
ist_weiblich(flachsland).
ist_weiblich(vulpius).
ist_weiblich(hillenbrand).

ist_maennlich(wieland).
ist_maennlich(goethe).
ist_maennlich(schiller).
ist_maennlich(herder)

sind_verheiratet(schiller,von_lengefeld).
sind_verheiratet(herder flachsland).
sind_verheiratet(wieland,hillenbrand).
sind_verheiratet(goethe vulpius).
```

   Erweitern Sie das Programm um die folgenden Prädikate *ist_ehefrau_von/2* und *ist_ehemann_von/2*!
   Verwenden Sie dafür die drei gegebenen Prädikate *sind_verheiratet /2*, *ist_weiblich/1* und *ist_maennlich/1*!
   Beachten Sie, dass die Prädikate *ist_ehefrau_von/2* und *ist_ehemann_von/2* keine Fakten enthalten dürfen!

h) Erläutern Sie, wie das PROLOG-System auf die folgende Anfrage alle Antworten sucht!

```
?- ist_ehefrau(F,M).
```

2.5 Thüringens Geographie

Gegeben sind die folgenden Sätze:

Der Adlersberg liegt im Thüringer Wald.

Der Poppenberg liegt im Unterharz.

Der Berg Ellenbogen liegt in der Vorderen Rhön.

Der Berg Wetzstein liegt im Thüringer Schiefergebirge.

Der Berg Großer Inselsberg liegt im Thüringer Wald.

Der Umpfenberg liegt in der Vorderen Rhön.

Der Berg Großer Beerberg liegt im Thüringer Wald.

a) Übertragen Sie die Sätze als Fakten in ein PROLOG-Programm!

b) Formulieren Sie zu diesem Programm die Anfragen:
   Liegt der Berg Wetzstein im Thüringer Schiefergebirge?
   Welche Berge liegen in der Vorderen Rhön?
   Wo liegt der Poppenberg?

c) Der Berg Kickelhahn liegt im Thüringer Wald.
   Begründen Sie, warum das PROLOG-System auf die Anfrage, welche Berge im Thüringer Wald liegen, Adlersberg, Großer Inselsberg und Großer Beerberg ausgibt, jedoch nicht Kickelhahn!

d) Erweitern Sie das Programm um ein rekursives Prädikat, das aus den Fakten der Teilaufgabe a) die Namen der Berge in eine PROLOG-Liste einfügt!
   Geben Sie zum Prädikat eine Anfrage an!

e) Gegeben ist das folgende PROLOG-Programm:

```
fluss(schleuse).
fluss(saale).
fluss(ilm).
fluss( schwarza).
fluss(orla).
fluss(ohra).

staut(talsperre_bleiloch,saale).
staut(talsperre_ohra,ohra).
staut(talsperre_schoenbrunn,schleuse).

ungestauter_fluss(X):- fluss(X),not staut(Y,X).
```

f) Erläutern Sie die Problemlösungsmethode "Backtracking" am Beispiel der Anfrage

```
?- ungestauter_fluss(W).
```

g) Das Prädikat *ungestauter_fluss/1* aus Teilaufgabe e) wird durch das Standardprädikat *cut/0* wie folgt verändert:

```
ungestauter_fluss(X):- fluss(X),not staut(Y,X),!.
```

   Begründen Sie, warum das PROLOG-System auf die Anfrage

```
?- ungestauter_fluss(W).
```

   nur die Antwort `W = ilm` ausgibt!

2.6 Expertensystem

Der Berg Großer Beerberg liegt im Thüringer Wald bei der Ortschaft Gehlberg und hat eine Höhe von 982 m. Der Umpfenberg liegt in der Vorderen Rhön bei der Ortschaft Kaltennordheim und ist 700 m hoch. Bei der Ortschaft Ilfeld im Unterharz liegt der Poppenberg, der eine Höhe von 600 m aufweist. Die Länge des Flusses Saale beträgt von Landesgrenze zu Landesgrenze innerhalb Thüringens 196 km. Die Werra fließt von der Quelle bis zur Landesgrenze 187 km durch Thüringen. Von der Quelle bis zur Mündung in die Saale legt die Ilm eine Strecke von 121 km zurück.

a) Übertragen Sie die Sätze zu ausgewählten Bergen und Flüssen des Freistaates Thüringen als Fakten in ein PROLOG-Programm!

b) Formulieren Sie zum Programm die Anfragen:
   In welchem Gebiet, bei welcher Ortschaft liegt der Umpfenberg und welche Höhe hat dieser Berg?
   Von wo bis wohin fließt die Ilm und wie lang ist die durchflossene Strecke?
   Wie heißt der Berg, der im Thüringer Wald nahe der Ortschaft Gehlberg liegt, und welche Höhe besitzt dieser Berg?
   Welcher Fluss fließt innerhalb Thüringens zwischen den Landesgrenzen?

c) Erläutern Sie, weshalb das PROLOG-System auf die Anfrage, ob der Fluss Unstrut von der Quelle bis zur Landesgrenze mit einer Länge von 159 km fließt, die Antwort `No` liefert!

d) Wipper, Gera, Weiße Elster, Weida, Schwarza, Orla und Felda sind weitere Namen von Flüssen, die in Thüringen fließen.
   Geben Sie eine PROLOG-Liste an, welche die Namen der Flüsse als Elemente enthält!

e) Ein Schüler führt am Computer mit einem Expertensystem einen Dialog in der Umgangssprache. Das Expertensystem kann Auskünfte zu geographischen Daten des Freistaates Thüringen geben.
   Der Schüler stellt die folgende Frage: Welcher Berg liegt in welchem Gebiet?
   Das Expertensystem versucht, eine Antwort zu finden. Hierfür wird die Frage in die PROLOG-Liste

```
[welcher, berg, liegt, in, welchem, gebiet]
```

   überführt.
   Das Prädikat *ignorieren/1* erfasst Wörter, die das Expertensystem nicht zum Beantworten von Fragen benötigt. Die folgenden drei Fakten bilden einen Teil dieses Prädikats:

```
ignorieren(welcher).
ignorieren(liegt).
ignorieren(welchem).
```

f) Ein Prädikat *filter/2* soll mithilfe des Prädikats *ignorieren/1* die Wörter aus der PROLOG-Liste

```
[welcher,berg,liegt,in,welchem,gebiet]
```

   herauslösen, die ignoriert werden sollen.
   Mithilfe des Prädikats *ignorieren/1* ist ein rekursives Prädikat *filter/2* anzugeben, das bei der Anfrage

```
?- filter([welcher,berg,liegt,in,welchem,gebiet],L).
```

   die Antwort `L = [berg,in,gebiet]` ausgibt!

2.7 Wanderungen

Von der Ortschaft Waldbach führen Wege über sieben Sehenswürdigkeiten zu drei Wanderzielen.

Übertragen Sie die nachfolgenden Sätze in ein PROLOG-Programm:

Die Teilstrecken der Wanderwege und ihre Weglängen sind:

- von Waldbach zur Waldquelle, 5 km,
- vom Karpfenteich zum Wolfsgrund, 7 km,
- vom Hexenhaus zum Feenstein, 4 km,
- von Waldbach zum Adlerhorst, 2 km,
- vom Fichtenkreuz zum Aussichtsturm, 8 km,
- von Adlerhorst zum Hexenhaus, 5 km,
- von Waldbach zum Fischbach, 1 km,
- vom Wolfsgrund zur Drachenschlucht, 9 km,
- von der Waldquelle zum Fichtenkreuz, 4 km
- vom Fischbach zum Karpfenteich, 6 km.

a) Stellen Sie die Wanderwege samt Teilstrecken in einer Skizze dar und ermitteln Sie die Wanderziele!

b) Geben Sie ein rekursives PROLOG-Prädikat an, das bei Anfrage einen Wanderweg von Waldbach zu einem Wanderziel sucht und die dazugehörige Länge des Weges ausgibt!

c) Formulieren Sie eine Anfrage, die als Antwort die Länge des Weges von Waldbach zu einem Wanderziel ausgibt!

d) Erweitern Sie das rekursive Prädikat aus Teilaufgabe b) so, dass bei Anfrage die besuchten Sehenswürdigkeiten auf dem Weg von Waldbach zu einem Wanderziel samt Weglänge ausgegeben werden!

2.8 Ampelmännchen

Unter den Lichtsignalanlagen der Stadt Erfurt gibt es verschieden gestaltete Fußgänger-Grün-Symbole, die landläufig als Ampelmännchen bezeichnet werden.

a) Übertragen Sie die folgenden Sätze als Fakten in ein PROLOG-Programm!

   Die Darstellung eines Bäckergesellen ist ein Ampelmännchen.
   Die Darstellung eines Eisessers ist ein Ampelmännchen.
   Die Darstellung eines Fußballers ist ein Ampelmännchen.
   Die Darstellung eines Wanderers ist ein Ampelmännchen.
   Die Darstellung eines Mannes mit Regenschirm ist ein Ampelmännchen.

Die Darstellung eines Mannes mit Kamera ist ein Ampelmännchen.

b) Übertragen Sie den folgenden Satz als Regel in das Programm!
   Jedes Ampelmännchen ist ein Fußgänger-Grün-Symbol.

c) Formulieren Sie für das Programm die Anfragen:
   Ist die Darstellung eines Bäckergesellen ein Ampelmännchen?
   Welche Darstellungen sind Fußgänger-Grün-Symbole?
   Ist die Darstellung eines Mannes mit Zylinder ein Ampelmännchen?

d) Die Darstellung eines Mannes mit Zylinder ist tatsächlich ein Ampelmännchen an einer Lichtsignalanlage in der Stadt Erfurt.
   Begründen Sie, weshalb das PROLOG-System auf die Anfrage, ob die Darstellung eines Mannes mit Zylinder ein Ampelmännchen ist, trotzdem `no` ausgibt!

2.9 Erfurts Sehenswürdigkeiten

Ein Besucher der Stadt Erfurt, der am Hauptbahnhof angekommen ist, möchte sich den Dom anschauen. Am Hauptbahnhof fragt er mehrere Leute nach einem Weg zum Dom. Das folgende PROLOG-Programm enthält alle Auskünfte, die sich der Besucher gemerkt hat.

```
gehe(hauptbahnhof,anger).
gehe(kraemerbruecke,alte_universitaet).
gehe(angerbrunnen,barfuesserkirche).
gehe(kraemerbruecke,fischmarkt).
gehe(anger,angerbrunnen).
gehe(barfuesserkirche,dom).
gehe(anger,kraemerbruecke).
gehe(fischmarkt,dom).
```

a) Bei Anfrage soll das PROLOG-System ausgeben, ob sich der Besucher einen Weg vom Hauptbahnhof zum Dom gemerkt hat. Dazu ist das Programm um ein rekursives Prädikat *weg/2* zu erweitern.
b) Formulieren Sie zum Programm die Anfrage, ob sich der Besucher einen Weg vom Hauptbahnhof zum Dom gemerkt hat!

Das folgende PROLOG-Programm enthält Sehenswürdigkeiten und Bauwerke der Stadt Erfurt. Unter den Sehenswürdigkeiten gibt es häufig besuchte Bauwerke.

```
sehenswuerdigkeit(gartenbauausstellung).
sehenswuerdigkeit(dom).
sehenswuerdigkeit(zoopark).
sehenswuerdigkeit(kraemerbruecke).

bauwerk(dom).
bauwerk(stadion).
bauwerk(kraemerbruecke).

haeufig_besucht(X):- sehenswuerdigkeit(X), bauwerk(X).
```

Erläutern Sie anhand der Anfrage

```
?- haeufig_besucht (B).
```

wie das Prolog-System mit Hilfe von Backtracking alle Antworten findet!

2.10 Gartenzwerge

Hinweis: Diese Aufgabe ist o h n e Computer zu lösen.

Bonifaz, Habakuk, Balthasar, Tobias, Utz, Zacharias, Theofan, Hieronymus, Thaddäus und Adolar sind Gartenzwerge. Jeder Gartenzwerg besitzt einen Gegenstand. Das PROLOG-Prädikat *benutzt/2* beschreibt die Zuordnung Zwerg – Gegenstand.

```
/* Programm Gartenzwerge */

benutzt(bonifaz,giesskanne).
benutzt(habakuk,brennholzrucksack).
benutzt(balthasar,sense).
benutzt(tobias,harke).
benutzt(utz,besen).
benutzt(zacharias,gartenschere).
benutzt(theofan,schubkarre).
benutzt(hieronymus,pfeife).
benutzt(thaddaeus,spaten).
benutzt(adolar,laterne).
```

a) Geben Sie zu dem Programm fünf verschiedene Anfragen und die dazugehörigen Antworten, die das PROLOG-System liefert, an!

   Das Programm wird um das Prädikat *gartenzwerge/1* erweitert.

```
gartenzwerge([bonifaz,habakuk,balthasar,tobias,utz,
              zacharias,theofan,hieronymus,thaddaeus, adolar]).
```

b) An das Programm werden die nachstehenden Anfragen gestellt:

```
?- gartenzwerg(A).
?- gartenzwerg([K|R]).
?- gartenzwerg([V,M|H]).
?- gartenzwerg([Z|_]).
```

   Geben Sie zu jeder Anfrage die vom PROLOG-System gelieferte Antwort an!

c) Erweitern Sie das Programm um ein Prädikat *gegenstand/1*, das aus einem Faktum besteht, welches als Argument eine PROLOG-Liste der Gegenstände der Gartenzwerge enthält!

d) Formulieren Sie eine Anfrage, mit deren Hilfe die ersten drei Gegenstände aus der Liste des Faktums der Teilaufgabe c) herausgelöst werden!

2.11 Rekursion

Gegeben sind die nachfolgenden Definitionen:

Produkt der natürlichen Zahlen n * m	pro(m,0)	=	0,
	pro(m,n)	=	pro(m,n -1) + m, n > 0
Potenz m^n ist eine natürlichen. Zahl	pot(m,0)	=	1,
	pot(m,n)	=	pot(m,n -1) * m, n > 0
Arithmetische Zahlenfolge	arith(0)	=	3,
	arith(n)	=	arith(n -1) + 14, n > 0
Fibonacci-Zahlen	fibo(0)	=	0,
	fibo(1)	=	1,
	fibo(n)	=	fibo(n -1) + fibo(n-2), n > 1
Ackermann-Funktion	ack(0,m)	=	m +1, m >= 0,
	ack(n,0)	=	ack(n -1,1), n > 0,
	ack(n,m)	=	ack(n -1, ack(n,m-1)), m > 0, m > 0

Binomialkoeffizienten von $(a + b)^n$, n, k sind natürliche Zahlen

binomial(n,0)	=	1,
binomial(n,n)	=	1, n > 0,
binomial(n,k)	=	binomial(n -1,k -1) +
		binomial(n -1,k), 0 < k < n

Übertragen Sie die Definitionen in Form rekursiver Prädikate in ein PROLOG-Programm!

2.12 Das ist das Haus vom Nikolaus

Gegeben ist der in Abbildung 2.12.1 dargestellte ungerichtete Graph mit den Knoten a, b, c, d, e und acht Kanten.

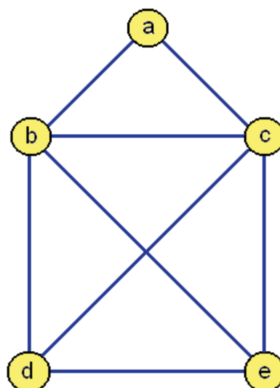


Abbildung 2.12.1 Graph zum Problem "Das ist das Haus vom Nikolaus"

Der Graph soll von einem Startknoten in einem Zug so durchlaufen werden, dass jede Kante genau einmal besucht wird.

a) Geben Sie jeden Knoten an, der Startknoten für einen Durchlauf sein kann!

b) Geben Sie für einen Durchlauf, beginnend mit einem Startknoten, alle besuchten Knoten in der richtigen Reihenfolge an!

c) Entwerfen und implementieren Sie ein PROLOG-Programm, das bei Anfrage alle voneinander verschiedenen Durchläufe ausgehend von einem Startknoten sucht und ausgibt!

   Beachten Sie: In der Anfrage ist der Startknoten anzugeben. Für jeden Durchlauf sollen, beginnend mit dem Startknoten, alle besuchten Knoten in der richtigen Reihenfolge ausgegeben werden.

d) Formulieren Sie zu dem Programm eine Anfrage, die alle voneinander verschiedenen Durchläufe von einem Startknoten aus sucht und ausgibt!

   Lösungshinweise:

   a) Ein Startknoten kann entweder der Knoten d oder der Knoten e sein.
      Von jedem Startknoten aus gibt es 44 verschiedene Durchläufe.

   b) Falls d als Startknoten gewählt wird, ist d – b – a – c – b – e – c – d – e ein möglicher Durchlauf.

2.13 Rundreise

In Abbildung 2.13.1 sind die Verbindungen der Städte a, b, c, d, e, f, g, h und i dargestellt.

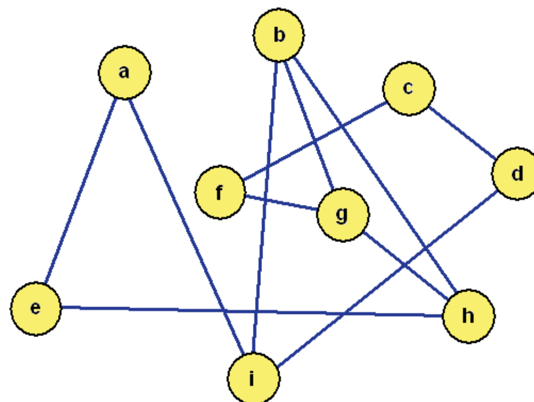


Abbildung 2.13.1 Darstellung der Städteverbindungen als ungerichteter Graph

Ein Reisender hat die Absicht von der Stadt a aus, die Städte b, c, d, e, f, g, h und i genau einmal zu besuchen. Seine Reise beginnt und endet in der Stadt a.

a) Entwerfen und implementieren Sie ein PROLOG-Programm, das bei Anfrage alle Reiserouten beginnend und endend in der Stadt a sucht und alle besuchten Städte ausgibt!

b) Geben Sie die Anfrage zur Ermittlung und Ausgabe der Reiserouten an!

2.14 Die Brücken von Paris

Durch Frankreichs Hauptstadt fließt die Seine. In Abbildung 2.14.1 sind vier Stadtteile von Paris (A, B, C, D) und deren 16 Brücken dargestellt.

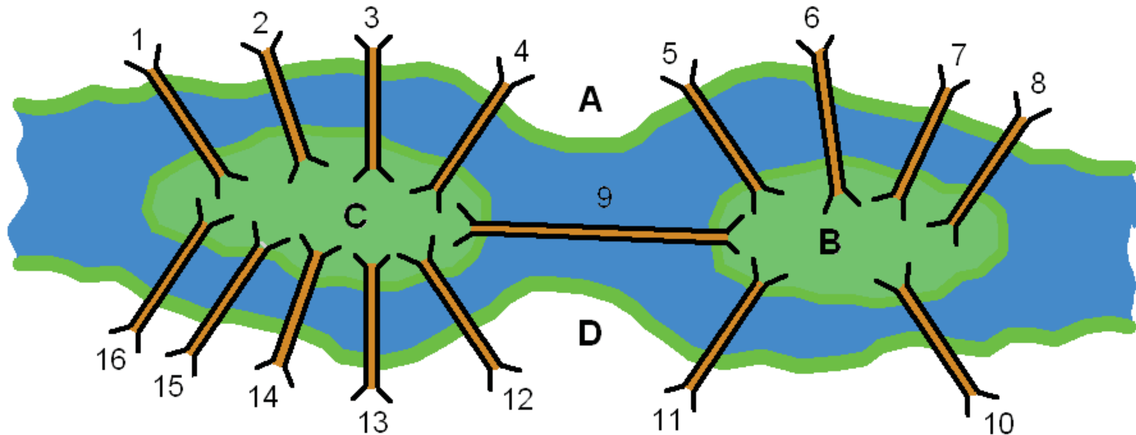


Abbildung 2.14.1 Pariser Brückenproblem (vgl. Weltbild's Mathematische Denkspiele S. 54)

Im Unterschied zum Königsberger Brückenproblem (L. EULER 1736) ist es in Paris möglich, Wege derart zu beschreiten, dass auf jedem Weg jede der 16 Brücken genau einmal überquert wird.

a) Geben Sie für einen Weg den Startort (Stadtteil), alle Brücken zwischen dem Start- und Zielort in der durchgangenen Reihenfolge und den Zielort (Stadtteil) an!

b) Entwerfen und implementieren Sie ein PROLOG-Programm, das bei Anfrage einen Weg von einem Startort- zu einem Zielort sucht!
Beachten Sie die Festlegung, dass vom Programm für den Weg der Startort (Stadtteil), alle Brücken zwischen dem Start- und Zielort in der durchgangenen Reihenfolge und der Zielort (Stadtteil) ausgegeben werden sollen!

Lösungshinweis: Nicht jeder der vier Stadtteile kann Startort eines Spaziergangs sein.

2.15 Kryptogramm

In Abbildung 2.15.1 ist ein Kryptogramm dargestellt.

	O	M	A
+	O	P	A
P	A	A	R

Abbildung 2.15.1 Kryptogramm

Jeder Buchstabe im Kryptogramm kann durch genau eine Ziffer substituiert werden. Eine korrekte Ziffernanordnung für die Buchstaben A, M, O, P, R liegt vor, wenn die Zahlen, die sich aus den Ziffern ergeben und die sich hinter OMA, OPA und PAAR verbergen, die Gleichung OMA + OPA = PAAR erfüllen!

Lösungen:

```
    8 4 6        7 3 4
+   8 1 6    +   7 1 4
---------    ---------
  1 6 6 2      1 4 4 8
```

Entwerfen und implementieren Sie ein PROLOG-Programm, das bei Anfrage alle korrekten Ziffernkombinationen ausgibt!

Lösungsvorschlag:

```
/* Fakten */

ziffer(0).
ziffer(1).
ziffer(2).
ziffer(3).
ziffer(4).
ziffer(5).
ziffer(6).
ziffer(7).
ziffer(8).
ziffer(9).

/* Regel */

loesung(P,A,R,O,M):- ziffer(A),
                     R is (A+A) mod 10,
                     R =\= A,
                     U1 is (A+A)/10,
                     ziffer(M),ziffer(P),
                     P =\= 0,M =\= P,M =\= A,M=\= R,R =\= P,
                     A is (M+P+U1) mod 10,
                     A =\= P,
                     U2 is (M+P+U1)/10,
                     ziffer(O),
                     O =\= M,O =\= P,O =\= A,O =\= R,
                     A is (O+O+U2) mod 10,
                     P is (O+O)/10.

?- loesung(P,A,R,O,M).
```

Hinweis: In Fix-PROLOG ist die Operation `/` die ganzzahlige Division. In anderen PROLOG-Versionen muss ggf. `/` gegen `div` ausgetauscht werden muss. Ähnlich verhält es sich mit der Relation `=\=`, die ggf. gegen `><` ausgetauscht werden muss.

2.16 Bibliothek

Die Mitglieder eines PROLOG-Fanclubs wollen ihre Bibliothek mit über 200 Bänden mittels Computer verwalten. Dazu haben sie die Tabellen Buch, Leser und Ausleihe entworfen.

In jeder der folgenden Tabellen sind nur einige Einträge angegeben.

Die Tabelle Buch:

Buchnummer	Buchtitel	Name des Autors
23	Prolog und kein Ende	Naumann
54	44 Prolog-Aufgaben	Schulze
12	Prolog in 20 Stunden	Meyer

Die Tabelle Leser:

Lesernummer	Name des Lesers	Anschrift des Lesers
14	Schmidt	Astadt, Rosenweg 2
9	Menzel	Dstadt, Deichgraben 42
43	Becker	Cdorf, Im Dorf 12
67	Kleinert	Bstadt, Burggasse 34

Die Tabelle Ausleihe:

Buchnummer	Lesernummer	Rückgabe des Buchs
12	14	15.01.2001
23	67	12.03.2001

a) Übertragen Sie die Tabellen Buch, Leser und Ausleihe als Prädikate *buch/3*, *leser/3* und *ausleihe/3* in ein PROLOG-Programm!

b) Formulieren Sie die Anfragen in PROLOG-Notation:
Wie lautet der Name des Autors und der Titel des Buchs mit der Nummer 54?
Wie lautet der Name, die Anschrift und Lesernummer eines beliebigen Lesers?
Wie lautet die Buchnummer des Buchs, das der Leser mit der Lesernummer 67 am 12.03.2001 zurückgeben muss?

c) Formulieren Sie eine Anfrage, ob Salzmann der Autor des Buchs »Prolog und kein Ende« ist!
Geben Sie die Antwort an, die das PROLOG-System auf diese Anfrage liefert und begründen Sie Ihre Antwort!

d) Unter Verwendung der Prädikate *buch/3*, *leser/3* und *ausleihe/3* ist das Programm um ein Prädikat zu erweitern, das bei Anfrage Auskunft gibt, welcher Leser welches Buch zurzeit ausgeliehen hat! Geben Sie zum Prädikat eine Anfrage an!

e) Erweitern Sie das Programm um ein Prädikat, das bei Anfrage ausgibt, auf welche Bücher zurzeit in der Bibliothek zugegriffen werden kann! Verwenden Sie ggf. dazu das Prädikat *buch/3* und das von Ihnen in Teilaufgabe d) implementierte Prädikat!
Geben Sie zum Prädikat eine Anfrage an!

f) Gegeben ist eine PROLOG-Liste, die als Elemente alle Lesernummern enthält.
Geben Sie ein rekursives Prädikat an, das bei Anfrage die Anzahl der Leser aus der Liste der Lesernummern ermittelt!
Geben Sie zu diesem Prädikat eine Anfrage an!
Verwenden Sie dafür die Liste [10,14,9,43,67,34,78]!

2.17 Sehen oder nicht sehen

Gegeben sind die PROLOG-Programme eins und zwei.

```
/* Programm eins */

spiegelbild(paul).
spiegelbild(X):- spiegelbild(X).

/* Programm zwei */

spiegelbild(X):- spiegelbild(X).
spiegelbild(paul).
```

Erläutern Sie, worin die Gemeinsamkeiten und Unterschiede der beiden Programme bestehen!

2.18 Binärer Baum

Hinweis: Diese Aufgabe ist für leistungsstarke Schülerinnen und Schüler gedacht.

a) Definieren Sie den Begriff "binärer Baum" rekursiv!

b) Das PROLOG-Prädikat *tree/3* enthält ein Faktum, das einen nichtleeren binären Baum beschreibt:

```
tree(1,tree(5,tree(7,tree(0,empty,empty),tree(4,empty,empty)),
     empty),tree(6,empty,tree(3,tree(8,empty,empty),empty))).
```

Hinweis: Der leere binäre Baum wird durch `empty` repräsentiert.
Skizzieren Sie den durch das Faktum beschriebenen binären Baum!

c) Erweitern Sie das Prädikat *tree/3* um ein Faktum, das den aus Abbildung 2.18.1 (siehe folgende Seite) dargestellten binären Baum beschreibt!

d) Entwerfen und implementieren Sie ein Prädikat, das bei Anfrage einen nichtleeren binären Baum postorder traversiert ausgibt!
Geben Sie eine Anfrage an, die bewirkt, dass das PROLOG-System den binären Baum aus Teilaufgabe c) postorder traversiert ausgibt!

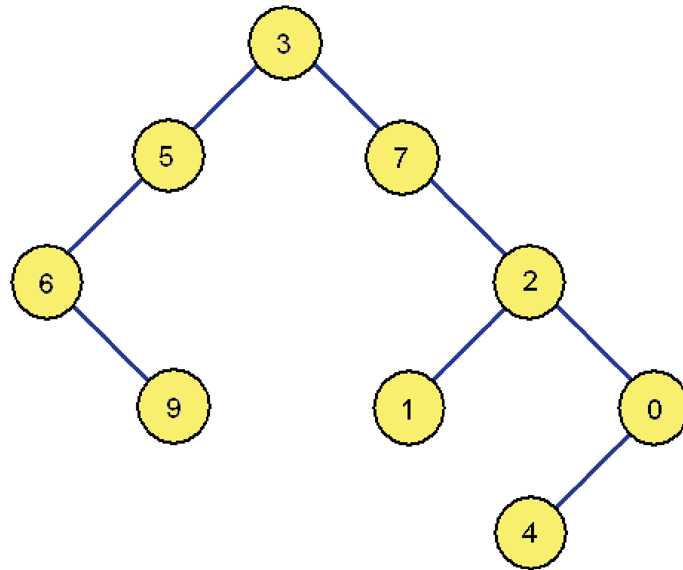


Abbildung 2.18.1 Darstellung eines binären Baums

e) Entwerfen und implementieren Sie ein Prädikat, das bei Anfrage prüft, ob eine beliebige positive ganze Zahl in einem nichtleeren binären Baum als Element enthalten ist!
Geben Sie zum Prädikat eine Anfrage an!

3. Aufgaben aus Abiturprüfungen

Grundfach 1994

1.a) Übertragen Sie die folgenden Sätze in ein PROLOG-Programm!

Dieter ist eine Person.
Hermann ist eine Person.
Renate ist eine Person.

Hermann ist männlich.
Renate ist weiblich.
Dieter ist männlich.

Eine Schülerin ist eine Person, die weiblich ist.
Ein Schüler ist eine Person, die männlich ist.

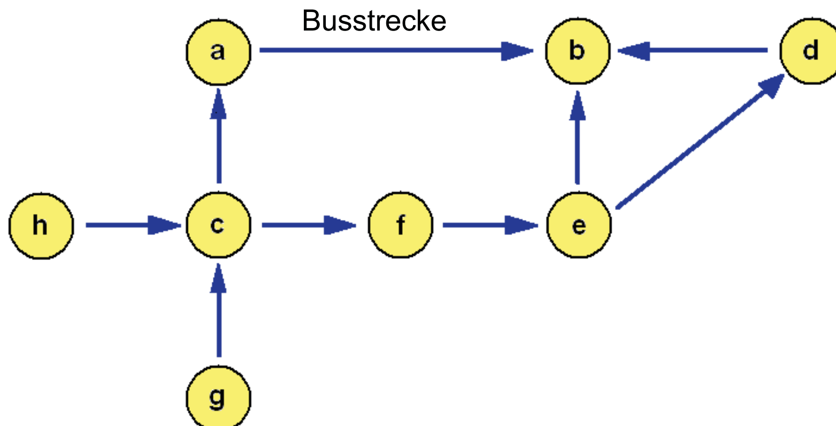
Jede Schülerin und jeder Schüler treibt Sport.

1.b) Formulieren Sie dazu die drei Anfragen in PROLOG!

Welche Person ist weiblich?
Wer ist Schüler?
Wer treibt Sport?

1.c) Geben Sie alle Antworten an, die das PROLOG-System auf Ihre drei Anfragen liefert!

2.a) Die Orte a, b, c, d, e, f, g, h sind durch neun Busstrecken folgendermaßen miteinander verbunden:



Beispiel: Ein Reisender von h nach f muss also zwei Busstrecken benutzen, da er in c umsteigen muss.
Beachten Sie, dass in der Abbildung keine Zyklen auftreten!

2.b) Schreiben Sie ein PROLOG-Programm, das als Fakten die in der Abbildung dargestellten neun Busstrecken mit Start- und Zielort erfasst!

Durch Verwendung von Rekursion ist eine Klausel anzugeben, die bei Anfrage darüber Auskunft gibt, ob ein gegebener Ort von einem anderen gegebenen Ort erreichbar ist.

3. Gegeben ist das folgende PROLOG-Programm:

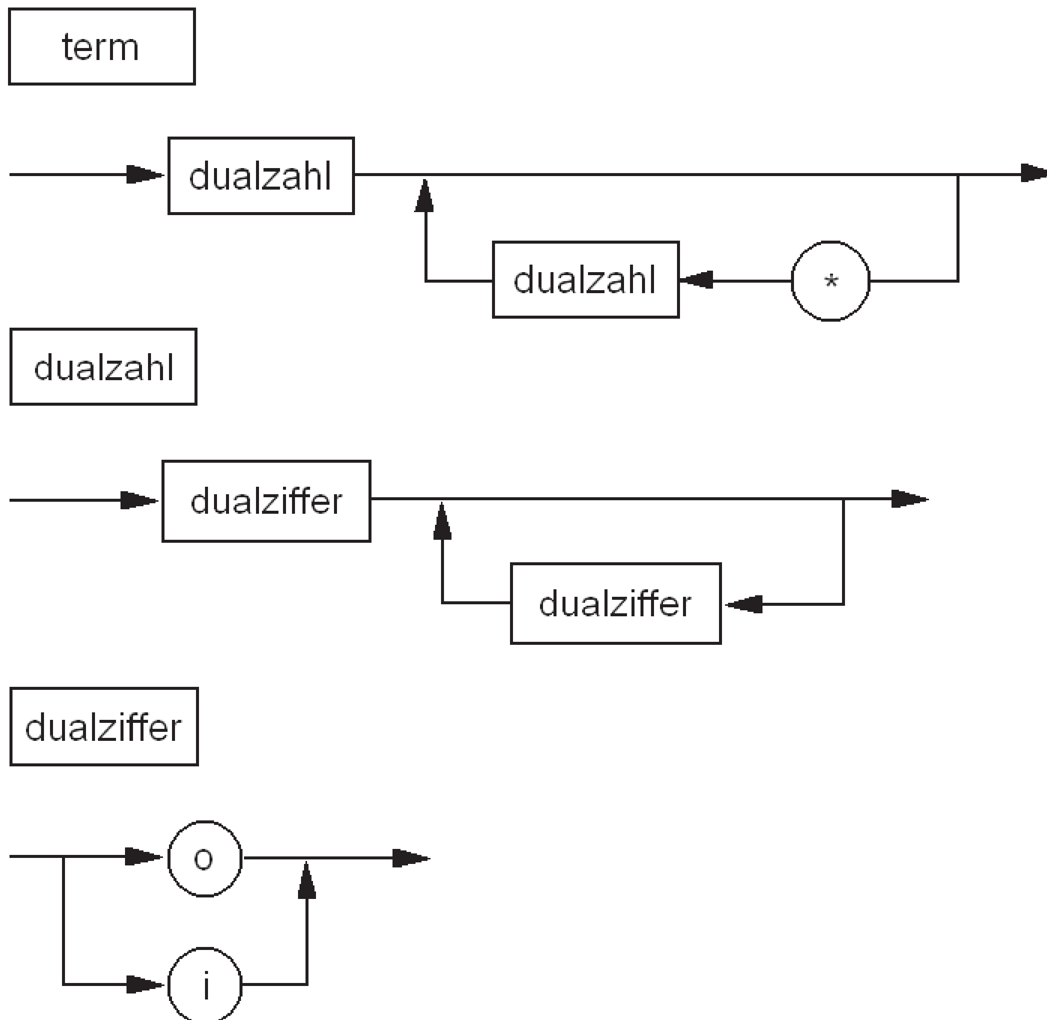
```
f(a).
f(b).
f(c).
```

```
f(d).
g(c).
h(X,d):-f(X),g(X).
```

Erläutern Sie an der Anfrage `?- h(Y,d).` die Problemlösungsmethode "Backtracking"!

Leistungsfach 1994

1. Gegeben sei die folgende vereinfachte Definition der Syntax von Term:



(Die Dualziffern sind durch die Kleinbuchstaben o und i dargestellt.)

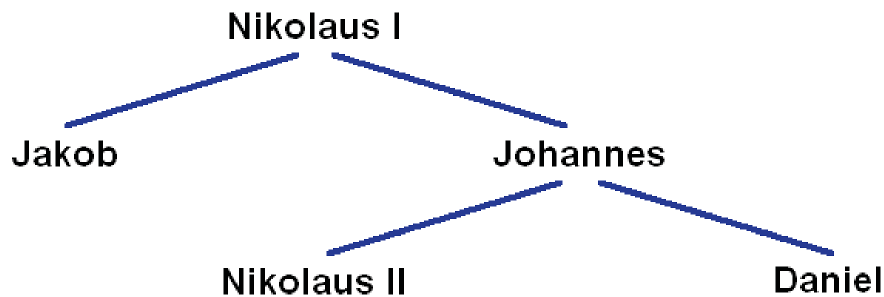
Entwerfen und implementieren Sie ein PROLOG-Programm, das feststellt, ob ein bei der Anfrage angegebener Term syntaktisch korrekt ist!

Bei Anfrage ist der Term als Liste anzugeben.

Beispiel: `?- korrekt([ i,o,i,i,*,i,*,o,i,o]).`

Grundfach 1995

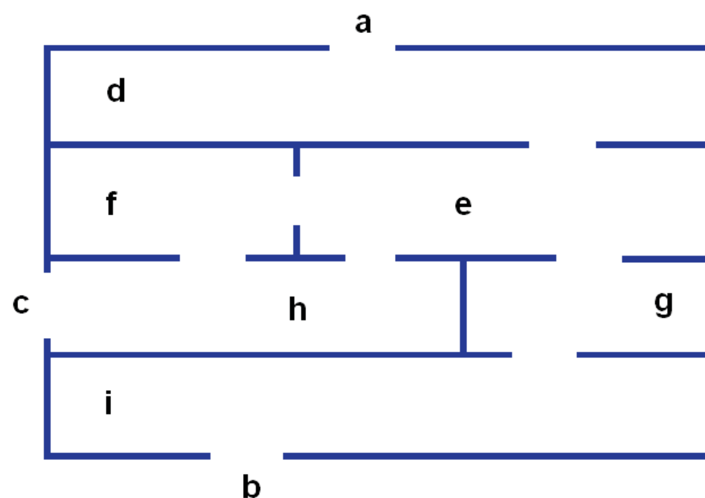
1. Aus dem Stammbaum der Schweizer Gelehrtenfamilie BERNOULLI ist ein Ausschnitt dargestellt:



1.a) Übertragen Sie alle Vater-Sohn-Beziehungen aus dem Ausschnitt des Stammbaums als Fakten in ein PROLOG-Programm!

1.b) Erweitern Sie Ihr PROLOG-Programm um eine Regel `bruder(X,Y)`!

2. Nach antikem Beispiel legten die Gartenarchitekten des Barocks Labyrinthe aus kunstvoll beschnittenen Hecken an.
Eine Draufsicht auf ein solches Labyrinth ist im folgenden Bild dargestellt:



Ein Labyrinth betritt man durch einen der Eingangsbereiche und sucht einen Weg zu einem der Ausgangsbereiche.

Das Labyrinth besteht aus den folgenden Bereichen: a, b, c. d, e, f, g, h und i.
Die Eingangs- bzw. Ausgangsbereiche sind a, b und c.

2.a) Erläutern Sie, wie mit Hilfe von Backtracking ein Weg innerhalb des Labyrinths von einem Eingangsbereich zu einem Ausgangsbereich gefunden werden kann!

2.b) Entwickeln Sie ein PROLOG-Programm, das alle Verbindungen benachbarter Bereiche des Labyrinths erfasst!

2.c) Durch Verwendung von Rekursion, Listen und dem PROLOG-Prädikat *member*

```
member(H,[H|_]).
member(H,[_|T]) :- member(H,T).
```

ist ein PROLOG-Prädikat *weg* anzugeben, das bei Anfrage einen Weg innerhalb des Labyrinths von einem Eingangsbereich zu einem Ausgangsbereich sucht!

Leistungsfach 1995

Listen

1. Erarbeiten und implementieren Sie ein PROLOG-Programm, das die folgenden Listenoperationen realisiert:

   a) Anfügen eines Elements am Kopf einer Liste,
   b) Ermitteln des ersten Elements einer Liste,
   c) Löschen des ersten Elements einer Liste,
   d) Zählen der Elemente einer Liste.

2. Formulieren Sie zu diesen Listenoperationen jeweils eine Anfrage in PROLOG!

Grundfach 1996

1.a) Übertragen Sie die folgenden Fakten in ein PROLOG-Programm! Es sind die zwei Prädikate *stern* und *planet* zu verwenden!

Venus, Erde, Mars, Jupiter, Saturn und Uranus sind Planeten.

Sonne ist ein Stern.

1.b) Erweitern Sie Ihr PROLOG-Programm um das Prädikat *mond*, das zu einem Mond den zugehörigen Planeten mit angibt!

Verwenden Sie dazu die folgenden Fakten:

Io ist Mond vom Jupiter.
Phobos ist Mond vom Mars.
Oberon ist Mond vom Uranus.
Ganymed ist Mond vom Jupiter.
Erdmond ist Mond von der Erde.
Titan ist Mond vom Saturn.

1.c) Sterne, Planeten und Monde sind Himmelskörper.

Erweitern Sie Ihr PROLOG-Programm um das Prädikat *himmelskoerper*!
Verwenden Sie dabei die Prädikate *stern*, *planet* und *mond*!

1.d) Formulieren Sie zu dem PROLOG-Programm die folgenden drei Anfragen:

Welche Monde hat der Jupiter?
Ist Ganymed ein Himmelskörper?
Von welchem Planeten ist Oberon der Mond?

1.e) Geben Sie die Antworten an, die das PROLOG-System auf Ihre drei Anfragen liefert! Begründen Sie für eine der formulierten Anfragen, warum das PROLOG-System die von Ihnen genannte Antwort liefert!

2.a) Entwickeln Sie ein vollständiges PROLOG-Programm, das bei Anfrage darüber Auskunft gibt, wie viele Elemente in einer PROLOG-Liste enthalten sind!

2.b) Formulieren Sie zu dem PROLOG-Programm die folgende Anfrage:
Wie viele Elemente enthält die Liste `[1,0,0,5,1,9,9,6]`?

3.a) Setzen Sie die folgenden Ableitungen und Differenziationsregeln in ein vollständiges PROLOG-Programm um!

Die Ableitung der Funktion $f(x) = x$ ist $f'(x) = 1$.

Die Ableitung der Funktion $f(x) = \sin(x)$ ist $f'(x) = \cos(x)$.

Die Differenziationsregel für die Funktion $f(x) = u(x) + v(x)$
ist $f'(x) = u'(x) + v'(x)$.

Die Differenziationsregel für die Funktion $f(x) = u(x) * v(x)$
ist $f'(x) = u'(x) * v(x) + u(x) * v'(x)$

3.b) Formulieren Sie zu dem PROLOG-Programm je eine Anfrage zum Differenzieren der folgenden Funktionen:

$f(x) = x + \sin(x)$
$f(x) = x * \sin(x)$

Leistungsfach 1996

Ein Beispiel für eine Zahl im ursprünglichen römischen Zahlensystem ist mdccccIxxxxvi. Erst zu Beginn des 16. Jahrhunderts bürgerten sich verkürzende Schreibweisen wie z.B. iv statt iiii oder xl statt xxxx ein.

Beschreibung der Syntax:

Es gibt die Ziffern m, d, c, l, x, v und i. Die PROLOG-Notation erzwingt die Verwendung von Kleinbuchstaben.

Die Reihenfolge der Ziffern ist vorgeschrieben: m d c l x v i.

Die Ziffern d, l und v können jeweils höchstens einmal vorkommen.

Die Ziffern c, x und i können jeweils höchstens viermal vorkommen.

Die Ziffer m kann beliebig oft vorkommen.

Beschreibung der Semantik:

Für die Umrechnung einer römischen Zahl in das Dezimalsystem gilt die folgende Tabelle:

römische Ziffer	Dezimalzahl
m	1000
d	500
c	100
l	50
x	10
v	5
I	1

Beispiel: mdcccclxxxxvi ist gleichwertig mit 1996.

Entwerfen und implementieren Sie ein PROLOG-Programm, das Folgendes leistet:

- Das Programm überprüft, ob eine eingegebene römische Zahl syntaktisch korrekt aufgebaut ist.
- Falls dies der Fall ist, rechnet das Programm die römische Zahl in die entsprechende Dezimalzahl um (Beispiel: mdcccclxxxxvi wird zu 1996).
- Beachten Sie die folgenden Festlegungen:
  Beim Lösen beziehen Sie sich auf das ursprüngliche römische Zahlensystem.

- Die römische Zahl ist als Liste einzugeben.
  Beispiel: `?- wert([m,d,c,c,c,c,l,x,x,x,x,v,i]).`
- Eine leere Liste darf nicht eingegeben werden.

Grundfach 1997

1. Aus dem Stammbaum von Karin sind die folgenden Fakten bekannt:

   Beate ist die Mutter von Karin.
   Bernd ist der Vater von Sabine.
   Ina ist die Mutter von Klaus.
   Ina ist die Mutter von Lisa.
   Ina ist die Mutter von Simone.
   Klaus ist der Vater von Beate.
   Lisa ist die Mutter von Thomas.
   Sebastian ist der Vater von Klaus.
   Sebastian ist der Vater von Lisa.
   Sebastian ist der Vater von Simone.
   Simone ist die Mutter von Bernd.
   Thomas ist der Vater von Bernd.

1.a) Stellen Sie diesen Stammbaum grafisch dar!

1.b) Übertragen Sie alle Mutter-Kind-Beziehungen und alle Vater-Kind-Beziehungen als Fakten in ein PROLOG-Programm!

1.c) Erweitern Sie das PROLOG-Programm um ein Prädikat für die Großvater-Enkel-Beziehung und um ein Prädikat für die Großmutter-Enkel-Beziehung!

1.d) Erweitern Sie das PROLOG-Programm um ein Prädikat Geschwister-Beziehung!

1.e) Beim Betrachten des Stammbaums stellt Karin fest:
»Die Geschwister meines Großvaters sind die Großmütter meiner Freunde.« Durch Verwendung der Prädikate für die Großmutter-Enkel-Beziehung, die Großvater-Enkel-Beziehung und die Geschwister-Beziehung ist das PROLOG-Programm um ein Prädikat zu erweitern, das bei Anfrage die Namen der Freunde von Karin liefert!

   Formulieren Sie dazu eine PROLOG-Anfrage!
   Welche Antworten liefert das PROLOG-System?

1.f) Erweitern Sie das PROLOG-Programm unter Verwendung von Rekursion um ein Prädikat für die Vorfahre-Beziehung!

Leistungsfach 1997

Symbolisches Differenzieren

1. Entwerfen und implementieren Sie ein PROLOG-Programm, das die 1. Ableitung einer gegeben Funktion ermittelt und ausgibt!

   Gegeben sind die folgenden Funktionen:

   $y = a$ (Konstante)
   $y = x$
   $y = x^n$ (n ist eine positive ganze Zahl)
   $y = \sin(x)$
   $y = \tan(x)$
   $y = e^x$
   $y = \ln(x)$

Diese Funktionen können durch die Operationen + und * verknüpft werden. Es können auch verkettete Funktionen gebildet werden.

Von Ihnen sind die folgenden Differenziationsregeln in dem PROLOG-Programm umzusetzen:

Die Faktorregel,
die Summenregel,
die Produktregel,
die Kettenregel für die oben genannten Funktionen.

Geben Sie zu Ihrem PROLOG-Programm für vier verschiedene Funktionen je eine Anfrage zum Ermitteln der 1. Ableitung an!

Beachten Sie die folgende Festlegung:
Ihr PROLOG-Programm erhält den Namen `diff.pro`!

2. Formulieren Sie zu dem Programm eine Anfrage zum Ermitteln der 1. Ableitung der Funktion $y = x^2 * \sin(e^x)$!

   Geben Sie die Antwort an, die das PROLOG-System auf Ihre Anfrage liefert!

   Begründen Sie, weshalb das PROLOG-System die von Ihnen genannte Antwort liefert!

Grundfach 1998

1. Die 22. Thüringenrundfahrt der Radelite führte 1997 von Eisenach über sechs Etappen nach Bad Salzungen. Die Städte Eisenach, Mühlhausen, Erfurt, Hildburghausen, Schmalkalden, Waltershausen und Bad Salzungen waren 1997 Start- bzw. Zielorte der sechs Etappen der Thüringenrundfahrt.

1.a) Übertragen Sie die folgenden Sätze als Fakten in ein PROLOG-Programm!

Die Stadt Eisenach liegt in Westthüringen.
Die Stadt Mühlhausen liegt in Nordwestthüringen.
Die Stadt Erfurt liegt in Mittelthüringen.
Die Stadt Hildburghausen liegt in Südthüringen.
Die Stadt Schmalkalden liegt in Südwestthüringen.
Die Stadt Waltershausen liegt in Westthüringen.
Die Stadt Bad Salzungen liegt in Südwestthüringen.

1.b) Der Große Inselsberg war das Etappenziel des Bergzeitfahrens der Thüringenrundfahrt.

Übertragen Sie den folgenden Satz als Faktum in das PROLOG-Programm!

Der Berg Großer Inselsberg liegt in Westthüringen.

1.c) Formulieren Sie zu Ihrem PROLOG-Programm die folgenden drei Anfragen:
Liegt die Stadt Mühlhausen in Nordwestthüringen?
Wo liegt der Berg Großer Inselsberg?
Welche Städte liegen in Südthüringen?

1.d) Die Stadt Suhl liegt in Südthüringen. Durch Suhl führte die vierte Etappe der Thüringenrundfahrt.

Begründen Sie, weshalb das PROLOG-System auf die Anfrage, welche Städte in Südthüringen liegen, Hildburghausen ausgibt, Suhl jedoch nicht!

2. Die Thüringenrundfahrt begann in Eisenach und führte über vier Etappen nach Schmalkalden. Mit Bussen wurden die Radsportler von Schmalkalden nach Waltershausen gefahren. Die fünfte Etappe war ein Bergzeitfahren von Waltershausen zum Großen

Inselsberg. Die sechste Etappe führte von Waltershausen nach Bad Salzungen. In Bad Salzungen endete die 22. Thüringenrundfahrt.
Das folgende PROLOG-Programm enthält unter anderem alle sechs Etappen der Thüringenrundfahrt.

```
etappe(eisenach,muehlhausen).
etappe(waltershausen,bad_salzungen).
etappe(erfurt,hildburghausen).
etappe(hildburghausen,schmalkalden).
etappe(waltershausen,grosser_inselsberg).
etappe(muehlhausen,erfurt).

fahrt(X,Y):- etappe(X,Y).
fahrt(X,Y):- etappe(X,Z),
             fahrt(Z,Y).
```

Das PROLOG-System gibt auf die Anfrage

```
?- fahrt(eisenach, schmalkalden).
```

die Antwort `yes` und auf die Anfrage

```
?- fahrt(eisenach, bad_salzungen).
```

die Antwort `No` aus.

2.a) Begründen Sie, warum das PROLOG-System die genannten Antworten ausgibt!

2.b) Erweitern und verändern Sie das Programm so, dass das PROLOG-System auf die Anfrage

```
?- fahrt(eisenach, bad_salzungen).
```

die Antwort `Yes` ausgibt!

Beachten Sie die folgende Festlegung:

Weder das Prädikat *etappe* noch das Prädikat *fahrt* dürfen durch Fakten erweitert werden!

3.a) Geben Sie eine PROLOG-Liste an, die die Städte Eisenach, Mühlhausen, Erfurt, Hildburghausen, Schmalkalden, Waltershausen und Bad Salzungen als Elemente enthält!

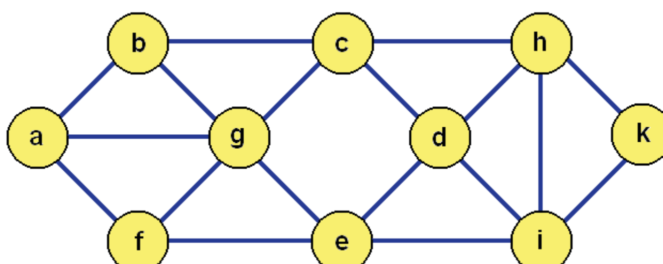
3.b) Durch Verwendung von Rekursion ist ein PROLOG-Prädikat anzugeben, das bei Anfrage ausgibt, ob eine beliebige Stadt in Ihrer PROLOG-Liste als Element enthalten ist!

3.c) Formulieren Sie zu dem PROLOG-Prädikat je eine Anfrage, für die das PROLOG-System `Yes` bzw. `No` ausgibt!

Leistungsfach 1998

Ungerichteter Graph

Gegeben ist der folgende ungerichtete Graph G:



Der Graph G besteht aus Knoten und Kanten.

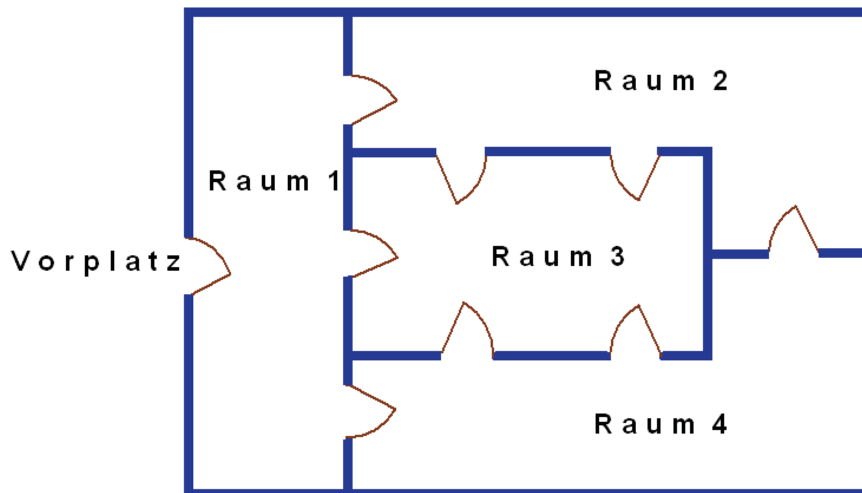
Die Knoten sind a, b, c, d, e, f, g, h, i und k.
Jede Kante verbindet zwei Knoten des Graphen G miteinander.
Entwerfen und implementieren Sie ein PROLOG-Programm, das im Graph G alle Wege von einem Start- zu einem Zielknoten sucht!
Formulieren Sie dazu eine Anfrage in PROLOG!

Beachten Sie die folgenden Festlegungen:

- Jeder Knoten des Graphen G kann Startknoten sein.
- Jeder Knoten des Graphen G kann Zielknoten sein.
- Jeder Knoten des Graphen G darf höchstens einmal durchlaufen werden.
- Für jeden gefundenen Weg ist der Startknoten, die Knoten, die auf dem Weg liegen, und der Zielknoten auszugeben.

Nachtermin Leistungsfach 1998

In der folgenden Abbildung ist der Grundriss eines Gebäudes mit dem Vorplatz, vier Räumen und neun Türen dargestellt:



Entwerfen und implementieren Sie ein PROLOG-Programm, das alle Wege vom Vorplatz zu einem beliebigen Raum sucht und ausgibt!

Beachten Sie die folgenden Festlegungen:

- Jede Tür ist in beiden Richtungen durchgehbar.
- Der Weg beginnt stets auf dem Vorplatz.
- Der Weg endet im Raum 1, im Raum 2, im Raum 3 oder im Raum 4.
- Jede der neun Türen ist genau einmal zu durchgehen.
- Jeder der vier Räume ist mindestens einmal zu betreten.
- Für jeden gefundenen Weg sind die durchgangenen Türen in der richtigen Reihenfolge anzugeben (Geben Sie dafür jeder Tür eine Nummer).
- Testen Sie Ihr PROLOG-Programm für jeden der vier Räume! Geben Sie dazu jeweils eine Anfrage an!

Grundfach 1999

2. Andreas, Beate, Christine, Diana, Erika, Frank, Gerd, Heike, Ines, Klaus, Lars, Manfred, Nina und Olaf wollten eine Party feiern. Über den Termin der Party wollten sie sich untereinander informieren.

2.a) Übertragen Sie die Fakten aus dem folgenden Text in ein PROLOG-Programm!

Gerd informiert Manfred über den Termin der Party.
Heike informiert Olaf über den Termin der Party.
Beate informiert Lars und Diana über den Termin der Party.
Lars informiert Erika und Frank über den Termin der Party.
Diana informiert Gerd und Heike über den Termin der Party.
Erika informiert Klaus und Christine über den Termin der Party.

2.b) Stellen Sie den Informationsfluss grafisch dar!

2.c) An der Party nahmen drei der oben genannten Personen jedoch nicht teil, weil sie über den Termin der Party nicht informiert worden waren. Alle anderen Personen haben an der Party teilgenommen.

Erweitern Sie Ihr PROLOG-Programm um ein Prädikat *teilnehmer*, das bei Anfrage und Wahl eines beliebigen Namens einer Person darüber Auskunft gibt, ob diese an der Party teilgenommen hat!

2.d) Formulieren Sie zu dem PROLOG-Programm folgende Anfragen:

Hat Ines an der Party teilgenommen?
Hat Olaf an der Party teilgenommen?

Geben Sie die Antworten an, die das PROLOG-System auf Ihre zwei Anfragen liefert!

Begründen Sie für beide Anfragen, warum das PROLOG-System die von Ihnen genannten Antworten liefert!

2.e) Gegeben ist das Faktum:

```
p([H|T],H,T).
```

Erläutern Sie, was die Anfrage

```
?- p([christine,diana,frank],X,Y).
```

liefert!

2.f) Erweitern Sie das PROLOG-Programm unter Verwendung von Rekursion um ein Prädikat *korrekt*! Dieses Prädikat soll bei Anfrage angeben, ob eine Liste von Personen korrekt ist.

Eine Liste von Personen ist korrekt, wenn sie nur Personen enthält, die an der Party teilgenommen haben.
Verwenden Sie dabei das Faktum p aus Teilaufgabe e) und das Prädikat *teilnehmer* aus Teilaufgabe c).

2.g) Formulieren Sie zu dem Prädikat *korrekt* aus Teilaufgabe f) eine Anfrage! Verwenden Sie dazu eine PROLOG-Liste mit von Ihnen ausgewählten Personen.

Leistungsfach 1999

3. Parser

Gegeben ist die folgende Definition der Syntax von Bezeichner im erweiterten Backus-Naur-Formalismus (EBNF):

bezeichner = buchstabe {buchstabe | ziffer}

buchstabe = "x" | "y".

ziffer = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".

3.1. Geben Sie für die Definition dieser Syntax von Bezeichner an

a) die Terminalsymbole,
b) die Nichtterminalsymbole,
c) das Startsymbol!

3.2. Überführen Sie die gegebene Definition in Syntaxdiagramme!

3.3. Entwerfen und implementieren Sie ein PROLOG-Programm für die gegebene Syntax von `bezeichner`!
Das Programm soll erkennen, ob ein bei der Anfrage angegebener `bezeichner` syntaktisch korrekt ist.

Beachten Sie die folgende Festlegung: `bezeichner` ist als Liste anzugeben.

Beispiel: `?- bezeichner([x,x,1,y,x,y,2,3,0]).`

Grundfach 2000

1. Ein Restaurant bietet den Gästen traditionelle Thüringer Gerichte an.

1.a) Übertragen Sie die Fakten aus dem folgenden Text in ein PROLOG-Programm!

Thüringer Rostbratwurst, Mutzbraten und Thüringer Rostbrätel sind Fleischbeilagen.
Bratkartoffeln und Kartoffelbrei sind Kartoffelbeilagen.
Gemüsebeilagen sind Rotkraut, Bohnengemüse und Sauerkraut.

1.b) Unter Verwendung der Fakten aus Teilaufgabe a) ist der folgende Satz als Regel in Ihr PROLOG-Programm zu übertragen!

Jedes Gericht besteht aus einer Fleischbeilage, einer Kartoffelbeilage und einer Gemüsebeilage.

1.c) Formulieren Sie zu Ihrem PROLOG-Programm die folgenden drei Anfragen!

Welche Fleischbeilagen werden angeboten?

Wird ein Gericht mit Thüringer Rostbratwurst, Bratkartoffeln und Sauerkraut angeboten?

Welche Gerichte werden angeboten?

1.d) Erweitern Sie Ihr PROLOG-Programm so, dass dieses auf die Anfrage, ob ein Gericht mit Rinderroulade, Thüringer Klößen und Rotkraut angeboten wird, die Antwort `Yes` ausgibt!

2. Gegeben ist das folgende PROLOG-Programm:

```
/* Wissensbasis */

rohkost(gurken).
dressing(franzoesisch).
rohkost(moehren).
```

```
dressing(italienisch)
salat(X,Y):- rohkost(X), dressing(Y).
```

2.a) Geben Sie von jedem Prädikat des PROLOG-Programms den Namen und die Stelligkeit an!

2.b) Erläutern Sie am Programm, wie das PROLOG-System auf die Anfrage

```
?- salat(R,D).
```

mittels Backtracking alle Antworten findet!

2.c) Die Regel *salat* des PROLOG-Programms wird um das Standardprädikat *cut* erweitert:

```
salat(X,Y) :- rohkost(X),!,dressing(Y).
```

Erläutern Sie am erweiterten Programm, warum das PROLOG-System auf die Anfrage

```
?- salat(R,D).
```

nur die Antworten

```
R = gurken, D = franzoesisch
```

und

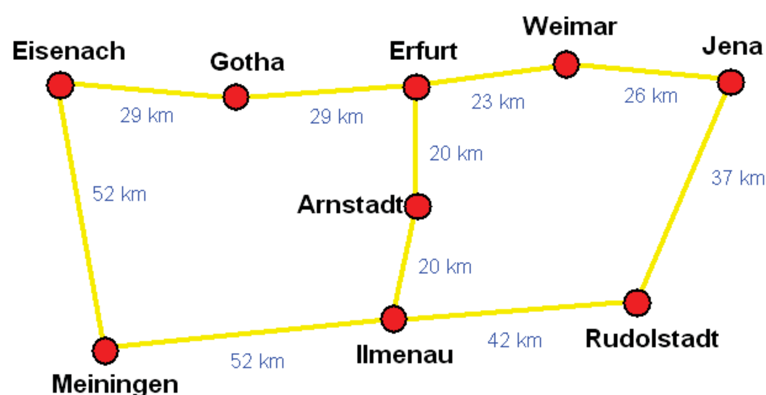
```
R = gurken, D = italienisch
```

findet!

Leistungsfach 2000

Auskunftssystem

Die Klassikerstraße Thüringens führt durch die Städte Weimar, Arnstadt, Ilmenau, Meiningen, Eisenach, Gotha, Erfurt, Jena und Rudolstadt. Die folgende Abbildung stellt den Verlauf der Klassikerstraße mit den Entfernungen zwischen benachbarten Städten dar.



Entwerfen und implementieren Sie ein PROLOG-Programm, das bei Anfrage alle Fahrtrouten von einer Startstadt zu einer Zielstadt sucht!
Beachten Sie die folgenden Festlegungen:

- Jede Stadt, durch die die Klassikerstraße führt, kann Startstadt sein.
- Zielstadt muss eine Stadt sein, durch die die Klassikerstraße führt, die nicht mit der Startstadt identisch ist.
- Eine Stadt darf höchstens einmal durchfahren werden.
- Vom Programm sind für jede Fahrtroute auszugeben:
  - die Städte, die auf der Fahrtroute liegen (in der Reihenfolge von der Startstadt bis zur Zielstadt) und
  - die Entfernung von der Start- bis zur Zielstadt.

Leistungsfach 2001

2. Die Postfixnotation eines Ausdrucks ist auch unter der Bezeichnung umgekehrte polnische Notation (UPN) bekannt.

2.1 Gegeben ist der folgende UPN-Ausdruck:

500 25 90 20 + * - 300 200 + -

Geben Sie zu diesem UPN-Ausdruck die Baumdarstellung an!
Erläutern Sie, wie der Wert des UPN-Ausdrucks berechnet wird!
Geben sie den Wert des Ausdrucks an!

2.2 Entwerfen und implementieren Sie ein PROLOG-Programm, das den Wert eines UPN-Ausdrucks berechnet und ausgibt!

Beachten Sie folgende Festlegungen:

- Die Operanten sind ganze Zahlen.
- Die Operationen sind Addition, Subtraktion, Multiplikation und ganzzahlige Division.
- Sie können voraussetzen, dass bei der Anfrage ein korrekter UPN-Ausdruck als Liste angegeben wird.
- Eine Anfrage ist zum Beispiel:

```
?- wert([500,25,90,20,'+','*',' - ',300,200,'+',' - '],W).
```

HINWEIS: Die Aufgabentexte des dritten Kapitels entsprechen nicht in jedem Fall den Texten der Originalaufgaben, die Grafiken wurden neu gezeichnet und dabei farbig dargestellt.

4. Lösungsvorschläge zu ausgewählten Abituraufgaben

Dieses Kapitel beinhaltet Hinweise und Vorschläge zum Lösen von PROLOG-Abituraufgaben, dabei wurden jedoch nicht alle Abiturjahrgänge und nicht alle Teilaufgaben eines Jahrgangs berücksichtigt.

Grundfach 1994

Teilaufgabe 2.a)

```
/* Praedikat busstrecke/2 */

busstrecke(h,c).
busstrecke(c,a).
busstrecke(c,f).
busstrecke(g,c).
busstrecke(a,b).
busstrecke(f,e).
busstrecke(e,b).
busstrecke(e,d).
busstrecke(d,b).
```

Teilaufgabe 2.b)

```
/* rekursives Praedikat verbindung/2 */
```

Entweder

```
verbindung(Z,Z).
verbindung(S,Z):- busstrecke(S,T),verbindung(T,Z).
```

oder

```
verbindung(S,Z):- busstrecke(S,Z).
verbindung(S,Z):- busstrecke(S,T),
                  verbindung(T,Z).

/* ggf. Bildschirmausgabe */

erreichbar(Start,Ziel):-  verbindung(Start,Ziel),
                          write('Vom Ort <'),
                          write(Start),
                          write('> aus ist der Ort <'),
                          write(Ziel),
                          write('> erreichbar.'),nl.
```

Teilaufgabe 3)

Die Problemlösungsmethode Backtracking kann durch einen Baum oder verbal beschrieben werden.

Grundfach 1995

Teilaufgabe 2.b)

Das Prädikat *verbindung/2* bringt die Verbindung zweier benachbarter Bereiche zum Ausdruck und besteht aus neun Fakten.

```
verbindung(a,d).
```

```
verbindung(d,e).
verbindung(e,f).
verbindung(f,h).
verbindung(h,c).
verbindung(e,h).
verbindung(e,g).
verbindung(g,i).
verbindung(i,b).
```

Teilaufgabe 2.c)

Jede Verbindung ist in beiden Richtungen traversierbar. Das Prädikat *durchgehen/2* bringt diese Symmetrie zum Ausdruck.

```
durchgehen(X,Y):- verbindung(X,Y);verbindung(Y,X).
```

Das Prädikat *member/2* ist vorgegeben.

```
member(H,[H|_]).
member(H,[_|T]):- member(H,T).
```

Damit die Wegsuche im Graphen nicht in Zyklen endet, wird mit einer Liste gearbeitet. Das Prädikat *weg/3* ist rekursiv und besteht aus einem Faktum und einer Regel.

```
weg(X,X,_).
weg(X,Y,L):- durchgehen(X,Z),
             not member(Z,L),
             weg(Z,Y,[Z|L]).
```

Beispiel für eine Anfrage: `?- weg(a,b,[a]).`

Das Prädikat *weg/3* könnte auch aus den folgenden beiden Regeln bestehen.

```
weg(X,Y,_):- durchgehen(X,Y).
weg(X,Y,L):- durchgehen(X,Z),
             not member(Z,L),
             weg(Z,Y,[Z|L]).
```

Grundfach 1996

Teilaufgabe 2.a)

```
/* rekursives Praedikat anzahl/2 */

anzahl([],0).
anzahl([_|R],N):-anzahl(R,M),N is M+1.
```

Teilaufgabe 2.b)

Eine Anfrage: `?- anzahl([1,0,0,5,1,9,9,6],A).`

Teilaufgabe 3.a)

```
/* Praedikate abl/2 und diff/2 */

abl(x,1).
abl(sin(x),cos(x)).
diff(U+V,DU+DV)    :-   abl(U,DU),abl(V,DV).
diff(U*V,DU*V+U*DV):-   abl(U,DU),abl(V,DV).
```

Zwei Anfragen sind beispielsweise:

```
?- diff(x+sin(x),D).
?- diff(x*sin(x),D).
```

Leistungsfach 1996

Das Prädikat *rz/2* mit den römischen Ziffern und deren dezimaler Entsprechung:

```
rz(m,1000).
rz(d,500).
rz(c,100).
rz(l,50).
rz(x,10).
rz(v,5).
rz(i,1).
```

Das Prädikat *b1/2* kontrolliert die Reihenfolge m, d, c, l, x, v, i.

```
b1([K],N)    :- rz(K,N).
b1([K|R],Z2):-  b1(R,Z1),
                rz(K,Z2),
                Z1 =< Z2.
```

Das Prädikat b2/3 ermittelt die Häufigkeit des Auftretens eines Buchstabens.

```
b2(_,[],0).
b2(B,[K|R],N):- b2(B,R,N1),
                (K==B,N is N1 + 1;true,N is N1),!.
```

Das Prädikat *konvertieren/2* ermittelt zu einer römischen Zahl die Entsprechung im Dezimalsystem.

```
konvertieren([],0).
konvertieren([K|R],Z):- konvertieren(R,Z1),
                        rz(K,E),
                        Z is Z1 + E.
```

Das Prädikat *wert/2* veranlasst das Überprüfen der Korrektheit einer römischen Zahl und das Konvertieren dieser Zahl in das Dezimalsystem.

```
wert(L,Z):- b1(L,_),
            (b2(d,L,N1),N1=<1),
            (b2(l,L,N2),N2=<1),
            (b2(v,L,N3),N3=<1),
            (b2(c,L,N4),N4=<4),
            (b2(x,L,N5),N5=<4),
            (b2(i,L,N6),N6=<4),
            konvertieren(L,Z).
```

Eine Anfrage lautet: `?- wert([m,d,c,c,c,c,l,x,x,x,x,v,i],W).`

Grundfach 1997

Teilaufgabe 1.c)

`X` ist Großvater von `Y`.

```
/* Praedikate grossmutter/2 und grossvater/2 */
```

```
grossvater(X,Y):- vater(X,Z),
                  (vater(Z,Y);mutter(Z,Y)).
grossmutter(X,Y):- mutter(X,Z),
                   (vater(Z,Y);mutter(Z,Y)).
```

Teilaufgabe 1.d)

`X` und `Y` sind Geschwister.

```
/* Praedikat geschwister/2 */

geschwister(X,Y):- vater(Z,X),
                   vater(Z,Y),X\==Y.
geschwister(X,Y):- mutter(Z,X),
                   mutter(Z,Y),X\==Y.
```

Teilaufgabe 1.e)

```
freunde(X):- grossvater(Y,karin),
             geschwister(Y,Z),
             grossmutter(Z,X).
```

Anfrage: `?- freunde(Wer).`

Teilaufgabe 1.f)

`X` ist Vorfahre von `Y`.

```
/* Praedikat vorfahre/2 */

vorfahre(X,X).
vorfahre(X,Y):- (vater(Z,Y);mutter(Z,Y)),
                vorfahre(X,Z).
```

oder

```
vorfahre(X,Y):- vater(X,Y);mutter(X,Y).
vorfahre(X,Y):- (vater(Z,Y);mutter(Z,Y)),
                vorfahre(X,Z).
```

Grundfach 1998

Teilaufgabe 1.d)

Begründung:

Das Faktum, die Stadt Hildburghausen liegt in Südthüringen, ist im PROLOG-Programm enthalten (siehe 1.a)). Deshalb findet das PROLOG-System auf die Anfrage, welche Städte in Südthüringen liegen, die Stadt Hildburghausen.

Ein Faktum, die Stadt Suhl liegt in Südthüringen, ist nicht im PROLOG-Programm enthalten (siehe 1.a)). Deshalb findet das PROLOG-System auf die Anfrage die Stadt Suhl nicht.

Teilaufgabe 2.a)

Begründung:

Von Eisenach führen vier Etappen hintereinander nach Schmalkalden. Das wird im PROLOG-Programm durch Fakten beschrieben. Auf die Anfrage, ob es eine Fahrt von Eisenach nach

Schmalkalden gibt, findet das PROLOG-System mithilfe des rekursiven Prädikats *fahrt/2* die Antwort `Yes`.

Im PROLOG-Programm ist keine Etappe von Schmalkalden nach Waltershausen als Faktum beschrieben. Das PROLOG-System findet auf die Anfrage, ob es eine Fahrt von Eisenach nach Bad Salzungen gibt, mithilfe des rekursiven Prädikats *fahrt/2*, eine Fahrt nach Schmalkalden, jedoch keine nach Waltershausen. Das PROLOG-System gibt die Antwort `No` aus.

Teilaufgabe 2.b)

Programmerweiterung:

```
/* Praedikat bustransfer/2 */

bustransfer(schmalkalden,waltershausen).

/* Praedikat fahrt/2 */

fahrt(X,X).
fahrt(X,Y):- (etappe(X,Z);bustransfer(X,Z)),
             fahrt(Z,Y).
```

oder

```
fahrt(X,Y):- etappe(X, Y).
fahrt(X,Y):- (etappe(X,Z);
             bustransfer(X,Z)),
             fahrt(Z,Y).
```

Nachtermin Leistungsfach 1998

```
/* Prolog-Programm Wegsuche */

tuer(vorplatz,raum1,1).
tuer(raum1,raum2,2).
tuer(raum1,raum3,3).
tuer(raum1,raum4,4).
tuer(raum2,raum3,5).
tuer(raum2,raum3,6).
tuer(raum2,raum4,7).
tuer(raum4,raum3,8).
tuer(raum4,raum3,9).

durchgehen(X,Y,Nr):- tuer(X,Y,Nr);tuer(Y,X,Nr).

/* Rekursives Praedikat weg/3 */

weg(X,X,L):- anzahl(L,A),A==9,
             ausgabe(L),nl.
weg(X,Y,L):- durchgehen(X,Z,Nr),
             not enthalten(Nr,L),
             weg(Z,Y,[Nr|L]),fail.

/* Hilfspraedikate */

enthalten(E,[E|_]).
enthalten(E,[_|L]):- enthalten(E,L).

anzahl([],0).
anzahl([_|L],Anz):- anzahl(L,H),
                    Anz is H+1.
```

```
ausgabe([]).
ausgabe([E|L]):- ausgabe(L),
                 write(E).
```

Es gibt nur vom Vorplatz zum Raum 3 Wege.

Die Anfrage lautet:

```
?-weg(vorplatz,raum3,[]).
```

Grundfach 1999

Teilaufgabe 2.f)

Es wird ein Name aus der Liste L herausgenommen und gefragt, ob eine Person mit diesem Namen an der Party teilgenommen hat. Das geschieht solange, bis entweder die Liste L leer ist, die Liste L ist dann korrekt oder ein Name von einer Person herausgelöst wird, die nicht an der Party teilgenommen hat, die Liste L ist dann nicht korrekt.

```
korrekt([]).
korrekt(L):- p(L,K,R),
             teilnehmer(K),
             korrekt(R).
```

Grundfach 2000

Teilaufgabe 2.c)

Das Standardprädikat *cut/0* verhindert nach Abarbeiten des Teilziels

```
rohkost(gurken),dressing(Y)
```

jedes weitere Backtracking im Prädikat *rohkost/1*.

Leistungsfach 2000

Die Lösung der Aufgabe besteht darin, eine Wegsuche in einem ungerichteten Graphen zu programmieren. Dafür können ggf. die Hilfsprädikate *hinein/3*, *heraus/3* und *umdrehen/3* verwendet werden. Das Prädikat *von_nach/3* enthält 11 Fakten:

```
von_nach(eisenach,gotha,29).
...
von_nach(meiningen,eisenach,52).
```

Das Prädikat *strasse/3* bringt zum Ausdruck, dass jede Straße zwischen zwei Städten in beiden Richtungen durchfahren werden kann.

```
strasse(X,Y,W):- von_nach(X,Y,W);
                 von_nach(Y,X,W).
```

Das Hilfsprädikat *ausgeben/1* dient dem elementweisen Ausgeben einer PROLOG-Liste.

```
ausgeben([]).
ausgeben(L):-  heraus(L,K,R),
               write(K),nl,
               ausgeben(R).
```

Das rekursive Prädikat *weg/4* beschreibt die Wegsuche im ungerichteten Graphen.

```
weg(X,X,N,L):-  umdrehen(L,[],LN),
                ausgeben(LN),
                write(N),write(' km'),nl.
weg(X,Y,N,L):-  strasse(X,Z,W),
                not enthalten(Z,L),
                N1 is N + W,
                hinein(Z,L,LN),
                weg(Z,Y,N1,LN),fail.
```

Das Standardprädikat *fail/0* erzwingt bei der Wegsuche alle Backtrackingmöglichkeiten.

Um einem Anwender das Formulieren von Anfragen zu vereinfachen, wird das Prädikat *fahrtroute/2* in das Programm aufgenommen.

```
fahrtroute(Start,Ziel):- weg(Start,Ziel,0,[Start]).
```

Bei der Anfrage

```
?- fahrtroute(weimar,meiningen).
```

sucht das PROLOG-System alle Fahrtrouten von Weimar nach Meiningen und gibt diese mit Entfernungsangaben aus.

Eine andere Implementierung könnte sein:

```
ausgeben([]).
ausgeben([K|R]):-  write(K),nl,
                   ausgeben(R).
weg(X,X,N,L):- ausgeben(L),
               write(N),write(' km'),nl.
weg(X,Y,N,L):- strasse(X,Z,W),
               not enthalten(Z,L),
               N1 is N + W,
               weg(Z,Y,[Z|L]),fail.
```

Das Prädikat *umdrehen/3*, welches die Reihenfolge der Elemente einer PROLOG-Liste umkehrt, ist im Prädikat *weg/3* nicht erforderlich, wenn das Prädikat *fahrtroute/2* wie folgt

```
fahrtroute(Start,Ziel):- weg(Ziel,Start,0,[Ziel]).
```

verändert wird.

Danksagungen

Frau Petra Bohn, Herr Dr. Michael Fothe, Frau Angela Merten, Herr Dr. Wolfgang Moldenhauer und Herr StD Hermann Stimm arbeiteten das Manuskript gründlich durch und halfen uns mit vielen sachdienlichen Vorschlägen und Korrekturhinweisen.

Dank an die nicht einzeln genannten Kolleginnen und Kollegen, die uns während der Erarbeitungsphase dieser ThILLM-Veröffentlichung zahlreiche Anregungen gaben.

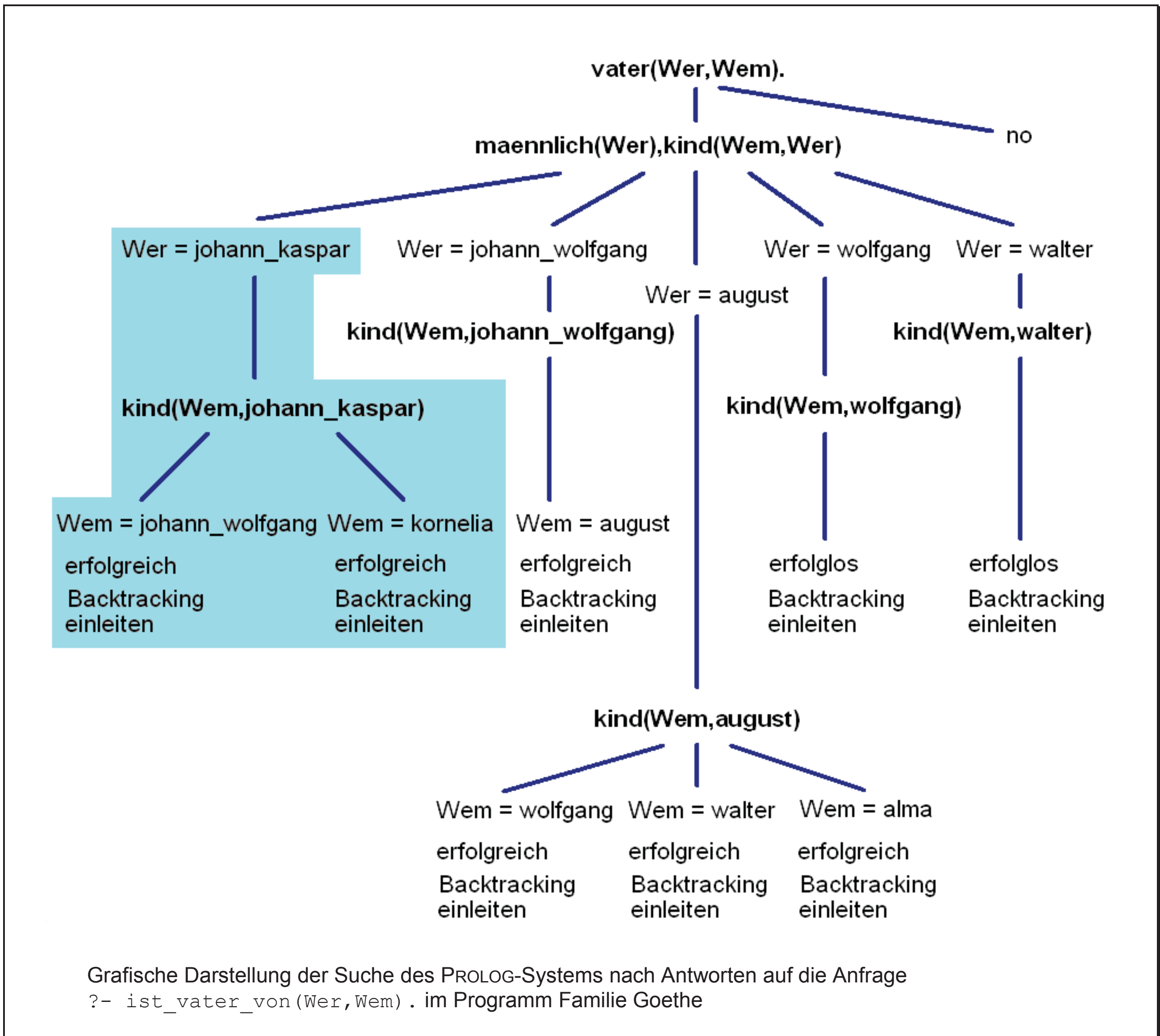
Das ThILLM ermöglichte uns Arbeitsberatungen an zentralen Orten. Hierdurch konnten unumgänglich notwendige Absprachen unkompliziert durchgeführt werden.

Anhang 1

Kopiervorlagen

Auszug aus dem Stammbaum der Familie Goethe:

Mitglieder der Familie Goethe sind Johann Kaspar, Johann Wolfgang, August, Wolfgang und Walter. Katharina Elisabeth, Kornelia, Christiane, Ottilie und Alma sind die weiblichen Familienmitglieder. Johann Kaspar und Katharina Elisabeth, Johann Wolfgang und Christiane sowie August und Ottilie sind miteinander verheiratet. Johann Wolfgang und Kornelia sind die Kinder von Johann Kaspar und Katharina Elisabeth. August ist das Kind von Johann Wolfgang und Christiane. Wolfgang, Walter und Alma sind die Kinder von August und Ottilie.



Grafische Darstellung der Suche des PROLOG-Systems nach Antworten auf die Anfrage `?- ist_vater_von(Wer,Wem).` im Programm Familie Goethe

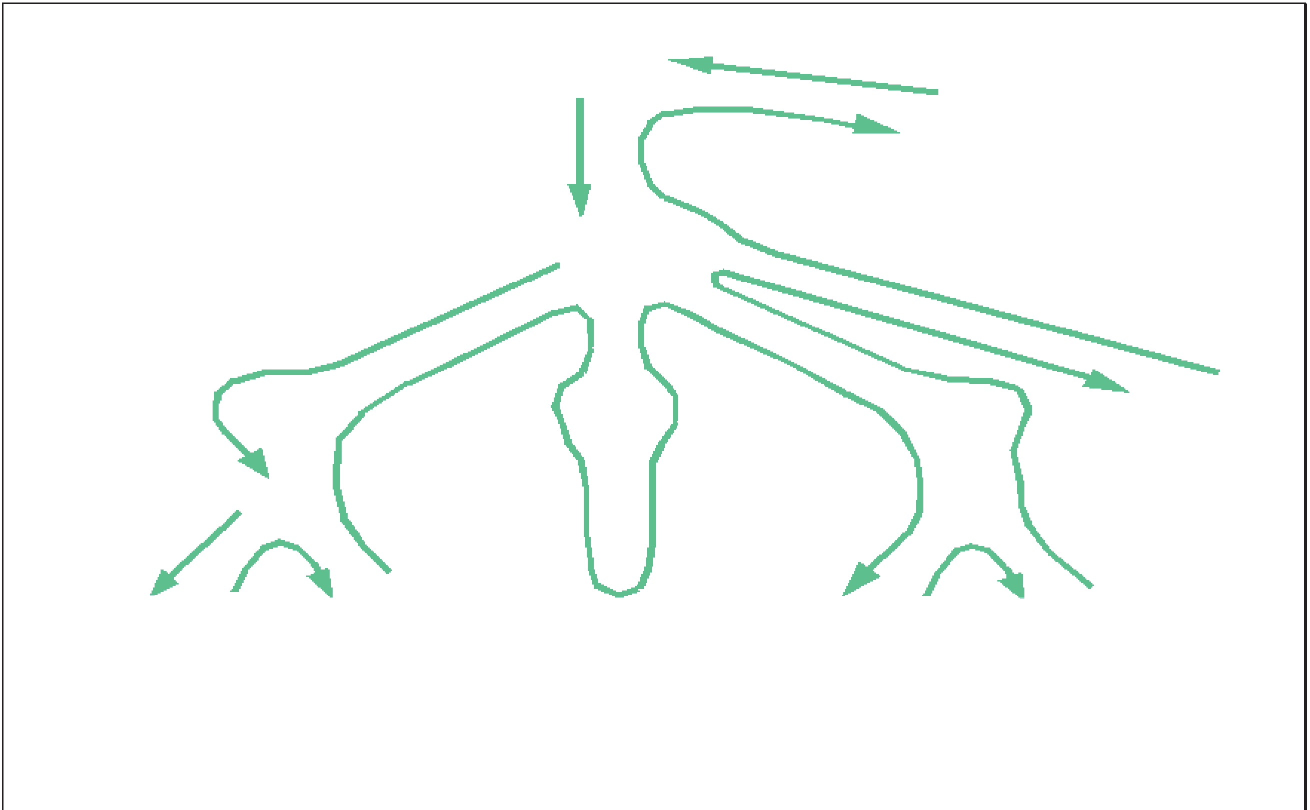
r(E)
no
p(E),q(E)
E = a
E = b
E = c
erfolglos ABT
q(a)
q(c)
erfolgreich WBT
erfolglos ABT
erfolglos ABT
erfolgreich WBT
erfolglos ABT

ABT = automatisches Backtracking WBT = gewillkürtes Backtracking

Grafische Darstellung der Tiefensuche des PROLOG-Systems nach Antworten auf die Anfrage `?- r(E).`

r(E)
no
p(E),q(E)
E = a
E = b
E = c
erfolglos ABT
q(a)
q(c)
erfolgreich WBT
erfolglos ABT
erfolglos ABT
erfolgreich WBT
erfolglos ABT

ABT = automatisches Backtracking WBT = gewillkürtes Backtracking



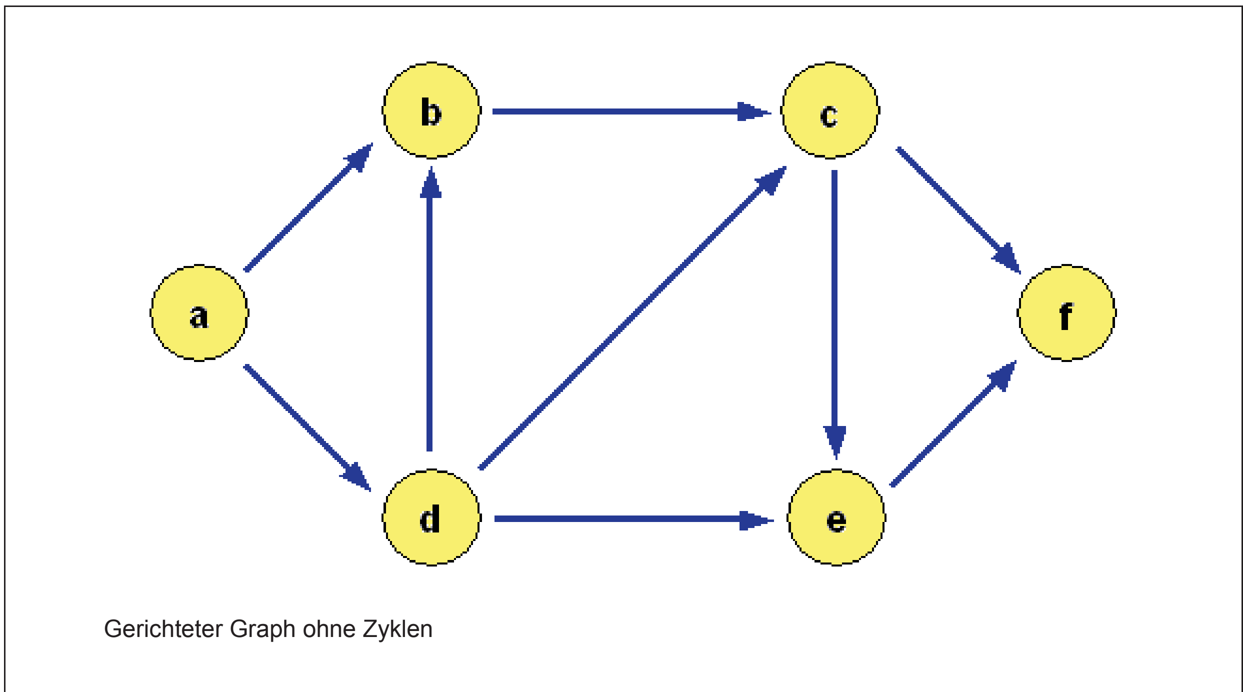
Relationale Datenbanken in Prolog

Tabelle Schüler		
Schülernummer	Name	Vorname
1	Bauer	Renate
2	Baumbach	Ilona
3	Franke	Frieder
7	Kunze	Ines
12	Maier	Peter
18	Schmidt	Sabine
19	Schulze	Thomas
24	Zimmermann	Uwe

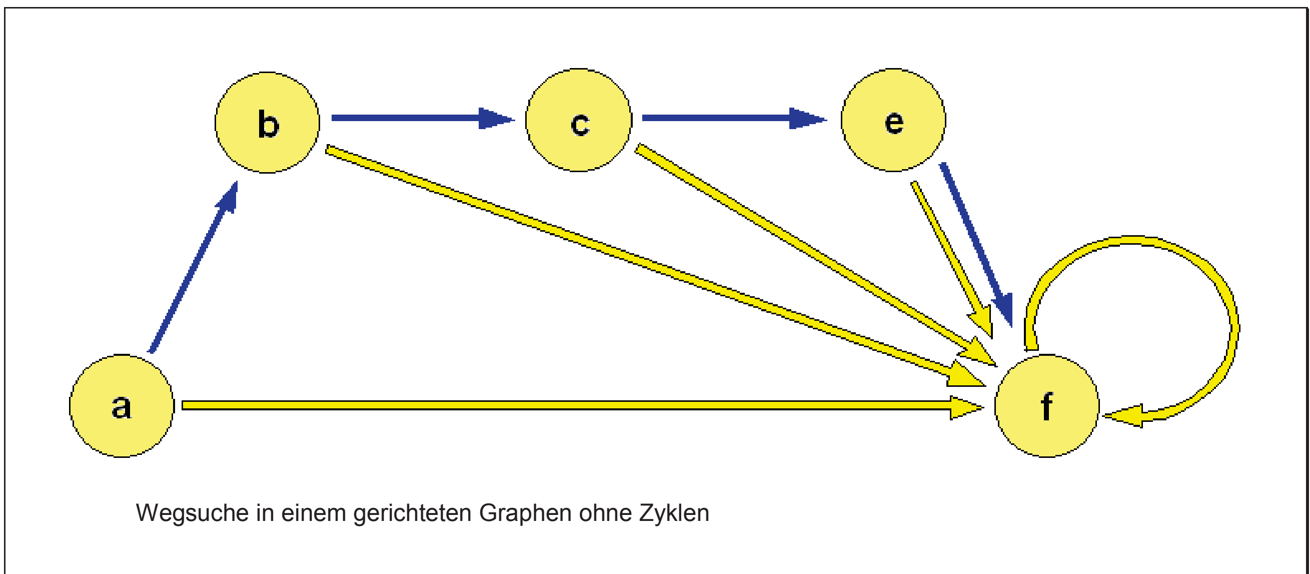
Tabelle Unterrichtsfach	
Fachnummer	Unterrichtsfach
1	Deutsch
2	Mathematik
3	Englisch
7	Chemie
8	Physik
11	Informatik

Tabelle Facharbeit		
Fach-arbeits-nummer	Thema	Fach-nummer
33	Kirchhoffsche Gesetze	8
35	Lösen linearer Gleichungssysteme	2
56	The life of Shakespeare	3
74	Goethes Schaffen in seiner Weimarer Zeit	1
106	Problemlösen mit Oberon	11
134	Elektronenkonfigurationen der Elemente	7

Tabelle Zuordnung	
Facharbeits-nummer	Schülernummer
33	19
33	7
35	19
56	3
74	24
106	2
106	18
134	1
134	12



Gerichteter Graph ohne Zyklen



Wegsuche in einem gerichteten Graphen ohne Zyklen

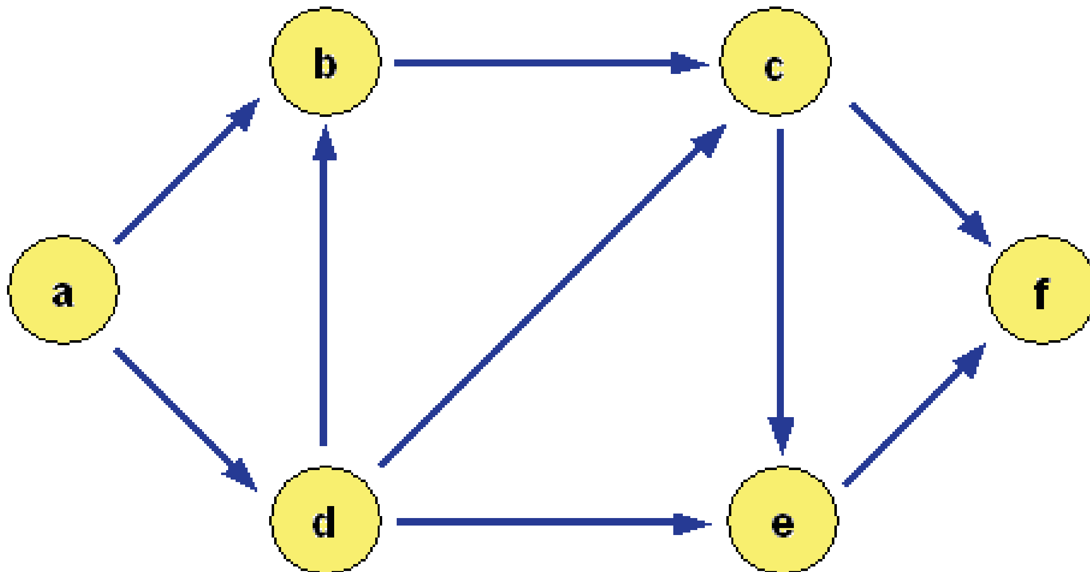
CARL FRIEDRICH GAUSS (1777-1855) erhielt mit neun Jahren von seinem Lehrer die Aufgabe die Zahlen 1 bis 60 zu addieren. Nach kurzer Zeit schrieb Carl Friedrich die Zahl 1830 auf seine Schiefertafel. Wie hat er das gemacht, dachte der Lehrer. Gauß erläuterte, er habe im Kopf gerechnet

	1,	2,	3,	4,	5,	...,	30
+	60,	59,	58,	57,	56,	...,	31
	61,	61,	61,	61,	61,	...,	61

und anschließend durch Multiplikation von 30 * 61 die Summe erhalten.

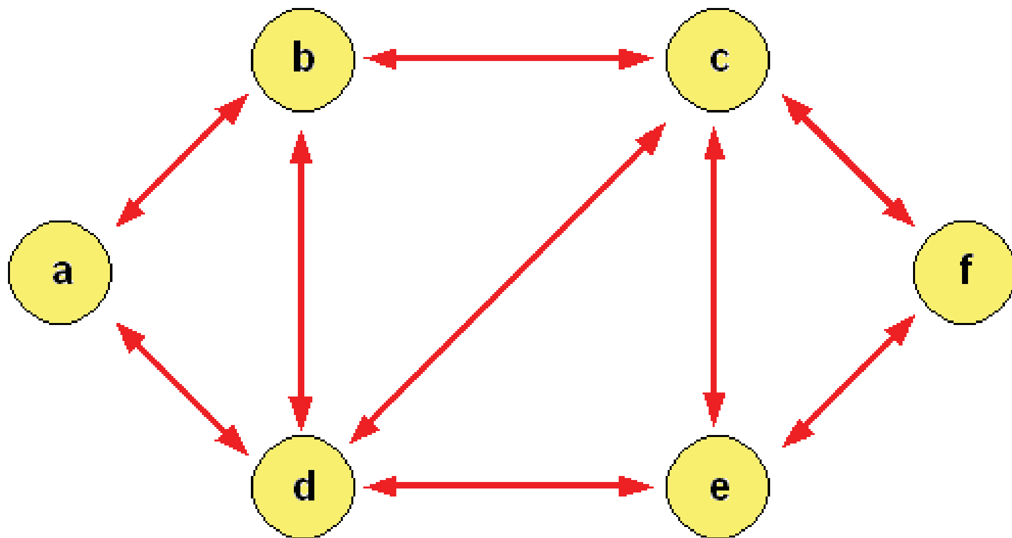
Wege vom Paradies-Bahnhof Jena zum Holzmarkt:

a	=	Paradies-Bahnhof	b	=	Hauptpost	c	= Goethe-Galerie
d	=	Gasthaus zum Roten Hirsch	e	=	Holzmarkt	f	= Rathaus

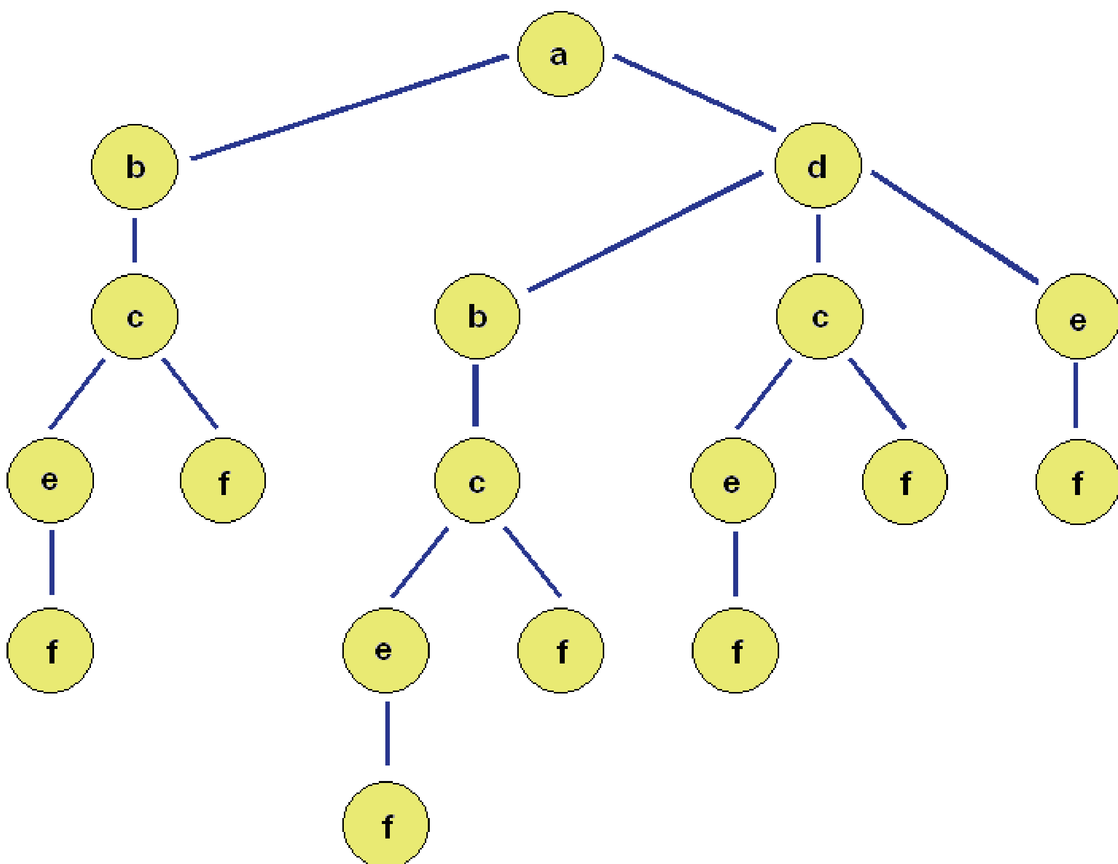


Gerichteter Graph ohne Zyklen

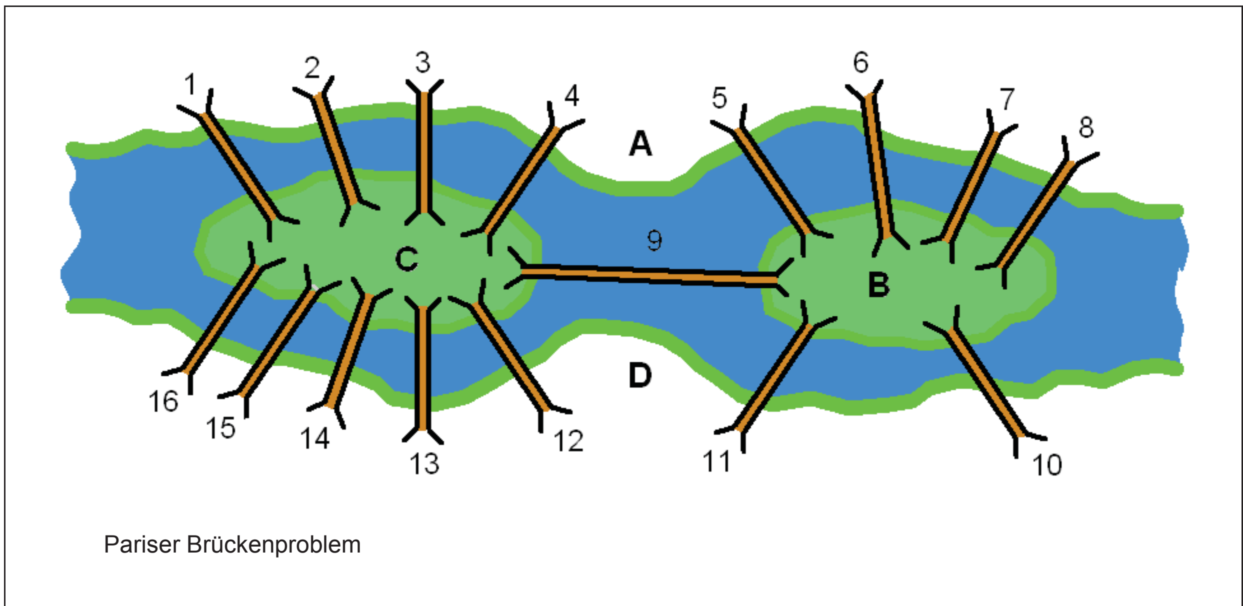
a	→	b	E.-Haeckel-Str.	b	→	c	Schillerstraße	c	→	f	Kollegiengasse
a	→	d	Neugasse	d	→	b	Grietgasse	d	→	c	Holzmarkt/Teichgraben
d	→	e	Holzmarkt	e	→	f	Löbderstraße	c	→	e	Teichgraben



Ungerichteter Graph



Darstellung aller Wege vom Startknoten a zum Zielknoten f



Pariser Brückenproblem

NORDEN

Westufer

Bruecke 5

Bruecke 4

Insel

Bruecke 3

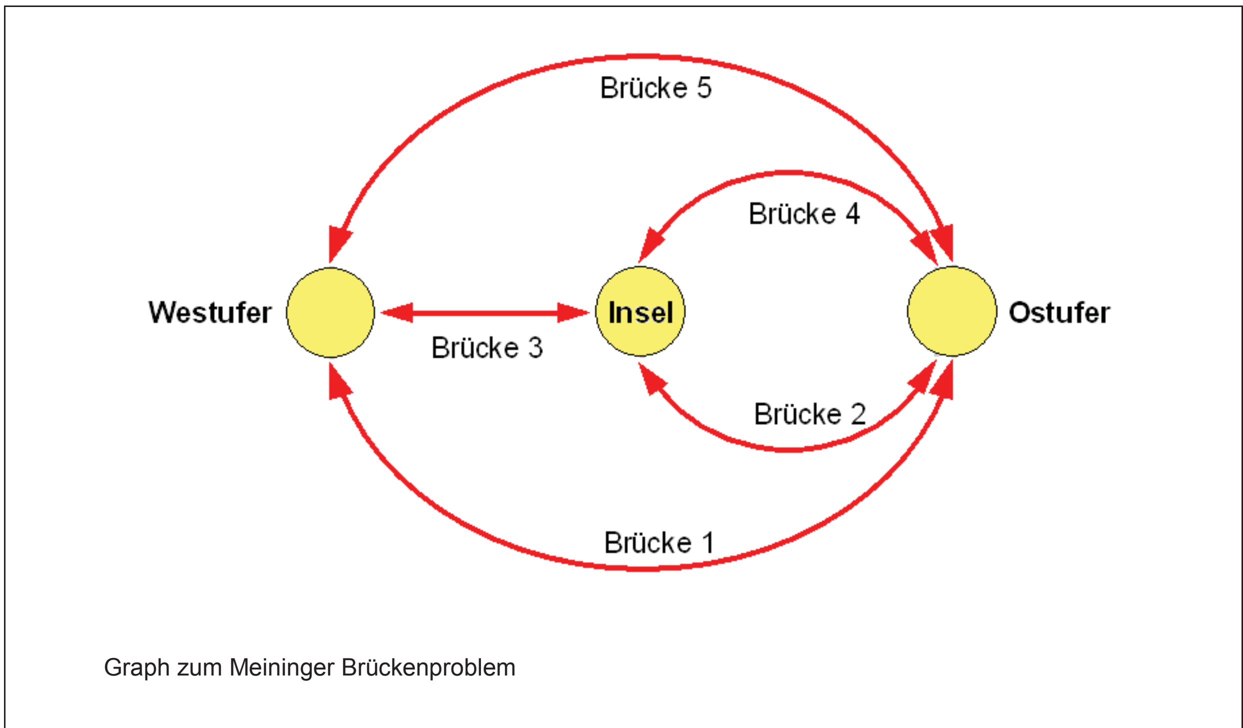
Ostufer

Bruecke 2

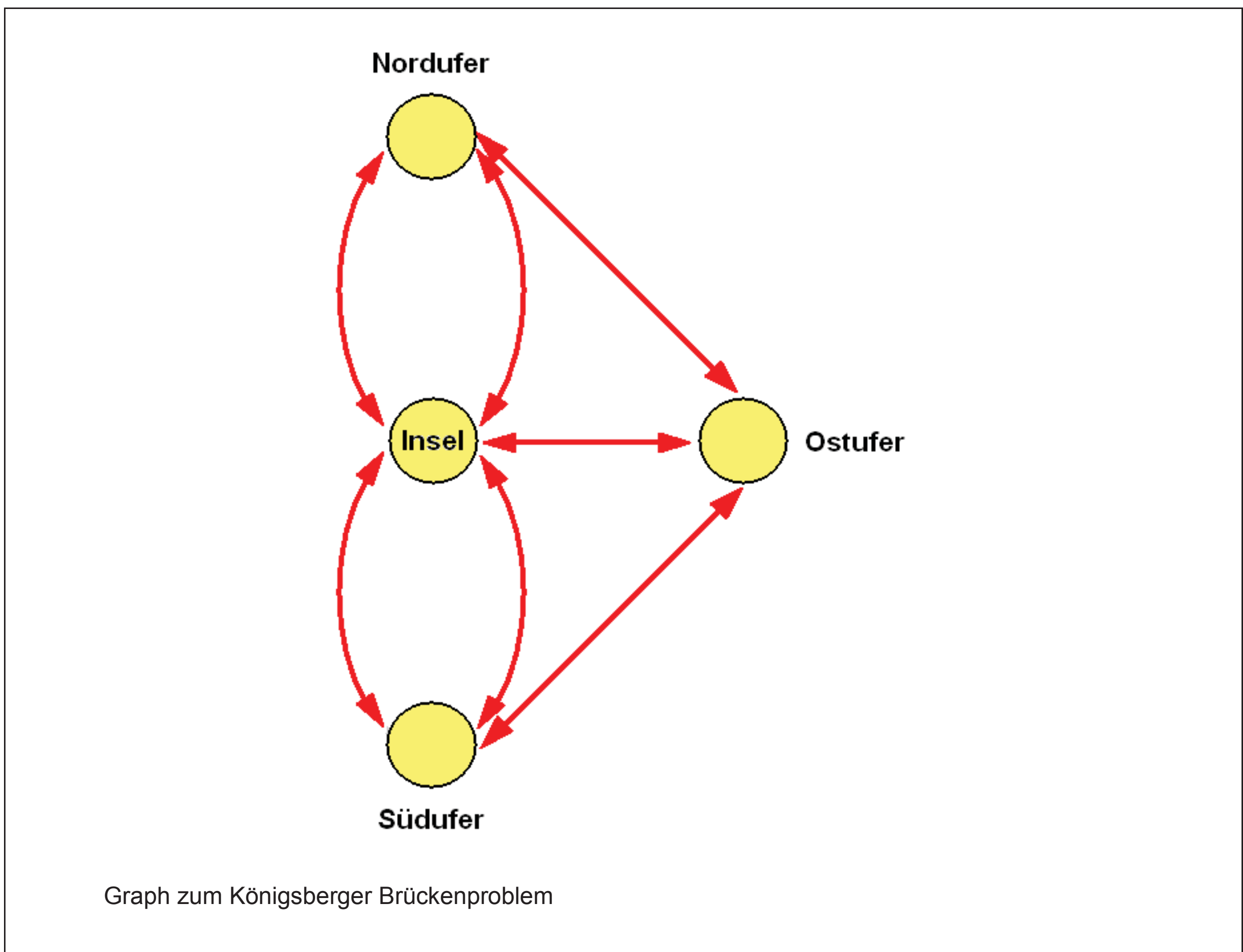
Bruecke 1

SÜDEN

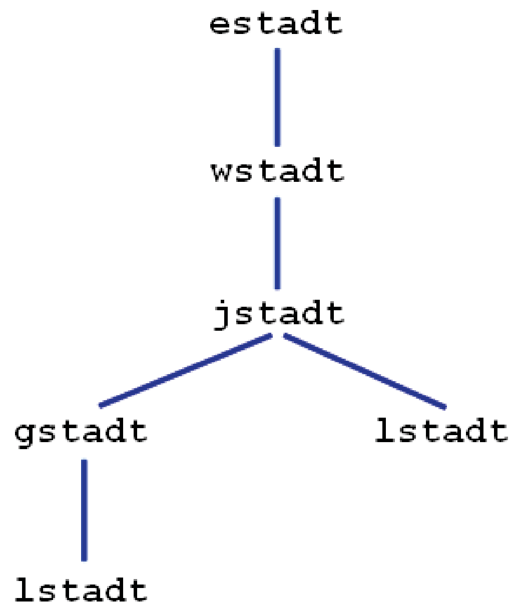
Meininger Brückenproblem



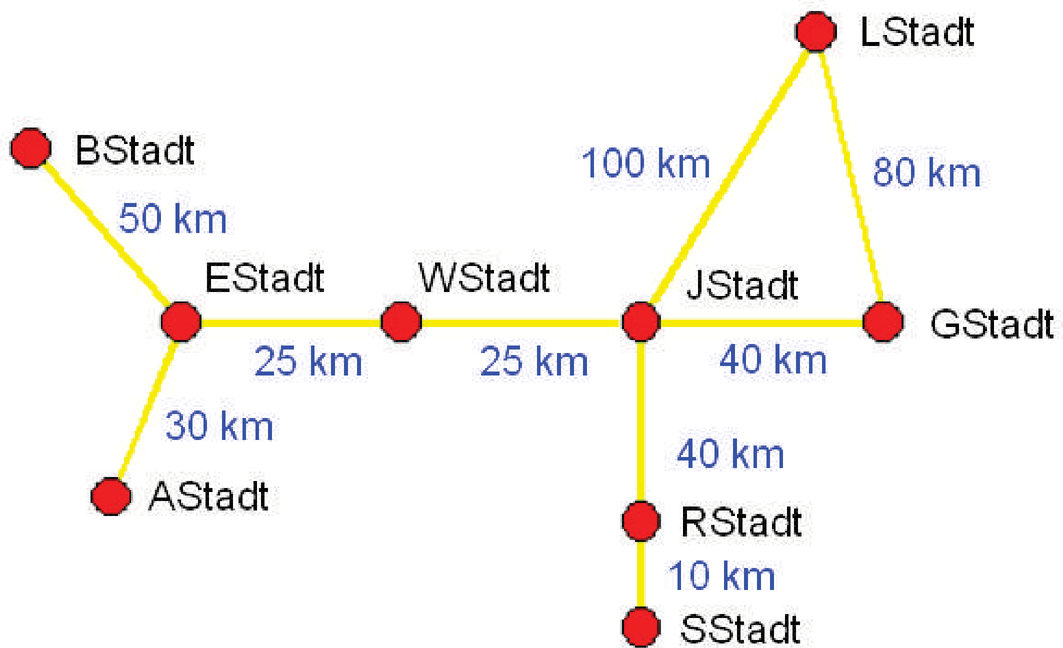
Graph zum Meininger Brückenproblem



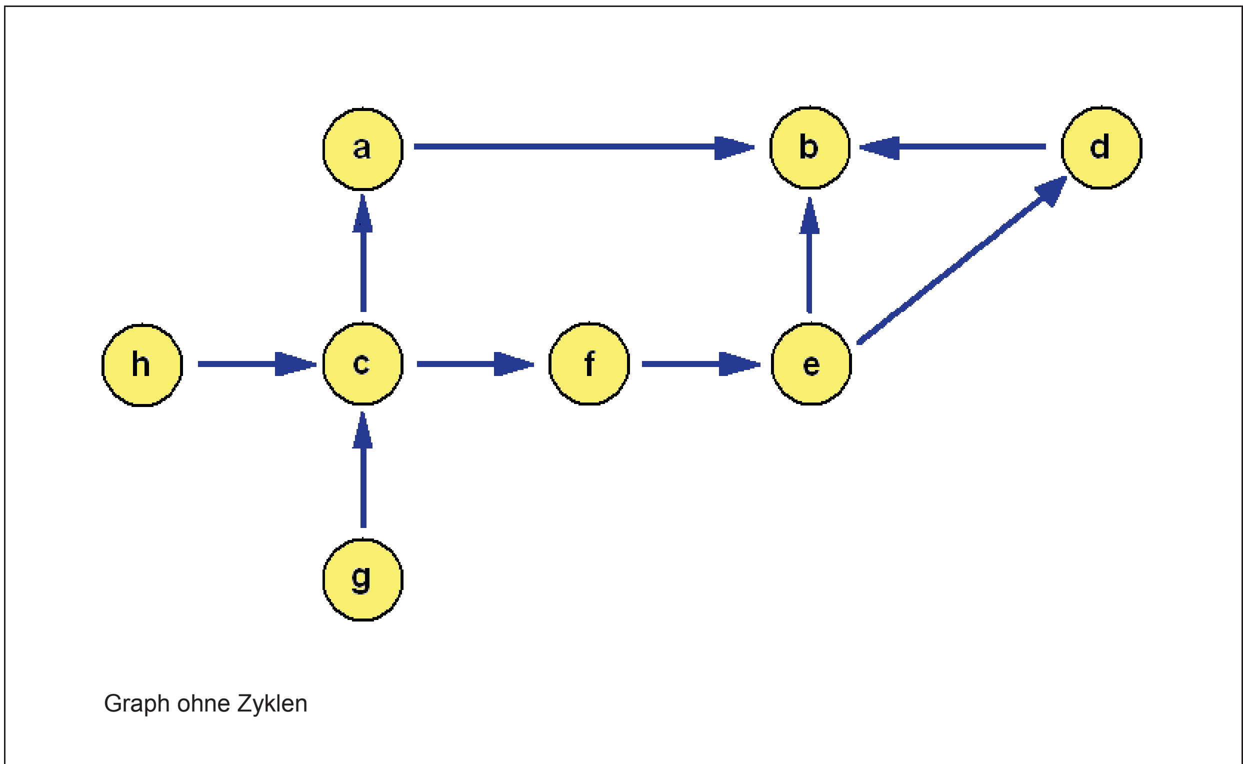
Graph zum Königsberger Brückenproblem



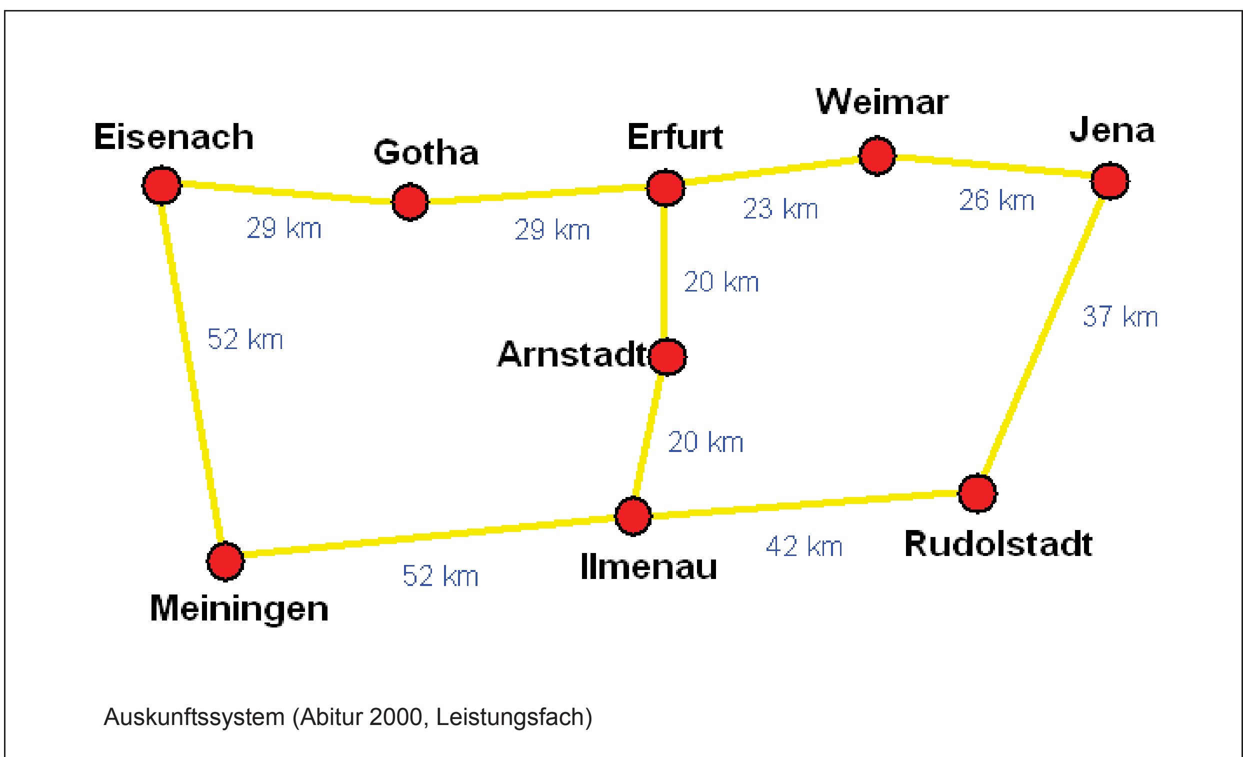
Darstellung der Verbindungen von estadt nach lstadt



Ungerichteter Graph (fiktives Streckennetz, nicht maßstabgerecht)

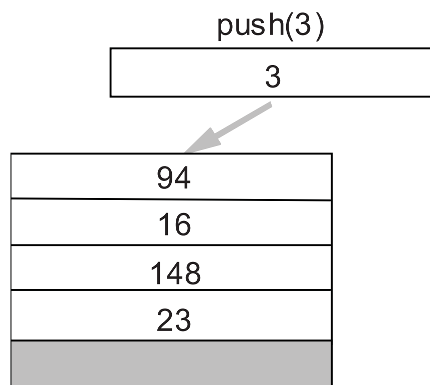


Graph ohne Zyklen

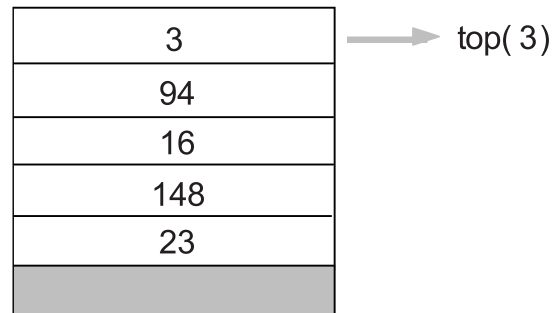


Auskunftssystem (Abitur 2000, Leistungsfach)

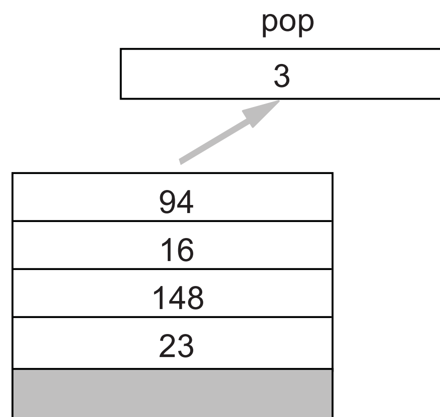
Leeren Stack erzeugen



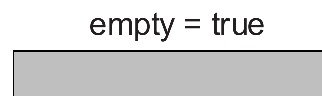
Zahl 3 als letztes (oberstes) Element in den Stack einfügen



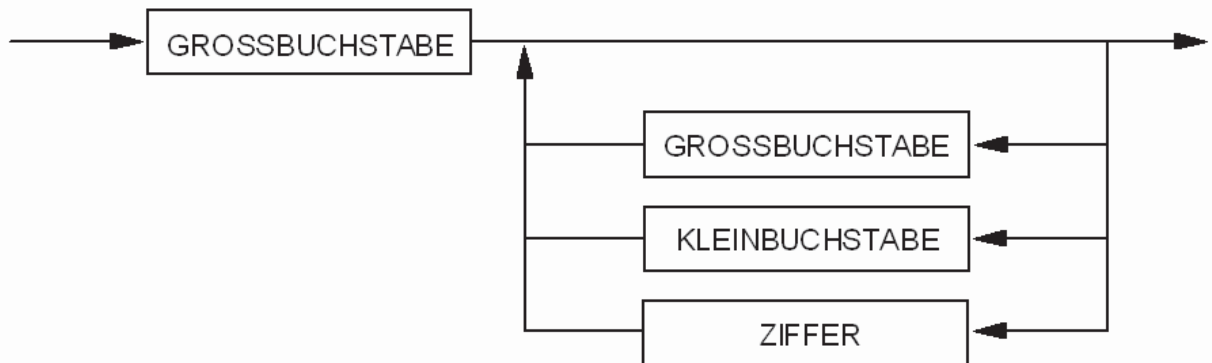
Liefert die Zahl E, die als letztes (oberstes) Element in den Stack eingefügt wurde

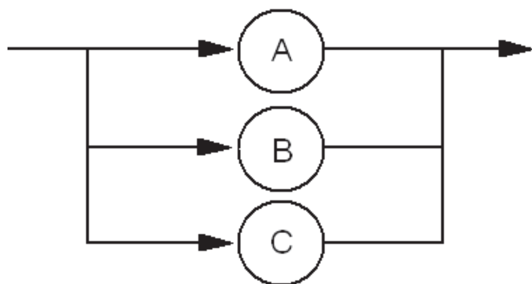


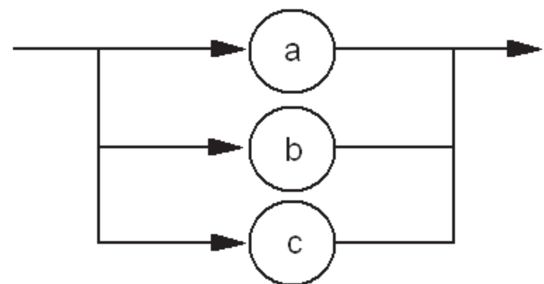
Entfernt das letzte (oberstes) Element aus dem Stack

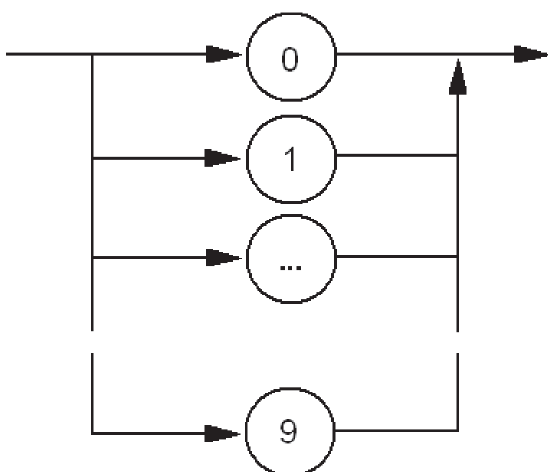


Wahr, wenn der Stack leer ist, sonst falsch

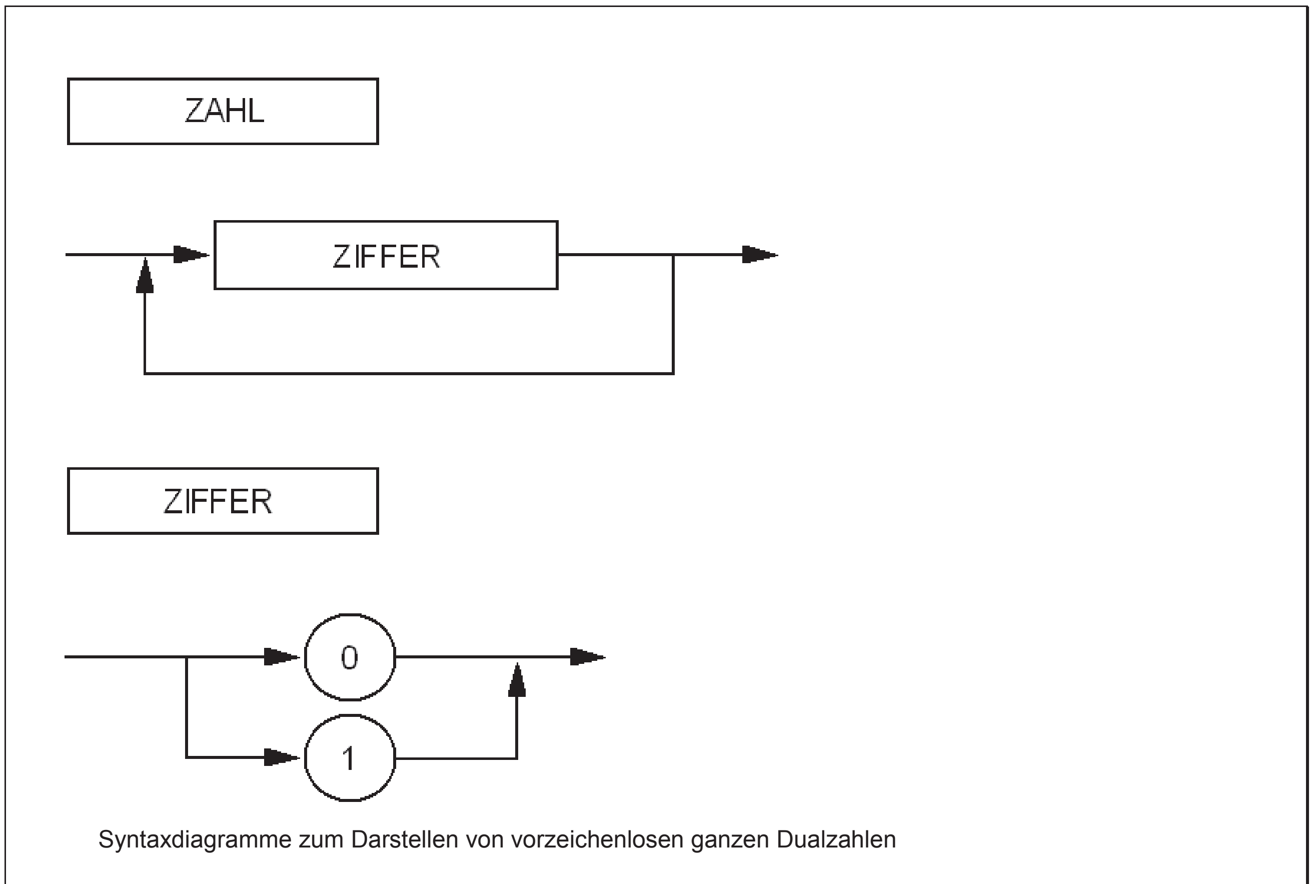




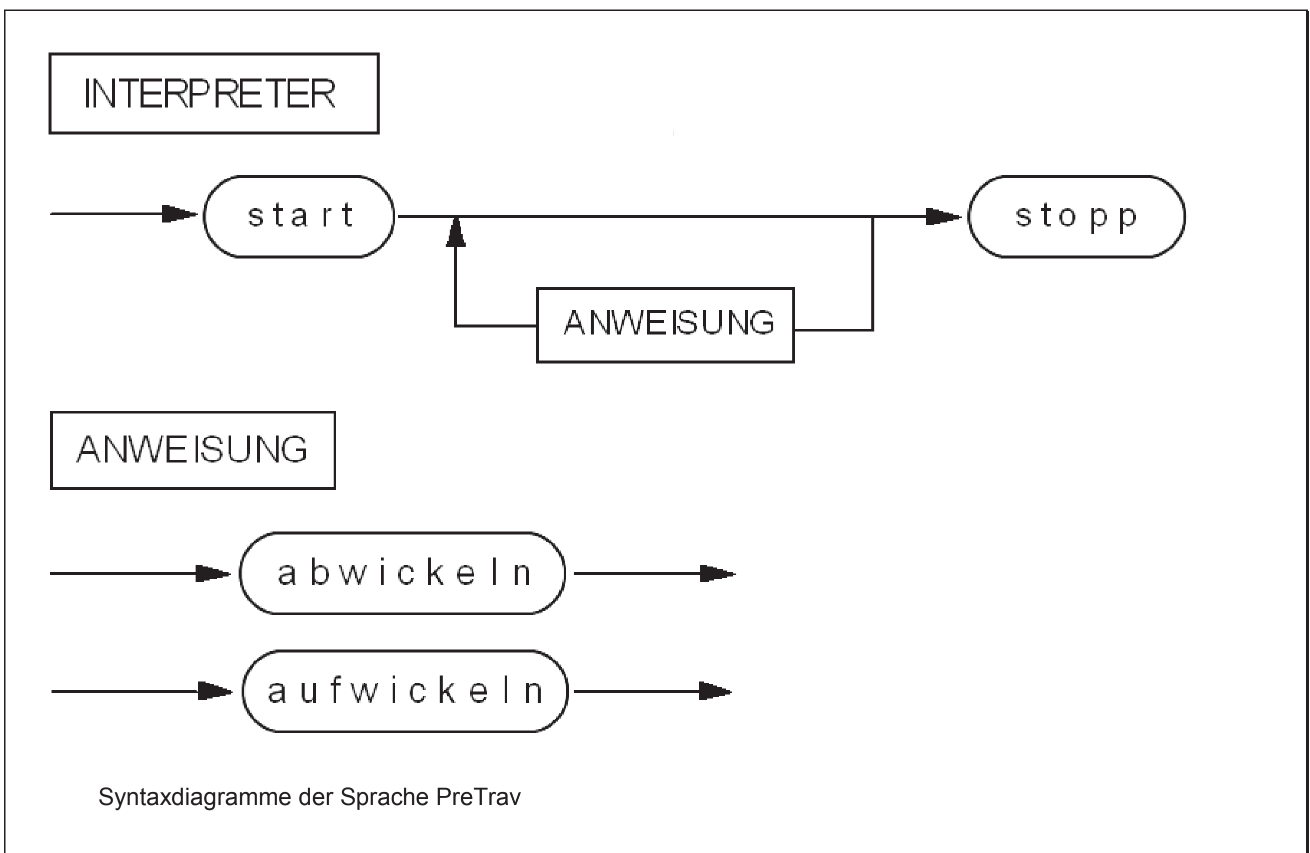




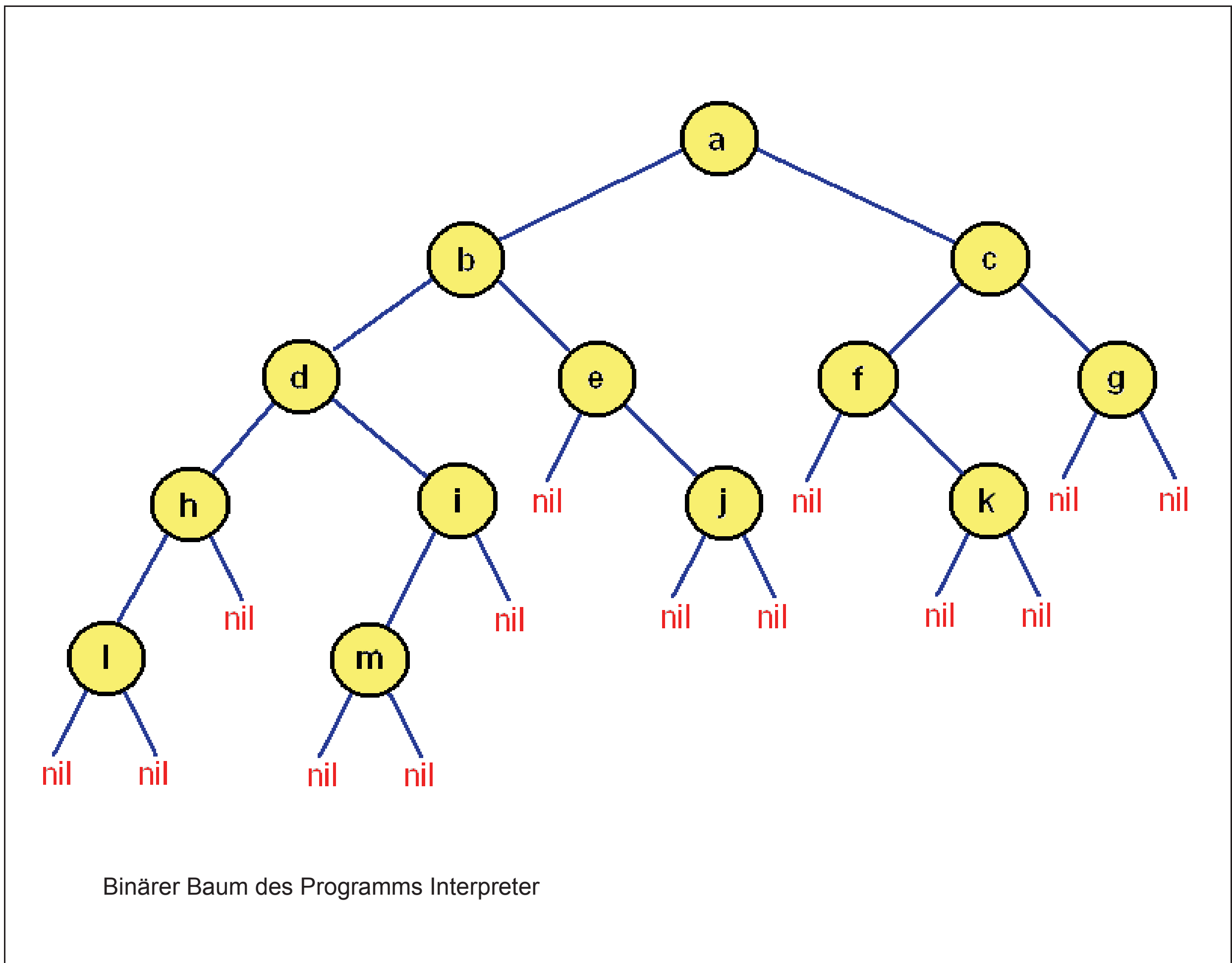
Syntaxdiagramme der Sprache L



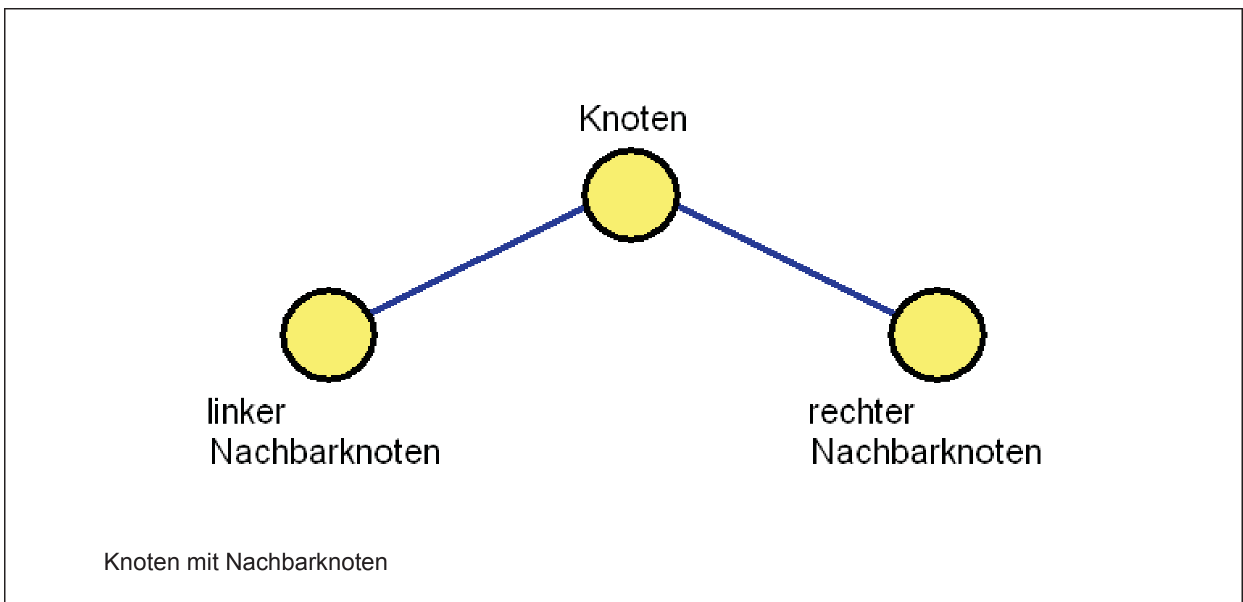
Syntaxdiagramme zum Darstellen von vorzeichenlosen ganzen Dualzahlen



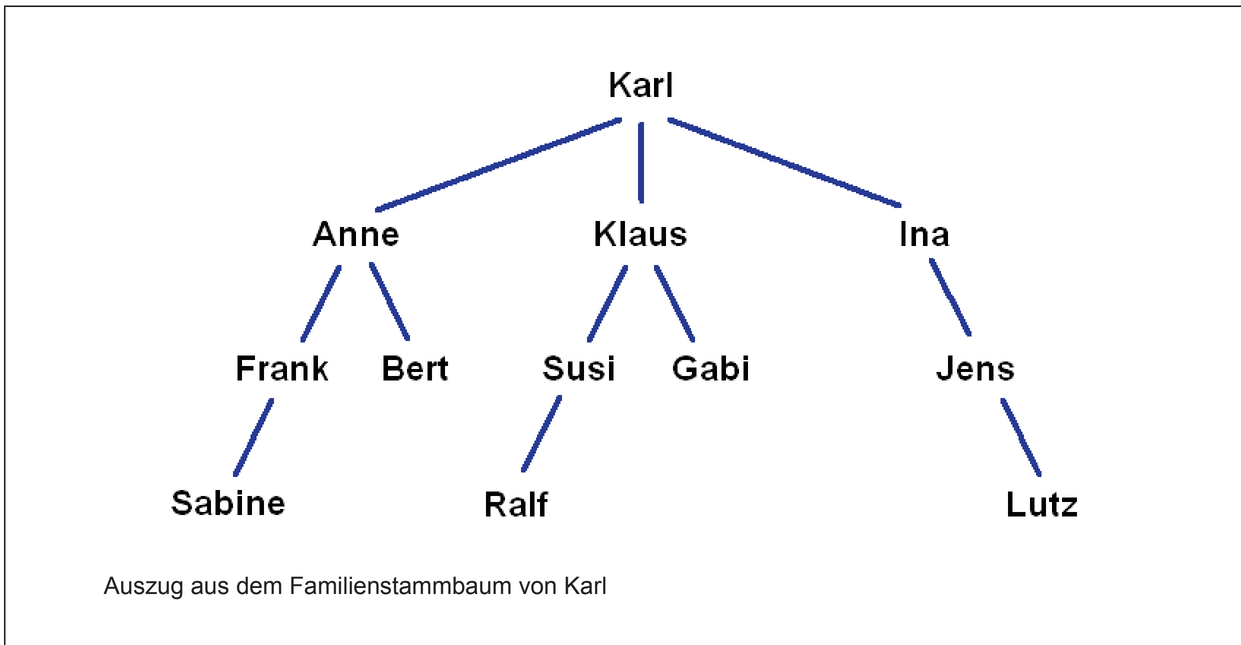
Syntaxdiagramme der Sprache PreTrav



Binärer Baum des Programms Interpreter



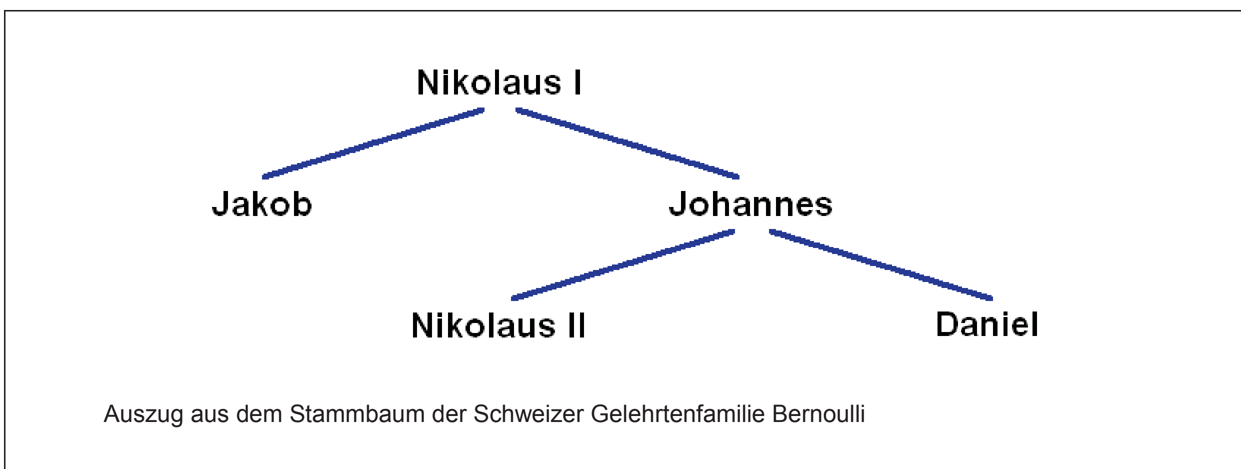
Knoten mit Nachbarknoten



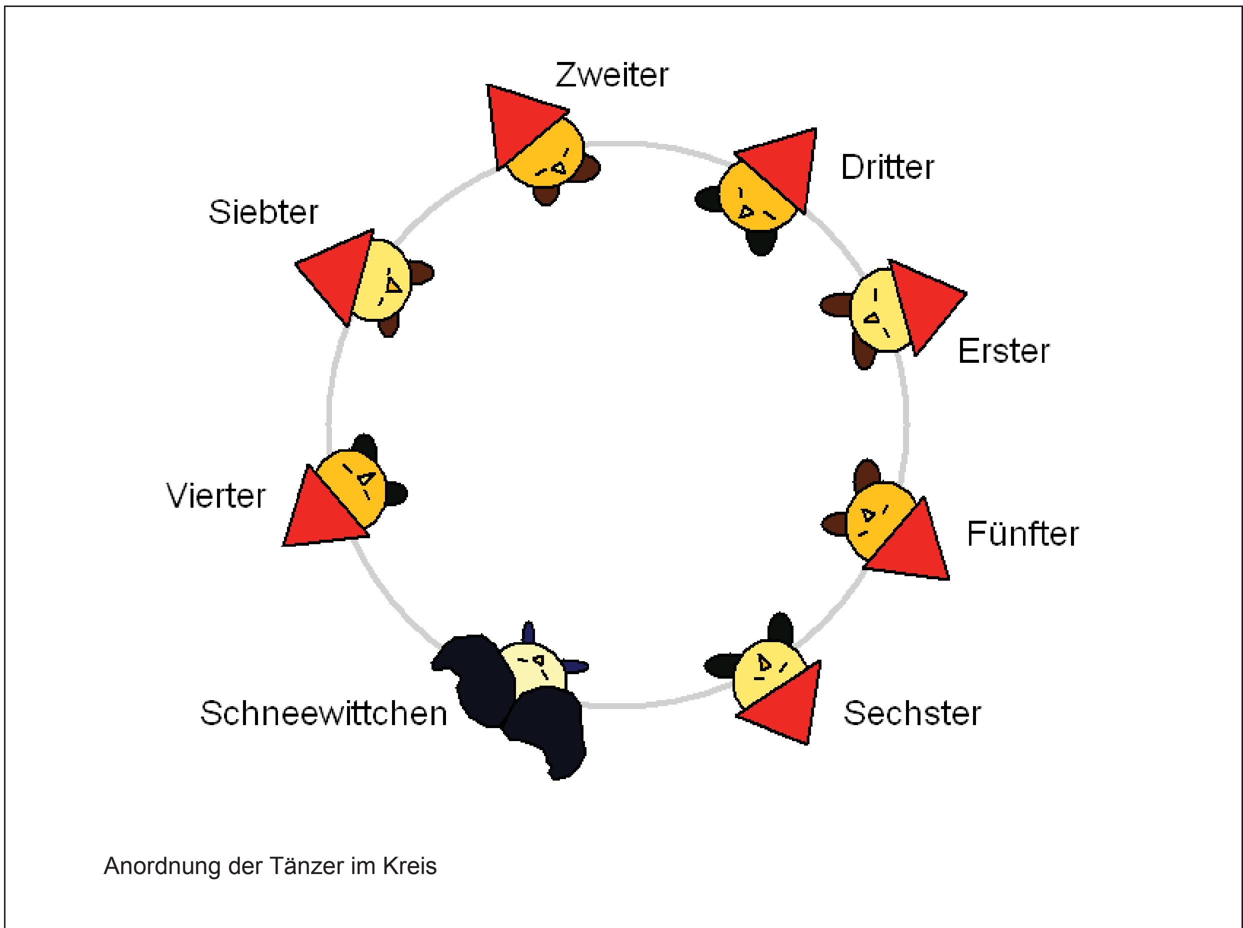
Auszug aus dem Familienstammbaum von Karl

Auszug aus dem Stammbaum der Musikerfamilie Bach

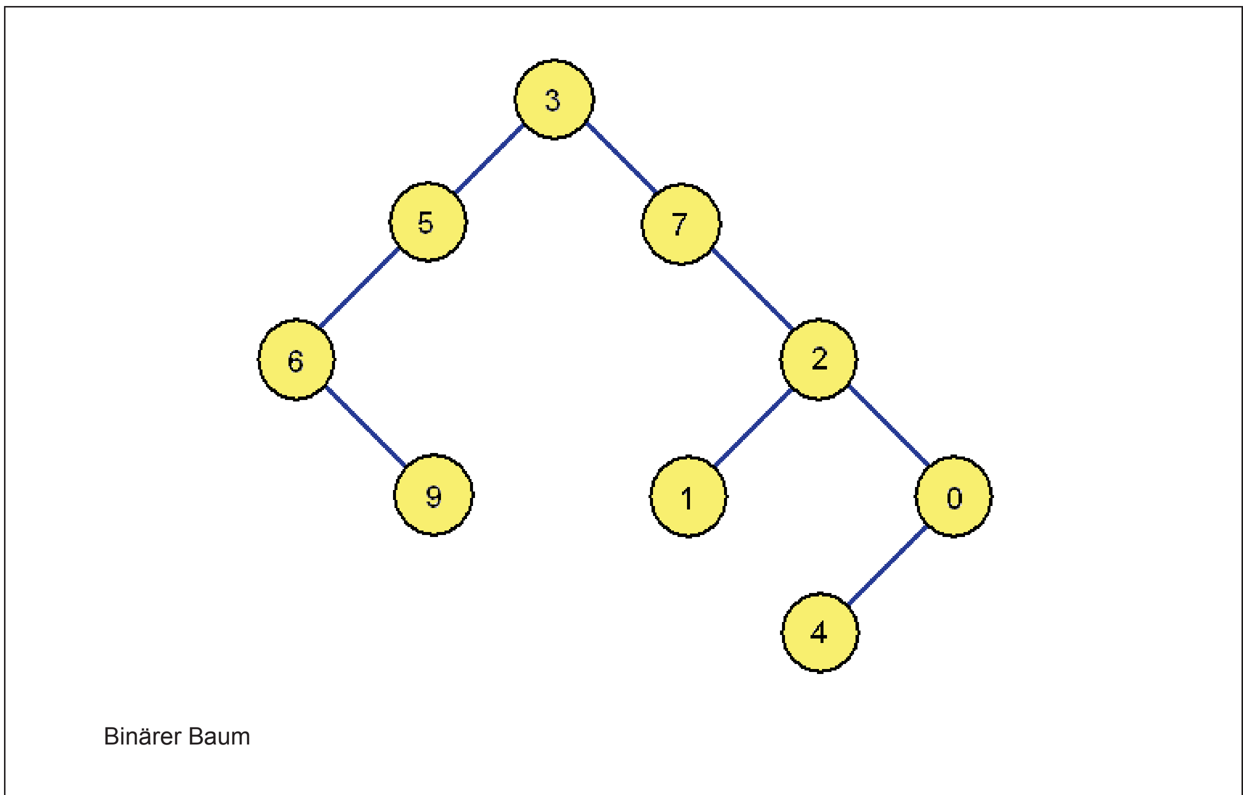
Christoph ist der Vater von Johann Ambrosius. Johann Ambrosius ist der Vater und Elisabeth die Mutter von Johann Christoph und Johann Sebastian. Maria Barbara ist die Mutter von Wilhelm Friedemann und Carl Philipp Emanuel. Anna Magdalena ist die Mutter von Johann Christoph Friedrich und Johann Christian. Johann Sebastian ist der Vater von Wilhelm Friedemann, Carl Philipp Emanuel, Johann Christoph Friedrich und Johann Christian.



Auszug aus dem Stammbaum der Schweizer Gelehrtenfamilie Bernoulli



Anordnung der Tänzer im Kreis



Binärer Baum

term

dualzahl

dualzahl

*

dualzahl

dualziffer

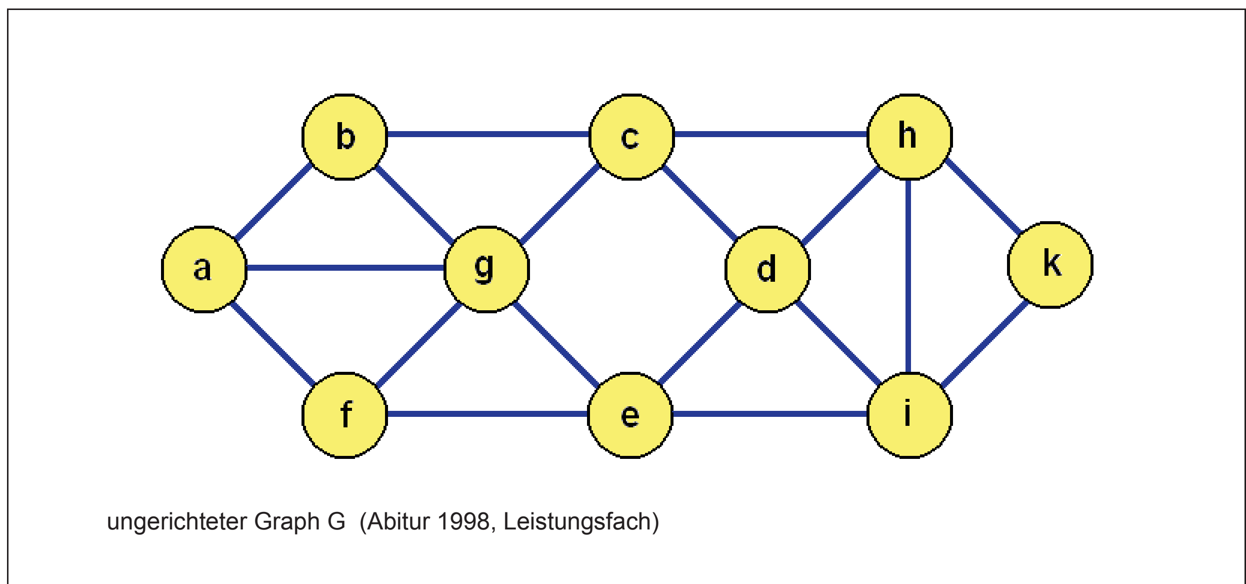
dualziffer

dualziffer

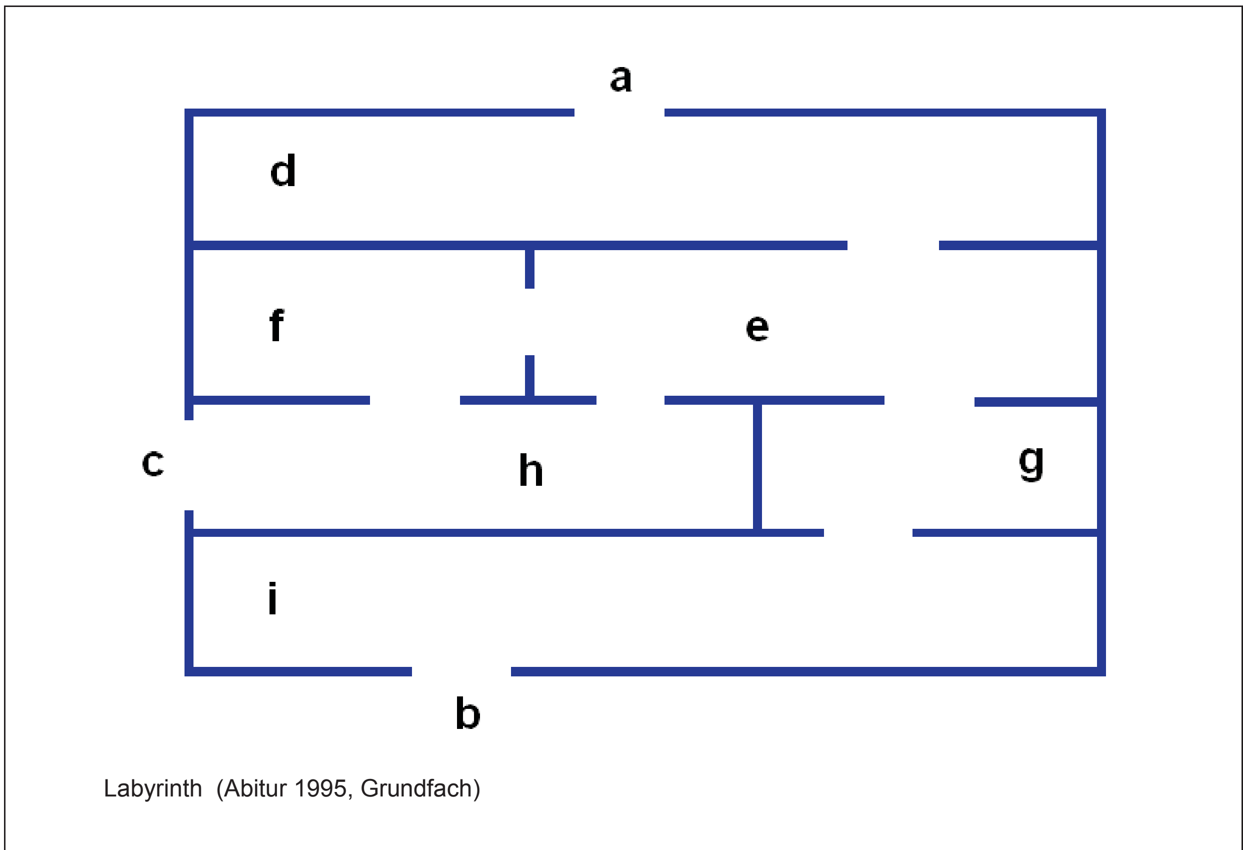
o

i

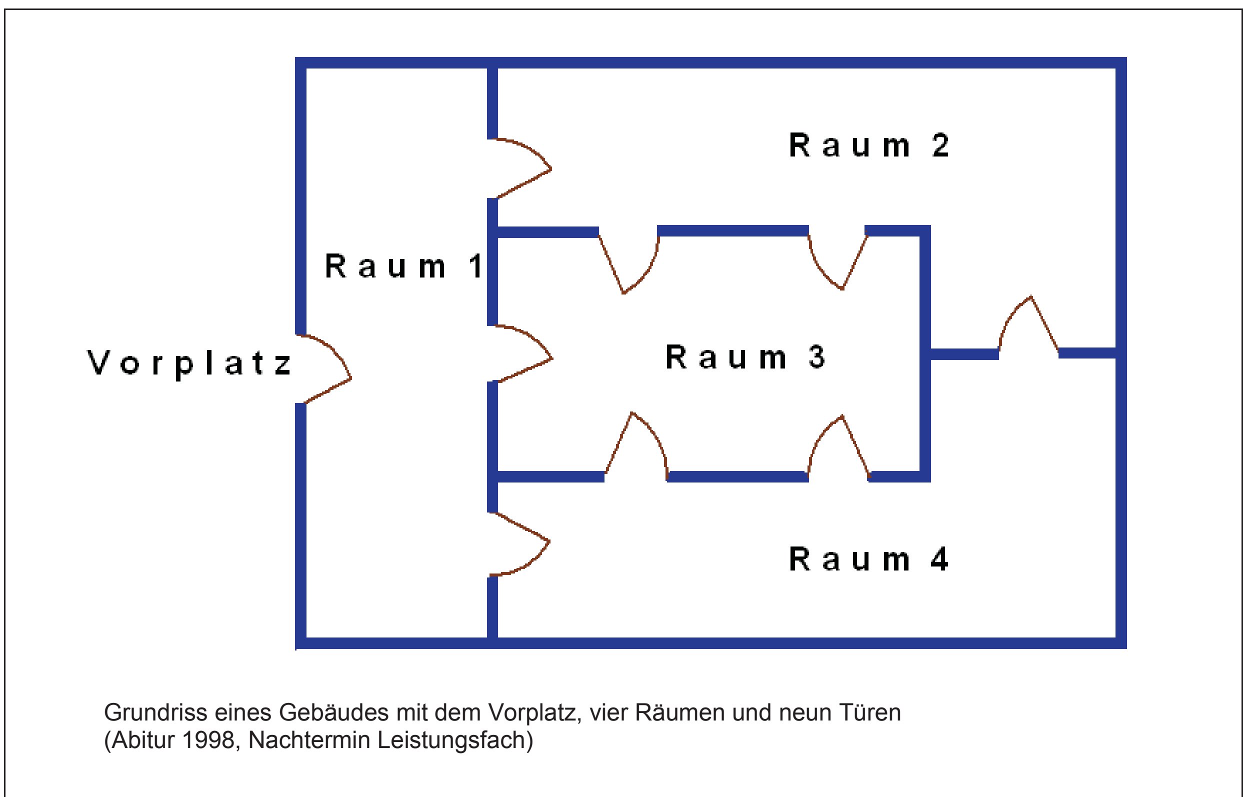
Vereinfachte Definition der Syntax von Term
(Dualziffern sind durch die Kleinbuchstaben o und i dargestellt)



ungerichteter Graph G (Abitur 1998, Leistungsfach)



Labyrinth (Abitur 1995, Grundfach)



Grundriss eines Gebäudes mit dem Vorplatz, vier Räumen und neun Türen
(Abitur 1998, Nachtermin Leistungsfach)

Anhang 2

Literaturverzeichnis und Quellenangaben

Wirth, N.: Compilerbau, Eine Einführung, Stuttgart, B. G. Teubner, 1986

Schüler-Duden, Die Informatik, Mannheim/Wien/Zürich, Dudenverlag, 1991

Statistisches Landesamt Thüringen i. G.: Statistisches Jahrbuch für Thüringen, Erfurt, 1991

Cordes, R.; Kruse, R.; Langendörfer; H., Rust, H.: Prolog, Eine methodische Einführung, Braunschweig/Wiesbaden, Vieweg, 1992

Geske, U.: Prolog, Grundlagen der Programmierung, Berlin, Akademie Verlag, 1993

Göhner, H.; Hafenbrak, B.: Arbeitsbuch PROLOG, Bonn, Fred. Duemmlers Verlag, 1993

Knülle-Wenzel, A.; Schütz, Ch.: Handbuch und Tutorial zu fïæ-PROLOG, Bochum, Selbstverlag, 1993

Agostini, F.: Weltbild's Mathematische Denkspiele, Augsburg, Weltbild Verlag, 1997

Thüringer Kultusministerium: Lehrplan für das Gymnasium, Informatik, Erfurt, 1999

Thüringer Kultusministerium: Abiturprüfung Grundfach Informatik, Erfurt, 1994 bis 2000

Thüringer Kultusministerium: Abiturprüfung Leistungsfach Informatik, Erfurt, 1994 bis 2000

Knülle-Wenzel; A., Schütz, Ch.: fïæ-Prolog, Version 3.1, 1989 - 1993

Colerus, E: Von Pythagoras bis Hilbert, Die Epochen der Mathematik und ihrer Baumeister. Augsburg, Weltbild Verlag, 1990

Clocksin, W. F.; Mellish, C. S.: Programmieren in Prolog. Berlin, Heidelberg, New York, Springer-Verlag, 1990

Drumm, H; Stimm, H.: Wissensverarbeitung mit PROLOG. Koblenz, Landesmedienzentrum Rheinland-Pfalz, 1995