1 Einleitung und Motivation

Am 16. Februar 2011 gewann IBMs Supercomputer Watson die Quizshow Jeopardy! gegen zwei menschliche frühere Gewinner. Dies stellt einen weiteren Meilenstein in der Geschichte der Künstlichen Intelligenz (KI) dar. Teile dieses Supercomputers wurden mit Hilfe der logischen Programmiersprache Prolog entwickelt. (vgl. [8]). Gerade in vielen Teilbereichen der KI, z.B. beim Verarbeiten von natürlicher Sprache (Natural Language Processing (NLP)) findet Prolog immer wieder Anwendung.

Die bereits Anfang der 1970er entwickelte Sprache Prolog ist auch heute noch eine der wichtigsten logischen Programmiersprachen. Ein Prolog-Programm besteht aus einer Menge an Hornklauseln (auch Wissensbasis genannt). (vgl. [4, S. 2]) Somit ist das Prolog zugrundeliegende Konstrukt die Hornklausel, welches auch die Basis dieser Arbeit darstellt.

Was ist eine Hornklausel?

Eine Hornklausel ist definiert als eine prädikatenlogische Formel, die lediglich aus einer Disjunktion von einem positiven und beliebig vielen negativen Literalen (null- oder mehrstellige Prädikate) besteht: (vgl. [15, S. 734])

$$\neg A_1 \vee ... \vee \neg A_n \vee B$$

Durch Äquivalenzumformungen erhält man eine Implikation mit der Konjunktion der negativen Literale als Voraussetzung und dem positiven Literal auf der rechten Seite:

$$A_1 \wedge ... \wedge A_n \rightarrow B$$

Im Folgenden wird B als Kopf und A_i als Teilziele bezeichnet. Stellt man eine Anfrage (ebenfalls ein Literal) an das System, so versucht es, aus der vorhandenen Wissens-

basis das Literal herzuleiten. Gelingt dies, liefert das System $true$ und eventuell die entsprechende Variablenbindung, sonst $false$.

Parallelität bei der Herleitung

Die traditionellen Herleitungsmechanismen bei Programmiersprachen wie Prolog sind meist streng sequentiell. Das heißt, die Laufzeitumgebung probiert nacheinander von oben nach unten jede Hornklausel aus. Innerhalb der Hornklausel werden die Teilziele von links nach rechts getestet. Deshalb haben sich immer wieder Forscher damit beschäftigt, wie man eine parallele Herleitung realisieren kann. Dabei gibt es vier grundsätzliche Arten der Parallelität bei Hornklauselprogrammen ([6]):

- **ODER-Parallelität**
- **UND-Parallelität**
- **Strom-Parallelität**
- **Such-Parallelität**

In der folgenden Arbeit wird ausschließlich UND-Parallelität betrachtet. Damit gemeint ist die gleichzeitige Herleitung von Teilzielen innerhalb einer Hornklausel, falls diese unabhängig von einander sind. Ein Ansatz der, die UND-Parallelität ausnutzt, ist das Datenflussmodell nach Schwinn ([13] , 1988). Diese Arbeit basiert vollständig auf diesem Modell.

Motivation dieser Arbeit

Das Modell nach Schwinn existiert nun bereits seit über 25 Jahren. Seitdem gab es im Bereich des Übersetzerbaus, wie in quasi allen Bereichen der Informatik, sehr viele neue Technologien. Es existieren heute Ansätze und Werkzeuge, die damals noch nicht existiert haben. Diese Arbeit soll das Modell aufgreifen und einen Übersetzer mit moderneren Werkzeugen entwerfen. Anschließend wird evaluiert, ob das Modell heute noch Relevanz hat und ob andere Forscher eventuell andere bzw. bessere Ansätze gefunden haben.

2 Analyse

Dieses Kapitel analysiert das Datenflussmodell nach [13]. Dazu werden zunächst Hornklausel-Programme im Allgemeinen betrachtet und die Prolog-Notation eingeführt (Abschnitt 2.1). In den folgenden Abschnitten wird dann das Datenflussmodell näher untersucht (Abschnitt 2.2. Schwinn erzeugt für jede Klausel einen Datenflussgraphen (ein besonderer gerichteter azyklischer Graph). Es werden zunächst die verschieden Knotentypen im Graphen analysiert. Anschließend werden die möglichen Abhängigkeitstypen, die bei der Übersetzung unterschieden werden müssen, betrachtet. Letztlich wird noch die Repräsentation des Datenflussgraphen (also das Ausgabeformat des Übersetzers) analysiert. Der Abschnitt 2.3 analysiert den Übersetzungsalgorithmus, der bereits in [13] gegeben ist. Dieser bildet dann die Grundlage für die Implementierung (Kapitel 3).

2.1 Hornklauselprogramme

Ein Hornklauselprogramm besteht aus einer endlichen Liste von Hornklauseln. Abbildung 2.1 zeigt die Funktion eines Prolog-Interpreters. Das "Programm" wird in diesem Zusammenhang auch Wissensbasis genannt. Die Wissensbasis wird vom Prolog-Interpreter verwendet, um Anfragen zu beantworten.

Wir betrachten die folgende Liste von Hornklauseln:

$$\begin{aligned} vater(X,Y) \wedge vater(Y,Z) &\rightarrow grossvater(X,Z) \\ \top &\rightarrow vater(alice, bob) \\ \top &\rightarrow vater(bob, charlie) \end{aligned}$$

Die zweite und dritte Hornklausel wird Fakt genannt, da sie keine Voraussetzung hat. Die erste Hornklausel nennt man Regel. In diesem Fall stellt diese Regel die

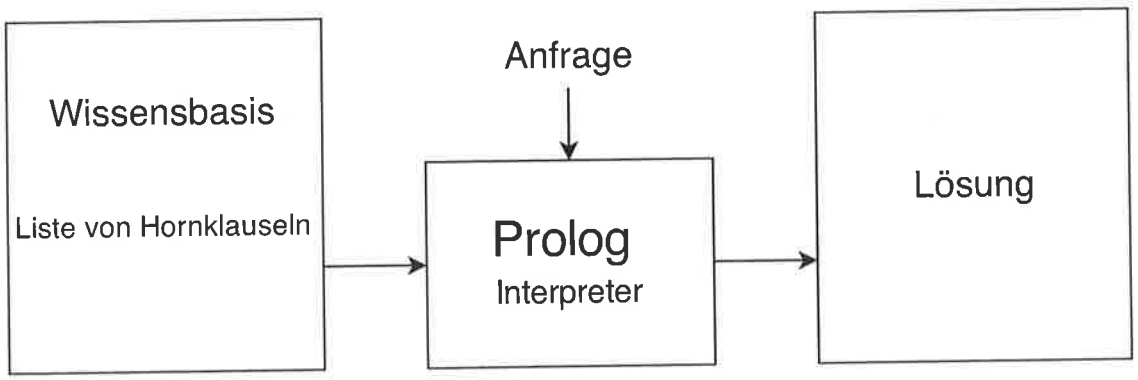

Abbildung 2.1: Funktion eines Prolog-Interpreter

Großvaterbeziehung dar: Z ist der Großvater von X, wenn ein Y existiert, welches Vater von X und Kind von Z ist. Dies lässt sich direkt in die Prolog-Notation übertragen:

```
1 grossvater(X,Z) :- vater(X,Y), vater(Y,Z).
2 vater(carl, bob).
3 vater(bob, charlie).
```

Listing 2.1: Einfaches Hornklausel-Programm in Prolog-Notation

Dabei beginnen Variablen stets mit einem Großbuchstaben und Konstanten mit einem Kleinbuchstaben. Nun kann dieses Programm im Prolog-Interpreter geladen und es Fragen gestellt werden:

$?- grossvater(carl, charlie)$
$true$

$?- grossvater(carl, bob)$
$false.$

$?- grossvater(carl, X)$
$X = charlie.$

2.2 Datenflussmodell

Das Datenflussmodell nach Schwinn [13] beschreibt, wie aus einem Hornklauselprogramm ein Reihe von Datenflussgraphen (spezielle gerichtete azyklische Graphen) erzeugt werden können, die dann von einem Interpreter verwendet werden, um Anfragen zu beantworten. Dabei wird jede Regel und jeder Fakt von einem Graphen repräsentiert. Das Modell ermöglicht die Ausnutzung der UND-Parallelität und intelligentes Backtracking.
Diese Arbeit beschreibt die Entwicklung eines Compilers, der Hornklauselprogramme in Datenflussgraphen und als Datenflusscode ausgibt. Dieser Datenflusscode kann dann an einen passenden Interpreter weitergegeben werden.
Zunächst betrachtet Abschnitt 2.2.1 die verschiedenen Knotentypen des Datenflussgraphen. Im Folgenden werden die notwendigen Schritte der Übersetzung analysiert. Abschnitt 2.2.2 stellt die verschiedenen Abhängigkeitsarten von Variablen innerhalb einer Hornklausel vor, und welche Datenflussgraphen dafür erzeugt werden müssen. Abschnitt 2.2.3 analysiert dann Beispieldatenflussmodelle, die in [13] gegeben sind. Abschließend beschreibt Abschnitt 2.2.4, wie die Datenflussgraphen in einem Format repräsentiert werden, welches an den Interpreter übergeben werden kann.

2.2.1 Knotentypen

Schwinn [13] verwendet in seinem Datenmodell 7 Knotentypen. Zur Laufzeit wird der Datenflussgraph von Datenflusspaketen (sogenannten Tokens) durchlaufen. Diese werden von Knoten zu Knoten "weitergegeben". Die Knotentypen haben also zur Laufzeit jeweils ein anderes Verhalten. Jeder Knoten hat 1-2 Eingabeports und 0-2 Ausgabeports (außer der Copy-Knoten). Auch wenn die Bedeutung der Knotentypen zur Entwicklung des Übersetzers eher weniger Relevanz hat, werden die Knotentypen kurz analysiert (siehe auch Abbildung 2.2):

- **Entry-Unification**
  Eingabeports: 2
  Ausgabeports: 2
  Startknoten jedes Graphen. Versucht das am linken Eingang anliegende Aufruftoken mit dem am rechten Eingang permanent anliegenden Kopf der Hornklausel zu unifizieren. Gelingt dies, werden an die beiden Ausgabeports Tokens angelegt, die das Herleiten der Teilziele anstoßen.

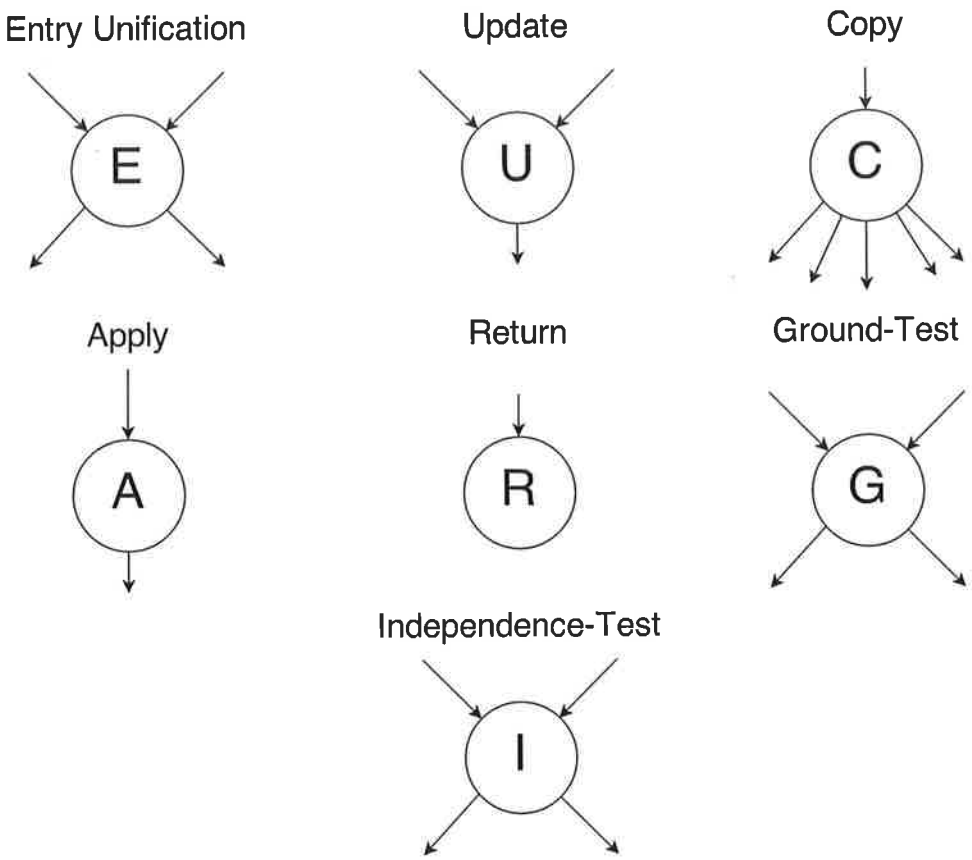

Abbildung 2.2: Knotentypen im Datenflussmodell nach [13]

- **Update**
  Eingabeports: 2
  Ausgabeports: 1
  Dient zum Austausch von Variablenbindungen.

- **Copy**
  Eingabeports: 1
  Ausgabeports: n
  Kopiert das Token, das am Eingang anliegt.

- **Apply**
  Eingabeports: 1
  Ausgabeports: 1
  Herleitung eines Teilziels in eigenständigem Prozess.

- **Return**
  Eingabeports: 1
  Ausgabeports: 0
  Letzte Aktion im Graphen. Dient zur Rückkehr zum Aufrufer.

- **Ground-Test**
  Eingabeports: 2
  Ausgabeports: 2
  Testet, ob eine oder mehrere Variablen an einen Grundterm gebunden sind. Ein Grundterm ist ein Term, in dem keine Variablen, sondern nur noch Konstanten vorkommen. Dabei ist der linke Eingang der Triggereingang, hier kommt das Token an, bei dem die Bindung getestet werden soll. Am rechten Eingang liegt permanent eine Liste der zu überprüfenden Variablen an. Sind alle Variablen gebunden, so wird ein Token an den rechten Ausgabeport gelegt, ansonsten an den linken.

- **Independence-Test**
  Eingabeports: 2
  Ausgabeports: 2
  Testet, ob Bindungen zwischen Paaren von Variablen vorhanden sind. Ähnlich wie beim Ground-Test ist der linke Eingangsport der Triggerport und am rechten Eingabeport liegt permanent eine Liste von Variablenpaaren an, die überprüft werden sollen. Die Ausgabeports werden identisch verwendet wie beim Ground-Test.

Die beiden Testknoten werden benötigt, um zur Laufzeit Abhängigkeiten oder Unabhängigkeiten zwischen Teilzielen festzustellen. Die möglichen Abhängigkeitsarten, die dabei auftreten können, werden im nächsten Abschnitt diskutiert.

2.2.2 Abhängigkeitsarten

Schwinn betrachtet stets die Abhängigkeit zwischen zwei Teilzielen. Dabei können Teilziele immer nur von den Teilzielen weiter links abhängig sein. Sind zwei Teilziele unabhängig, so können sie parallel hergeleitet werden, sonst müssen sie sequentiell hergeleitet werden. Diese Abhängigkeiten können zum Teil zur Übersetzungszeit und zum Teil erst zur Laufzeit bei der Aktivierung festgestellt werden. Es gibt 5 mögliche Abhängigkeitsarten, die im Folgenden betrachtet werden. Bei jeder Abhängigkeitsart muss ein anderer Datenflusscode erzeugt werden. Dabei gilt stets:

P: linkes Teilziel
Q: rechtes Teilziel
M_P: Menge der Variablen in P
NH_P: Menge der neuen Hilfsvariablen in P
M_Q: Menge der Variablen in Q
NH_Q: Menge der neuen Hilfsvariablen in Q

NH_P und NH_Q sind dabei jeweils die Mengen der Variablen, die im Teilziel P bzw. Q vorkommen, aber in allen Literalen links davon (also inklusive dem Kopfliteral) noch nicht vorkommen.

Unbedingte Abhängigkeit

Bei der unbedingten Abhängigkeit ist in jedem Fall keine parallele Verarbeitung möglich. Ein Beispiel, wann diese Abhängigkeitsart auftritt, ist in Listing 2.2 zu sehen. Dies tritt immer dann auf, wenn im linken Teilziel eine neue Hilfsvariable eingeführt wird, die auch im rechten Teilziel verwendet wird.

```
a(X,Y) :- b(X,H), c(Y,H).
```

Listing 2.2: Hornklausel mit zwei Unterzielen und unbedingter Abhängigkeit

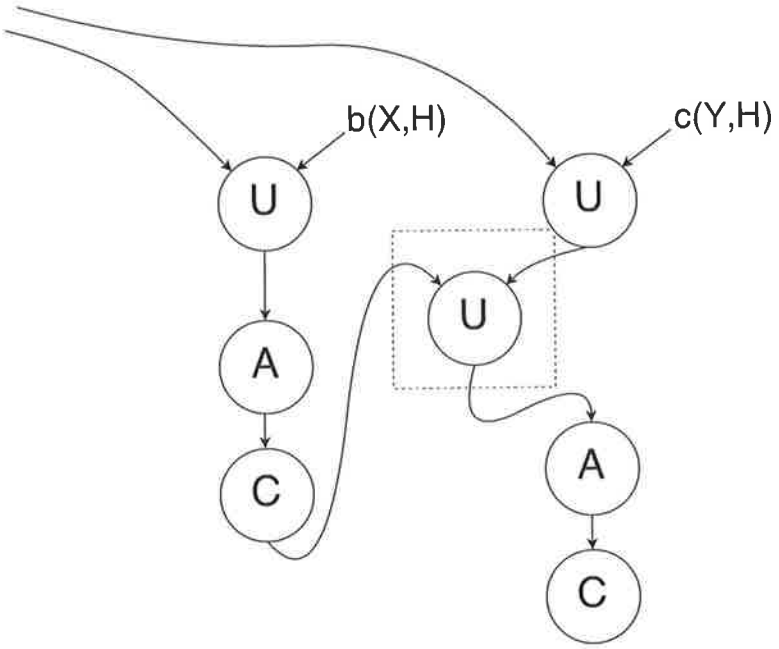

Abbildung 2.3: Datenflussgraphen bei unbedingter Abhängigkeit (Listing 2.2)

Da keine parallele Herleitung möglich ist, ergibt sich der Datenflussgraph aus Abbildung 2.3.

Zur Bestimmung, welche Abhängigkeitsart vorliegt, hat Schwinn Formeln aufgestellt, welche auch zur Entwicklung des Übersetzers verwendet werden. Auf den Beweis der Korrektheit und Geschlossenheit wird hier verzichtet.

$$Unbedingt - abhaengig(P, Q) \Leftrightarrow NH_P \cap M_Q \neq \emptyset$$

G-Unabhängigkeit

```
a(X) :- b(X), c(X).
```

Listing 2.3: Hornklausel mit zwei Unterzielen und G-Unabhängigkeit

Bei der G-Unabhängigkeit kann die Unabhängigkeit erst zur Laufzeit mit einem G-Test Knoten festgestellt werden. Listing 2.3 zeigt diesen Fall. Hierbei können b und c parallel hergeleitet werden, wenn bei der Aktivierung von a die Variable X an einen Grundterm gebunden ist. Der Datenflussgraph dazu ist Abbildung 2.4.

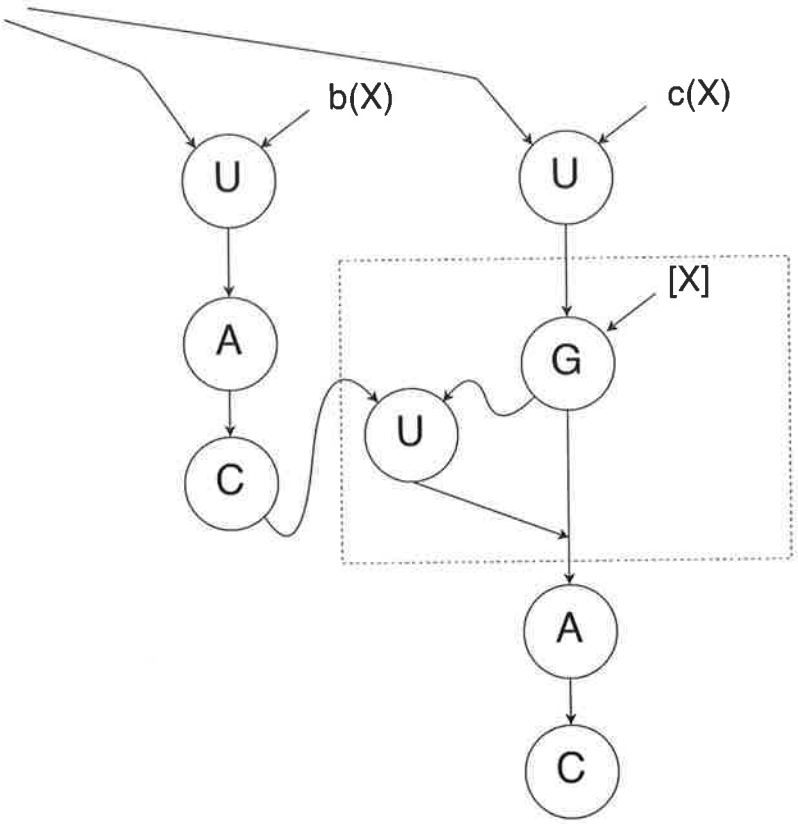

Abbildung 2.4: Datenflussgraph bei G-Unabhängigkeit (Listing 2.3)

Die formale Erkennung der G-Unabhängigkeit erfolgt mit:

$$\begin{aligned} G - unabhaengig(P,Q) \Leftrightarrow & (M_P \cap M_Q \neq \emptyset) \wedge \\ & (\forall V \in (M_P \cap M_Q) : V \notin NH_P) \wedge \\ & ((M_P \setminus (M_Q \cup NH_P) \neq \emptyset) \vee (M_Q \setminus (M_P \cup NH_Q \neq \emptyset)) \end{aligned}$$

I-Unabhängigkeit

```
a(X,Y) :- b(X), c(Y).
```

Listing 2.4: Hornklausel mit zwei Unterzielen und I-Unabhängigkeit

Liegt eine I-Unabhängigkeit vor, so sind zwei Teilziele parallel herleitbar, wenn ein Independence-Test positiv ist. Die Hornklausel aus Listing 2.4 ist I-unabhängig. b und c können parallel hergeleitet werden, wenn bei der Aktivierung zwischen X und Y keine Abhängigkeit besteht.

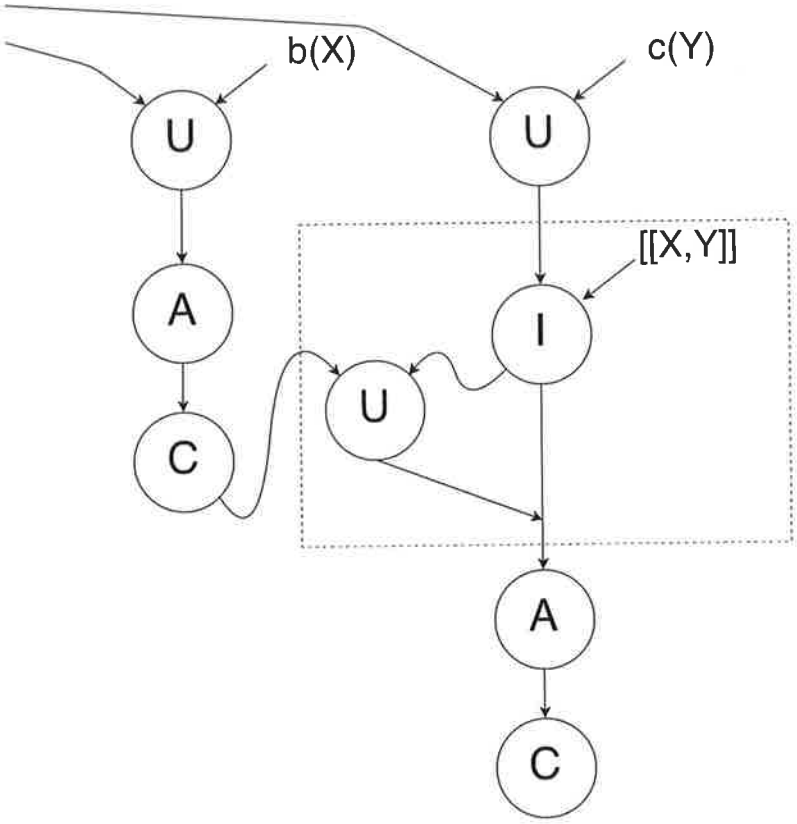

Abbildung 2.5: Datenflussgraph bei I-Unabhängigkeit (Listing 2.4)

Zur Übersetzungszeit kann die I-Unabhängigkeit durch diesen Ausdruck erkannt werden:

$$\begin{aligned} I-unabhaengig(P,Q) \Leftrightarrow & (M_P \cap M_Q = \emptyset) \wedge \\ & (M_P \setminus (M_Q \cup NH_P) \neq \emptyset) \wedge \\ & (M_Q \setminus (M_P \cup NH_Q) \neq \emptyset) \end{aligned}$$

G/I-Unabhängigkeit

```
a(X,Y,Z) :- b(X,Y), c(X,Z).
```

Listing 2.5: Hornklausel mit zwei Unterzielen und G/I-Unabhängigkeit

In Listing 2.5 ist eine G/I-unabhängige Hornklausel zu sehen. Dies ist eine Kombination aus G- und I-Unabhängigkeit. Es wird sowohl ein G-, als auch ein I-Test benötigt, um die Unabhängigkeit festzustellen (siehe Abbildung 2.6).

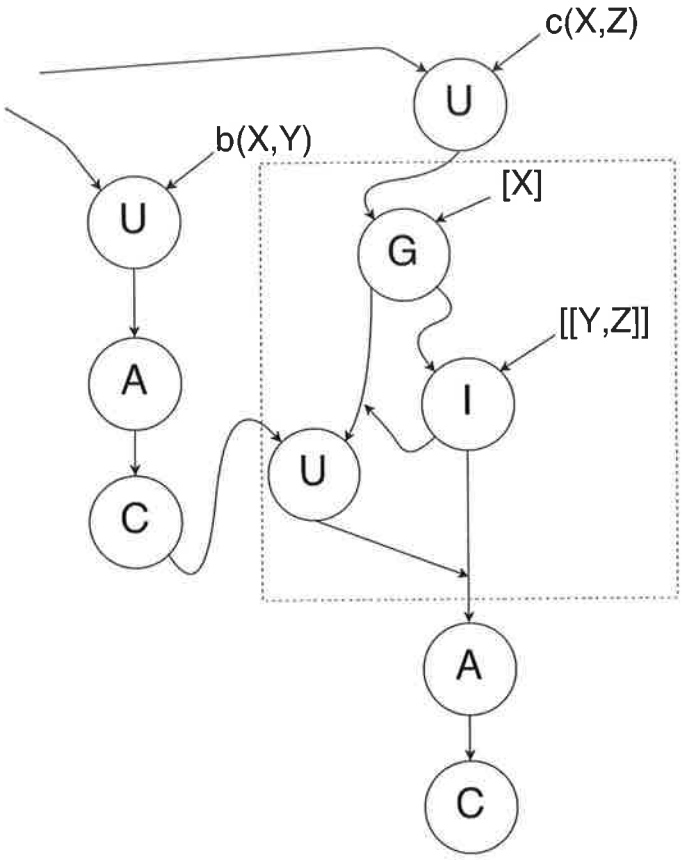

Abbildung 2.6: Datenflussgraph bei G/I-Unabhängigkeit (Listing 2.5)

$$
\begin{aligned}
G/I - unabhaengig(P,Q) \Leftrightarrow & (M_P \cap M_Q \neq \emptyset) \wedge \\
& (\forall V \in (M_P \cap M_Q) : V \notin NH_P) \wedge \\
& ((M_P \setminus (M_Q \cup NH_P) = \emptyset) \wedge (M_Q \setminus (M_P \cup NH_Q = \emptyset))
\end{aligned}
$$

Unbedingte Unabhängigkeit

```
a(X) :- b(Y), c(Z).
```

Listing 2.6: Hornklausel mit zwei Unterzielen und unbedingter Unabhängigkeit

In bestimmten Fällen kann es auch sein, dass die Unabhängigkeit bereits zur Übersetzungszeit feststeht. Ein Beispiel dafür gibt Listing 2.6. Dann können die Teilziele in jedem Fall parallel hergeleitet werden, wie in Abbildung 2.7 zu sehen ist.

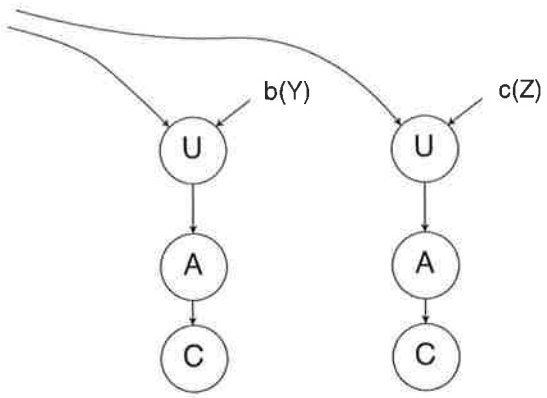

Abbildung 2.7: Datenflussgraph bei unbedingter Unabhängigkeit (Listing 2.6)

$$Unbedingt - unabhaengig(P,Q) \Leftrightarrow (M_P \cap M_Q = \emptyset) \wedge \\ ((M_P \setminus (M_Q \cup NH_P) = \emptyset) \vee (M_Q \setminus (M_P \cup NH_Q) = \emptyset)$$

2.2.3 Beispiel Datenflussgraphen

Nachdem die Knotentypen und die Abhängigkeitsarten vorgestellt wurden, betrachten wir nun einige sehr einfache Hornklauseln und deren Datenflussgraphen. Die Hornklausel aus Listing 2.7 besitzt nur ein Teilziel. Somit kann auch keine parallele Herleitung erfolgen. Abbildung 2.8 zeigt den entsprechenden Datenflussgraphen. Der Graph besitzt einen E-Knoten, bei dem konstant das Kopfliteral angelegt ist. Der rechte Ausgabeport des E-Knotens führt zu einem U-Knoten, bei dem das b-Literal anliegt. Darauf folgt die Herleitung des b-Teilziels im A-Knoten. Im Anschluss muss die eventuell erfolgte Variablenbindung noch auf das Triggertoken vom E-Knoten weitergegeben werden. Den Abschluss bildet ein R-Knoten.

```
a(X) :- b(X).
```

Listing 2.7: Hornklausel mit einem Unterziel

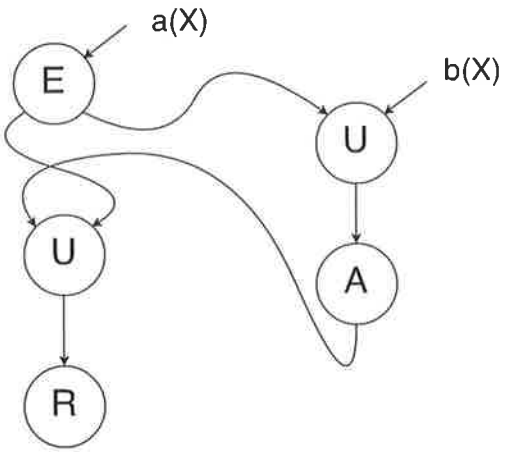

Abbildung 2.8: Datenflussgraph zu Listing 2.7

Nun betrachten wir noch einen komplexeren Fall, bei dem eine G-Abhängigkeit vorliegt. In Listing 2.8 ist eine G-abhängige Hornklausel zu sehen. Sie besitzt 2 Teilziele. Im Vergleich zum vorherigen Graphen werden hier zwei Teilziele hergeleitet, d.h. es müssen zwei U- und A-Knoten dafür erzeugt werden. Da beide das Token vom E-Knoten benötigen, wird ein C-Knoten erzeugt, der dieses Token kopiert. Es wird also sofort die Herleitung des b-Literals angestoßen. Gleichzeitig wird überprüft, ob X an einen Grundterm gebunden ist. Ist dies der Fall (Token am rechten Ausgang des G-Knotens), so kann direkt mit der Herleitung des c-Teilziels begonnen werden, ohne auf die Herleitung des b-Literals zu warten. Sonst muss aus die Herleitung des b-Teilziels gewartet und die Variablenbindung durch den U-Knoten weitergegeben werden. Am Ende müssen die Variablenbindungen der jeweiligen Herleitungen noch vereinigt werden.

```
a(X) :- b(X), c(X).
```

Listing 2.8: Hornklausel mit zwei Unterzielen

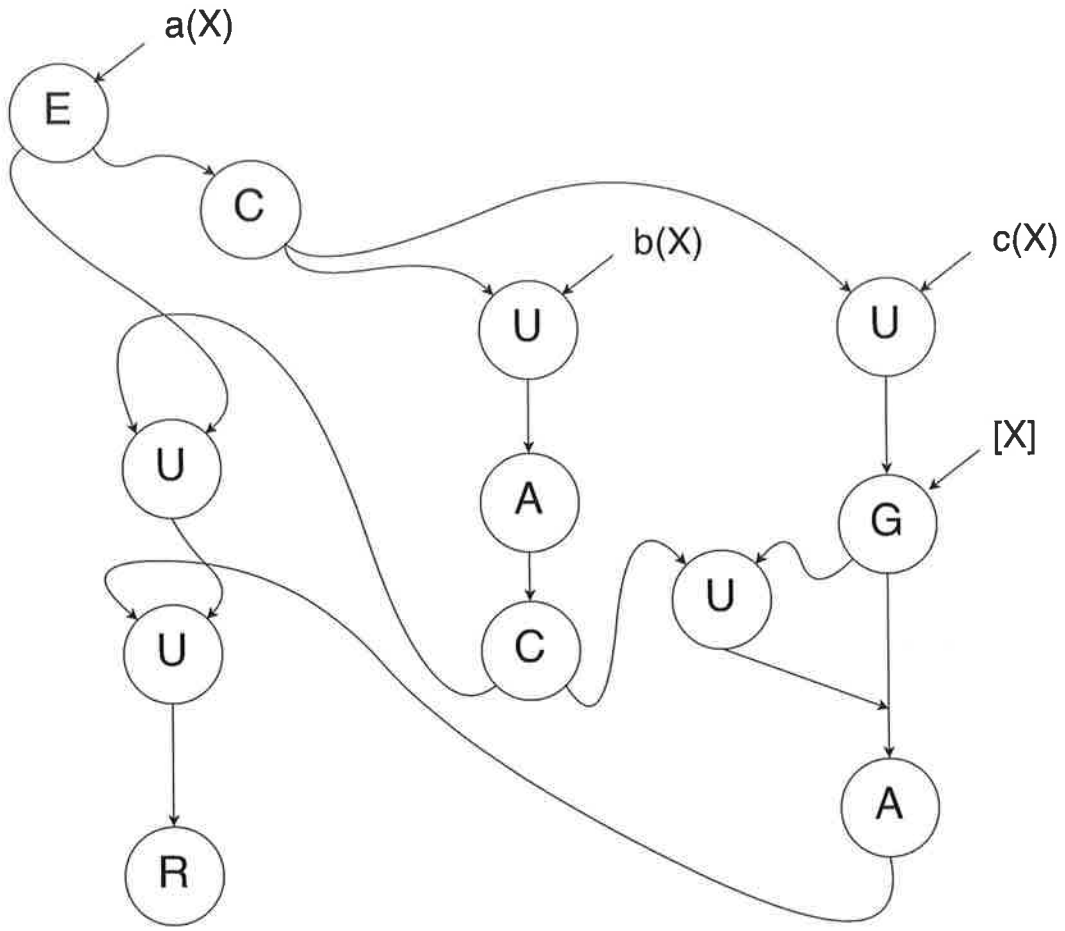

Abbildung 2.9: Datenflussgraph zu Listing 2.8

2.2.4 Repräsentation

Schwinn definiert ein tabellarisches Datenformat, in dem die Datenflussgraphen in eine Datei geschrieben und an den Interpreter weitergeben werden können. Dieses Format soll vom entwickelten Übersetzer ausgegeben werden. Die Tabelle besteht aus den Spalten:

- **Knotennummer**: Fortlaufende Nummer
- **Knotentyp**: Art des Knotens, z.B. E-Knoten
- **Rechter Nachfolger**: Verbindung des rechten Ausgangsport mit einem Knoten mit (ZielKnoten ZielPort)
- **Linker Nachfolger**: Verbindung des linken Ausgangsports mit einem Knoten mit (ZielKnoten ZielPort)
- **Zusätzliche Information**: Hier werden konstante Tokens angegeben, z.B. wird bei einem E-Knoten hier das zu unifizierende Kopfliteral angegeben.

Ein Beispiel Eintrag wäre:

```
<Nr>  <Typ>   (<RNr> <RPort>)   (<LNr> <LPort>)   Info
1       E     (1 2)             (2,1)             a(X)
```

Listing 2.9: Datenrepräsentation eines Knotens

Ein Beispiel für den Datenflussgraphen aus Abbildung 2.9 ist in Listing 2.10 zu sehen.

```
1   E (6,2)   (2,1)   a(X)
2   C (3,1)   (7,1)
3   U (4,1)   -       b(X)
4   A (5,1)   -   -
5   C (6,1)   (8,1)
6   U (12,2)  -   -
7   U (9,1)   -       c(X)
8   U (10,1)  -   -
9   G (8,2)   (10,1)  X
10  A (11,1)  -   -
11  C (12,1)
12  U (13,1)  -   -
13  R -       -   -
```

Listing 2.10: Datenrepräsentation des Graupen in Abbildung 2.3

2.3 Übersetzungsalgorithmus nach Schwinn

[13, S. 90f] liefert bereits einen Übersetzungsalgorithmus für eine einzelne Klausel, der hier in verkürzter Länge wiedergeben wird:

1. Erzeuge E-Knoten, der im rechten Eingang das konstante Kopfliteral erhält.
2. Verarbeite Teilziele von links nach rechts. Für jedes Teilziel Q.
   - 2.1. Erzeuge U-Knoten, der im rechten Eingang das Teilziel erhält.
     - 2.1.1. Wenn Q das erste Teilziel ist: Verbinde rechten Ausgang des E-Knotens mit dem linken Eingang des U-Knotens.
     - 2.1.2. Wenn Q das zweite Teilziel ist: Erzeuge einen C-Knoten, der das rechte Ausgangstoken des E-Knotens kopiert. Verbinde den C-Knoten mit den beiden U-Knoten des ersten und zweiten Teilziels.
     - 2.1.3. Sonst: Verbinde C-Knoten aus 2.2.2 mit linken Eingang des U-Knotens.
   - 2.2. Bestimme zu jedem vorangehenden Teilziel P von Q die zugehörige Abhängigkeitsklasse und erzeuge entsprechende Knoten. Verbinde die Knoten mit den zuletzt unter 2.2 erzeugten Knoten, oder falls noch nicht vorhanden. mit dem U-Knoten aus 2.1.
   - 2.3. Erzeuge A-Knoten und verbinde den Eingang mit dem Ausgang des zuletzt unter 2.2 erzeugten Knoten.
   - 2.4. Erzeuge einen C-Knoten, der das Ausgangstoken des A-Knotens kopiert.
   - 2.5. Erzeuge einen U-Knoten. Verbinde den Ausgang des unter 2.3 erzeugten C-Knotens mit dem linken Eingang des U-Knotens.
     - 2.5.1. Falls Q das erste Teilziel ist: Verbinde den linken Ausgang des E-Knotens mit dem rechten Eingang des U-Knotens von 2.5.
     - 2.5.2. Sonst: Verbinde den Ausgang des zuletzt unter 2.5 erzeugten U-Knotens (des vorangehenden Teilziels) mit dem in 2.5 rechten Eingang des neu erzeugten U-Knotens (des aktuellen Teilziels).
   - 2.6. Erzeuge einen R-Knoten, dessen Eingang mit dem Ausgang des zuletzt unter 2.5 erzeugten U-Knotens verbunden wird.