Universität Dortmund – Fachbereich Informatik
Wintersemester 2004/05
Seminar: “Plausibilitäten und Präferenzen in der Wissensrepräsentation”
Dozentin: Prof. Dr. G. Kern-Isberner
Referent: Michael Striewe

# NoMoRe:
# Non-monotonic Reasoning with Logic Programs

# Inhaltsverzeichnis



# Abbildungsverzeichnis



# Tabellenverzeichnis

# I Einleitung

Das an der Universität Potsdam entwickelte System `NoMoRe` implementiert Antwortmengenberechnung für logische Programme.[1][2]
Es basiert auf einer deklarativen Repräsentation eines Problems durch logische, regelbasierte Programme in Prolog-Syntax und berechnet für diese Graphen, in denen die Regeln die Knoten bilden. Über die Berechnung von Graphfärbungen auf diesen Graphen, die die Anwendbarkeit oder Nicht-Anwendbarkeit von Regeln kennzeichnen, gelangt es zu Antwortmengen für das gegebene Problem. Damit steht `NoMoRe` in Konkurrenz zu anderen Lösungssysteme für regelbasierte Programme, die Graphen über die atomaren Aussagen konstruieren.
Das Programm wurde für das Betriebssystem UNIX entwickelt, ist aber auch unter Windows mit Einschränkungen nutzbar. Es arbeitet als Erweiterung für Prolog-Umgebungen wie Eclipse oder SWI und ist für die wissenschaftliche Nutzung kostenlos im Internet unter der Adresse `http://www.cs.uni-potsdam.de/~linke/nomore/` verfügbar. Für die grafische Ausgabe der berechneten Graphen bietet es eine Schnittstelle zu DaVinci (`http://www.informatik.uni-bremen.de/daVinci/`) an.
Im Folgenden wird zunächst der theoretische Hintergrund der Berechnung erläutert, dann die Implementierung beschrieben und abschließend eine Evaluierung der Ergebnisse vorgenommen.

# II Theoretischer Hintergrund

## *Aufbau von Regeln*

Ein logisches Programm P besteht aus einer Regelmenge. Die verwendeten Regeln r haben die Form $p \leftarrow q_1, \ldots, q_n, not\, s_1, \ldots, not\, s_k$ mit Atomen $p, q_i (0 \leqslant i \leqslant n)$ und $s_j (0 \leqslant j \leqslant k)$. Für $n = k = 0$ ist eine Regel ein Fakt. In den Regeln gilt *not* als *negation as failure*. Die Entwickler von `NoMoRe` verzichten auf eine Unterscheidung von Atomen und Literalen und verwenden grundsätzlich nur den Begriff des Atoms. Zur einfacheren Handhabung und Benennung von Teilen der Regeln werden außerdem folgende Abkürzungen definiert:

$$head(r) = p$$
$$body^+(r) = \{q_1, \ldots, q_n\}, body^-(r) = \{s_1, \ldots, s_n\}$$
$$body(r) = body^+(r) \cup body^-(r)$$

Die Definitionen lassen sich wie üblich auf Regelmengen erweitern. Ferner bezeichnet *Facts(P)* die Menge aller Fakten in eines Programms P und *Atoms(P)* die Menge aller verwendeten Atome. Zu einer Regel $r$ erhält man die Regel $r^+$, indem man $body^-$(r) ignoriert und nur $body^+$(r) verwendet. Die Menge der $r^+$-Regeln bildet dann $P^+$.

## *Definition von Antwortmengen*

Die “Reduktion” $P^X$ eines Programmes P relativ zu einer Atommenge X ist definiert durch

$$P^X = \{r^+ \mid r \in P \wedge body^-(r) \cap X = \emptyset\}$$

und enthält damit nur genau die $r^+$-Regeln, für die $body^-$(r) nicht den in X gegebenen Annahmen widerspricht.

Eine Menge X von Atomen ist "abgeschlossen unter" einem Programm P, genau dann, wenn gilt:

$$\forall r \in P : body(r) \subseteq X \Rightarrow head(r) \in X$$

Eine Atommenge ist also genau dann abgeschlossen, wenn das Anwenden einer Regel keine Aussage erzeugt, die nicht schon bekannt ist. Die kleinste abgeschlossene Atommenge unter einem Programm P wird als Cn(P) bezeichnet.
Aus diesen beiden Definitionen ergibt sich die Definition von Antwortmengen, denn einen Atommenge X ist Antwortmenge für ein Programm P genau dann, wenn $Cn(P^X)=X$. X ist also dann eine Antwortmenge, wenn sich nach dem Weglassen aller nicht anwendbaren Regeln (Reduktion) und der anschließenden Anwendung aller anwendbaren Regeln keine zusätzlichen atomaren Aussagen treffen lassen (Abgeschlossenheit) und sich gleichzeitig keine Aussagen als unnötig erweisen (Minimalität). Dies entspricht der intuitiven Vorstellung davon, wie sinnvolle Aussagen auf der Basis einer Regelmenge getroffen werden. Antwortmengen sind daher geeignet, logisches Schließen als berechenbaren Vorgang abzubilden.
Die Menge der "erzeugenden Regeln" (generating rules) für ein Programm P bzgl. einer Atommenge X ist definiert als

$$GR(P, X) = \{ r \in P \mid body^+(r) \subseteq X, body^-(r) \cap X = \emptyset \}$$

Damit ist X ebenfalls dann eine Antwortmenge, wenn X = $Cn(GR(P, X)^+)$ gilt. Im allgemeinen Fall ist dabei $GR(P, X)^+$ nicht gleich $P^X$, wie an P={a ←, b ← c} und X = {a} leicht nachzurechnen ist.
Beiden Definitionen von Antwortmengen ist gemein, dass sie als Eingabe neben dem Programm P auch eine Menge X erwarten und sich aus ihnen somit keine direkte Berechnungsvorschift für die Erzeugung von Antwortmengen gewinnen lässt.

## *Aufbau des "block graph"*

Der von `NoMoRe` implementierte graphische Ansatz benötigt eine sortierte Regelmenge und bezeichnet daher eine Regelmenge P als "begründet", genau dann, wenn

$\langle r_i \rangle_{i \in I}$ eine Aufzählung ist, so dass
$\forall i \in I : body^+(r_i) \subseteq head(\{r_1, \ldots, r_{i-1}\})$

gilt. Für jedes Programm P gibt es eine bzgl. ihrer Größe eindeutige maximale begründete Untermenge P'. Mit Hilfe dieser Untermenge kann ein sogenannter "block graph" $\Gamma_P$ mit folgenden Merkmalen definiert werden:

$$\Gamma_P = (V_P, A_P^0 \cup A_P^1)$$
$$V_P = P'$$
$$A_P^0 = \{(r', r) \mid r', r \in P' \wedge head(r') \in body^+(r)\}$$
$$A_P^1 = \{(r', r) \mid r', r \in P' \wedge head(r') \in body^-(r)\}$$

Der Graph enhält die Regeln als Knoten und enthält in seinen Kanten Informationen über die Beziehungen zwischen Knoten. Eine 0-Kante verläuft von einer Regel r' zu einer Regel r, wenn beide Regeln in P' liegen und in r' auf eine atomare Aussage geschlossen wird, die in $body^+(r)$ benötigt wird. 0-Kanten kodieren damit die Begründetheit von Teilmengen in

den Graphen.
Eine 1-Kante verläuft von einer Regel r' zu einer Regel r, wenn in r' auf eine atomare Aussage geschlossen wird, die in body⁻(r) vorkommt und damit die Anwendung dieser Regel blockiert. 1-Kanten kodieren damit Blockade von Regeln in den Graphen.

Abbildung1 zeigt den block graph für ein einfaches Anwendungsbeispiel in der grafischen Ausgabe des DaVinci-Tools. Die zugehörige Regelmenge in Prolog-Syntax lautet:

```
penguin.
bird :- penguin.
wings :- bird.
fly :- bird, not nofly.
nofly :- penguin, not fly.
```

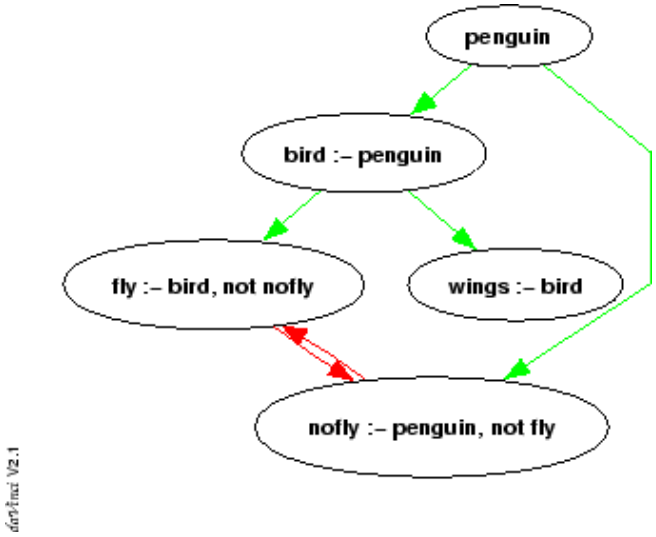


*Abbildung 1: Block graph in der Ausgabe des DaVinci-Tools.*
*0-Kanten sind grün markiert, 1-Kanten rot. Die Regeln sind in den Knoten in Prolog-Syntax angegeben.*

Gültige Antwortmengen für dieses Programm sind $X_1$ = {penguin, bird, wings, fly} und $X_2$ = {penguin, bird, wings, nofly}.

## *Knotenfärbung auf dem block graph*

`NoMoRe` versucht nun, auf dem erzeugten Graphen eine Knotenfärbung $C: P \rightarrow \{\ominus, \oplus\}$ zu finden, die angibt, welche Regeln anwendbar sind und welche nicht. Dabei gilt eine Regel r in einem partiell oder vollständig gefärbten Graphen als "blockiert", wenn sie einen 1-Vorgänger hat, für den gilt $C(r') = \oplus$. Eine Regel r gilt als "begründet", wenn die folgenden vier Bedingungen erfüllt sind:

1. es existiert ein azyklischer Subgraph $G_r = (V_r, A_r)$ auf $\Gamma_P$ mit $A_r \subseteq A_P^0$
2. $r \notin V_r$ und $body^+(r) \subseteq head(V_r)$
3. für alle $r' \in V_r$ und $q' \in body^+(r')$ existiert ein $r'' \in V_r$ mit $q' = head(r'') \wedge (r'', r') \in A_r$
4. $C(V_r) = \oplus$

Der Subgraph $G_r$ gibt damit eine Art Beweisfolge an, wie man über die Anwendung vorhergehender Regeln die Regel r begründen kann.
Bei einer vollständigen Graphfärbung C(r) handelt es sich um ein sogenanntes *a-coloring*, wenn für alle Regel r in P die folgende Bedingung erfüllt ist:

$$C(r) = \oplus \Leftrightarrow \text{r ist begründet und nicht blockiert bzgl.} \Gamma_P \text{ und } C$$

Diese Definition kann genutzt werden, um eine Antwortmenge zu berechnen, denn es gilt $GR(P, X) = \{r \in P \mid C(r) = \oplus\}$ [3], wodurch die explizite Angabe der Atommenge X durch eine implizite Angabe durch die Einfärbung anwendbarer Regeln ersetzt wird. Nach der Definition von erzeugenden Regeln werden dort diejenigen Regeln r nicht berücksichtigt, für die andere Regeln auf ein Atom aus $body^-(r)$ schliessen. Dies entspricht der Blockade von Regeln im Graphen. Die Begründetheit von Regeln im Graphen entspricht der Berücksichtigung von Regeln für die Menge der erzeugenden Regeln, denn es werden in beiden Fällen nur Regeln r berücksichtigt, für die alle Atome aus $body^+(r)$ durch andere Regeln erschlossen werden.
Auf der Basis dieses Zusammenhangs lässt sich ein Algorithmus entwickeln, der ein Programm P als Eingabe erwartet und über die Erzeugung eines Graphen und die Berechnung eines a-colorings zur Ausgabe einer Antwortmenge kommt. `NoMoRe` implementiert diesen Algorithmus.

## III Implementierung

`NoMoRe` erwartet die Eingabe von Regeln in Prolog-Notation und bereits in begründeter Reihenfolge. Das Einlesen erfolgt über einen Parser, dem bei Bedarf ein herkömmlicher Grounder vorgeschaltet werden kann. Nach dem Parsen berechnet das System zunächst den Graphen und kompiliert daraus Prolog Code zur Berechnung der Graphfärbung und zu deren Interpretation als Antwortmengen.

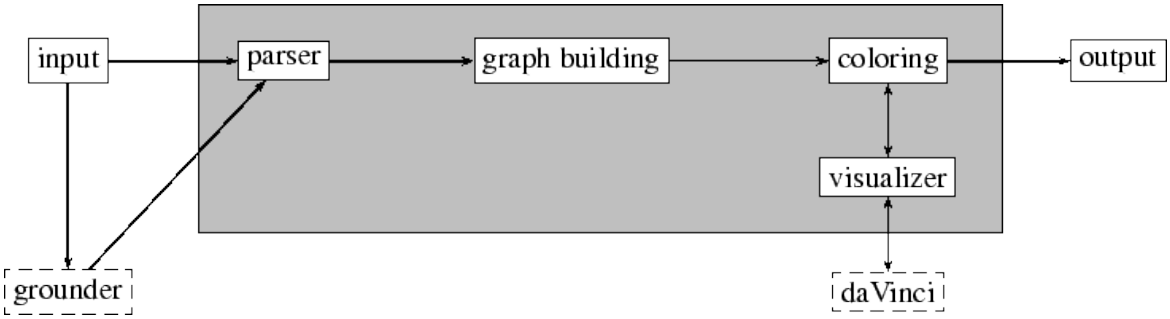


*Abbildung 2: Architektur von NoMoRe*

### *Propagation*

Die vollständige Färbung des Graphen wird durch die schrittweise Erweiterung einer partiellen Graphfärbung über eine rekursive Funktion erzeugt. Die wichtigste Vorgehensweise stellt dabei die sogenannte “Propagation” dar, bei der ausgehende von einem gefärbten Knoten bestimmte benachbarten Knoten ebenfalls eingefärbt werden. Das

Problem der Berechnung einer vollständigen Graphfärbung wird somit auf die Berechnung lokaler Erweiterungen für partielle Graphfärbungen reduziert.
Die Definitionen von Begründetheit und Blockiertheit müssen für partielle Graphfärbungen entsprechend angepasst werden. Es gilt:

$$p-begründet(r) \Leftrightarrow \forall q \in body^+(r): \exists r' \in Pred0(r): q = head(r') \wedge C(r') = \oplus$$
$$p-unbegründet(r) \Leftrightarrow \exists q \in body^+(r): \forall r' \in Pred0(r): q \neq head(r') \vee C(r') = \ominus$$
$$p-blockiert(r) \Leftrightarrow \exists r' \in Pred1(r): C(r') = \oplus$$
$$p-unblockiert(r) \Leftrightarrow \forall r' \in Pred1(r): C(r') = \ominus$$

Propagation kann in den folgenden Fällen eingesetzt werden:
(A) Ist eine Regel r mit $\oplus$ markiert und r' ein 1-Nachfolger, dann wird r' mit $\ominus$ markiert.
(B) Ist eine Regel r mit $\ominus$ markiert und r' ein 0-Nachfolger und es gilt nicht p-begründet (r'), dann wird r' mit $\ominus$ markiert.
(C) Ist eine Regel r mit $\oplus$ markiert und r' ein 0-Nachfolger und es gilt p-begründet(r') sowie p-unblockiert(r'), dann wird r' mit $\oplus$ markiert.
(D) Ist eine Regel r mit $\ominus$ markiert und r' ein 1-Nachfolger und es gilt p-begründet(r') sowie p-unblockiert(r'), dann wird r' mit $\oplus$ markiert.

Falls mehrere ungefärbte Knoten im nächsten Schritt eingefärbt werden könnten, wird die Entscheidung, mit welchem Knoten fortgesetzt wird, nicht-deterministisch getroffen. Über Backtracking werden alternative Entscheidungen getroffen, wenn die erste Wahl zu keiner Lösung führt. Die rekursive Basisfunktion von `NoMoRe` sieht in Pseudocode folgendermaßen aus:

```
function color_P(U, N : list; C : partial mapping)
 var r : node
  if propagate_P(N, C) then
    U := U \ (C_⊖ ∪ C_⊕)
    if choose_P(U, C, r) then
      U := U \ {r}
      if color_P(U, {r}, (C_⊖, C_⊕ ∪ {r})) then
        return true
      else
        return color_P(U, {r}, (C_⊖ ∪ {r}, C_⊕))
    else
      return propagate_P(U, (C_⊖ ∪ U, C_⊕))
  else
    return false
```

Dabei implementiert **choose**$_P$ die nicht-deterministisch Auswahl eines Knoten und **propagate**$_p$ die deterministischen Folgen seiner Einfärbung.
Sei $L_1(P) = \{r \in P | r \in Pred1(r)\}$ die Menge aller 1-Schleifen in einem Programm P.
Wird **color**$_P$ auf einem Programm zum ersten Mal aufgerufen, so beginnt es mit $C = (L_1(P), Facts(P)), U = P \setminus (Facts(P) \cup L_1(P))$ und $N = Facts(P) \cup L_1(P)$. Zu Beginn sind also alle Fakten mit $\oplus$ und alle 1-Schleifen mit $\ominus$ markiert. Die Funktion **color**$_P$ wählt nun einen ungefärbten Knoten aus $U$ aus und versucht ihn mit $\oplus$ einzufärben. Führt dies zu keiner Lösung, versucht **color**$_P$ diesen Knoten mit $\ominus$ zu markieren. Schlägt auch dies fehl, gibt **color**$_P$ false zurück und es gibt keine Lösung. Die Propagation von

Färbungen geschieht dabei unmittelbar nach der Einfärbung eines Knotens und führt dann ggf. auch sofort zu einem Fehler.
Die Erweiterung einer partiellen Färbung nur über Propagation in Richtung der Graphkanten ist nicht effektiv, da es Fälle gibt, in denen Wissen über nötige Knotenfärbungen auch entgegen der Richtung der Graphkanten weitergegeben werden könnte.

## *Backward Propagation*

Wie bei der Propagation in Richtung der Graphkanten gibt es auch bei der "Backward Propagation" theoretisch vier Fälle, in denen entgegen der Richtung der Kanten eine Färbung weitergegeben werden könnte. Allerdings können für diese Richtung keine einfachen lokalen Bedingungen mehr erstellt werden. Abbildung 3 zeigt einen Ausschnitt aus einem block graph mit einer partiellen Graphfärbung. Einerseits ist klar, dass r' mit ⊕ markiert werden muss, damit r blockiert ist. Andererseits kann r' nicht mit ⊕ markiert werden, denn es gilt nicht p-begründet(r').

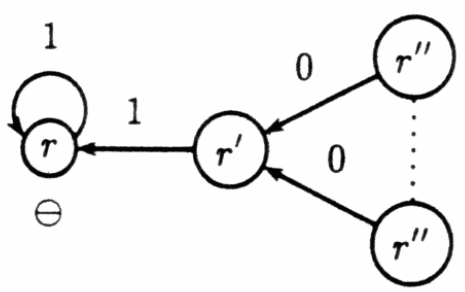

*Abbildung 3: Ausschnitt aus einem block graph mit partieller Färbung*

Zur Lösung dieses Problems wird eine dritte Farbe + eingeführt, um Knoten r' zu markieren, auch wenn p-begründet(r') noch nicht gilt. Im späteren Verlauf der Berechnung der Färbung über Propagation wird diese dann durch ⊕ ersetzt, falls dies möglich ist. Muss ein mit + gefärbter Knoten später mit ⊖ eingefärbt werden, so schlägt die Färbung fehl und **color**$_P$ gibt false zurück.
Daraus ergeben sich vier weitere lokale Färbungsregeln für Backward Propagation:

(E) Ist eine Regel r' mit ⊕ markiert und r ein 1-Vorgänger von r', dann wird r mit ⊖ markiert.
(F) Ist eine Regel r' mit ⊖ markiert und es gilt p-begründet(r') und r ist der einzige 1-Vorgänger von r', der nicht mit ⊖ markiert ist, dann wird r mit + markiert.
(G) Ist eine Regel r' mit ⊕ markiert und r ist der einzige begründbare und nicht mit ⊖ markierte 0-Vorgänger für ein Atom aus body$^+$(r), dann wird r mit + markiert.
(H) Ist eine Regel r' mit ⊖ markiert und es gilt p-unblockiert(r') und r ist der einzige unbegründete und nicht mit ⊕ markierte 0-Vorgänger für ein Atom aus body$^+$(r'), dann wird r mit ⊖ markiert.

Der Vorteil der Anwendung von Backward Propagation ist eine Reduktion der notwendigen Anzahl von nicht-deterministischen Entscheidungen.

Abbildung 4 zeigt ein Beispiel für die Anwendung von Backward Propagation. Zu Beginn kann nichts propagiert werden, denn es gibt weder Fakten noch 1-Schleifen. O.b.d.A. wähle der Algorithmus nun Knoten $r_a$. Zunächst wird dieser mit ⊕ markiert, was zur Propagierung von ⊖ für $r_b$ und $r_d$ führt und dies wiederum zur Propagierung von ⊕ für $r_c$. Damit ist eine erste vollständige Färbung gefunden worden, ohne dass Backward

Propagation benötigt wurde.
Nun wird $r_a$ mit $\ominus$ markiert. In dieser Situation ist noch keine Regel anwendbar, so dass eine weitere nicht-deterministische Entscheidung getroffen werden muss. Angenommen, der Algorithmus wählt nun $r_b$ und markiert den Knoten mit $\oplus$, so ergibt sich wiederum nur durch Propagation eine weitere gültige Graphfärbung. Wird $r_b$ dagegen ebenfalls mit $\ominus$ markiert, so ist mit den Regeln (A)-(D) keine Propagation möglich und ohne Backward Propagation müsste eine weitere nicht-deterministische Entscheidung getroffen werden. Regel (F) ermöglicht es dagegen in diesem Fall, $r_c$ mit + zu markieren. Dann bleibt nur noch $r_d$ unmarkiert. Beide Färbeversuche schlagen dort wegen $r_a$ bzw. $r_c$ sofort fehl und der Versuch kann sofort abgebrochen werden. In diesem Beispiel wurde nur einmal eine Regel des Backward Propagation angewandt, was aber sofort zu einer Vermeidung unnötiger Versuche geführt hat, da im anderen Fall zunächst ein Knoten hätte gewählt werden müssen und dann möglicherweise ein erfolreicher Propagation-Schritt möglich gewesen wäre, bevor ein Widerspruch auftritt.

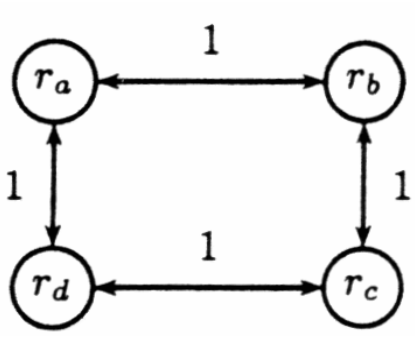


*Abbildung 4: Beispielgraph für die Anwendung von Backward Propagation*

## *Jumping*

Eine weitere Verbesserung des Färbealgorithmus stellte das sogenannte Jumping dar. Bei der Backward propagation enthalten die Regeln (F)-(H) sehr viele Bedingungen, von denen häufig einige nicht erfüllt sind. Das Eintreten dieser Bedingungen kann aber eindeutig von anderen Entscheidungen abhängig sein. Abbildung 5 zeigt eine solche Situation. Es ist eindeutig, dass $r_a$ mit $\ominus$ markiert werden muss, um überhaupt zu einer vollständigen Graphfärbung zu gelangen. Demnach muss $r_a$ blockiert werden. Dazu reicht die Markierung von einem der beiden Knoten $r_b$ und $r_d$ mit $\oplus$ aus. Wenn später $r_b$ gewählt und mit $\ominus$ markiert wird, dann wird genau dies durch Jumping erreicht. Regel (F) wird dann erneut für $r_a$ angewandt und damit die Färbung von $r_d$ erreicht. Dies wiederum führt unmittelbar zur Färbung von $r_e$ nach Regel (A). Das Jumping hat in diesem Fall die sonst später nötige, nicht-determninistische Wahl und Färbung von $r_d$ oder $r_e$ überflüssig gemacht.

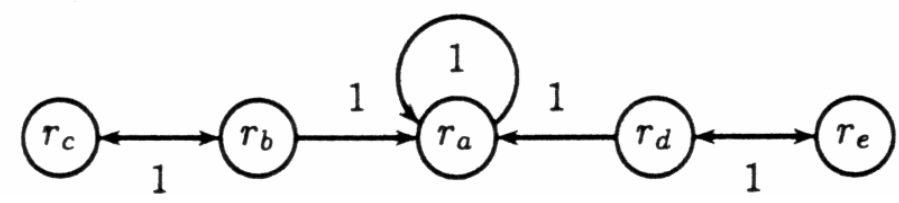


*Abbildung 5: Beispielgraph für die Anwendung von Jumping*

Damit ergeben sich weitere Regeln für die Propagation:
(I) Ist eine Regel r' mit $\ominus$ markiert und wird ein 1-Vorgänger r ebenfalls mit $\ominus$

markiert, so wird auf r' erneut Regel (F) angewandt.

(J) Ist eine Regel r' mit $\oplus$ markiert und wird ein 0-Vorgänger r mit $\ominus$ markiert, so wird auf r' erneut Regel (G) angewandt.

(K) Ist eine Regel r' mit $\ominus$ markiert und wird ein 0-Vorgänger r mit $\oplus$ markiert, so wird auf r' erneut Regel (H) angewandt.

Die vollständige Propagation-Funktion von `NoMoRe` sieht damit in Pseudocode folgendermaßen aus:

```
procedure propagate_P(N : list, C : part. mapping)
 var r' : node;
   while N ≠ ∅ do
     select r' from N;
     if (r' ∈ C_⊕) then
       if propA_P(r', C) fails then fail;
       if propC_P(r', C) fails then fail;
       if backpropA_P(r', C) fails then fail;
       if backpropC_P(r', C) fails then fail;
       if jumpC_P(r', C) fails then fail;
     else
       if propB_P(r', C) fails then fail;
       if propD_P(r', C) fails then fail;
       if backpropB_P(r', C) fails then fail;
       if backpropD_P(r', C) fails then fail;
       if jumpB_P(r', C) fails then fail;
       if jumpD_P(r', C) fails then fail;
```

Die Prozeduren **propA**$_P$, **propB**$_P$, **propC**$_P$ und **propD**$_P$ implementieren die Regeln (A)-(D), die Prozeduren **backpropA**$_P$, **backpropB**$_P$, **backpropC**$_P$ und **backpropD**$_P$ implementieren die Regeln (E)-(H) und die Prozeduren **jumpB**$_P$, **jumpC**$_P$ und **jumpD**$_P$ implementieren die Regeln (I)-(K).

## *Korrektheit und Vollständigkeit*

Um das in der Theorie gewonnene Ergebnis, dass $GR(P, X) = \{r \in P \mid C(r) = \oplus\}$ gilt, anwenden zu können, muss sichergestellt sein, dass **color**$_P$ genau alle a-colorings für einen gegebenen block graph findet.

Aus der bisherigen Beschreibung der Propagation-Möglichkeiten geht hervor, dass diese keine Knoten-Markierungen entfernen, sondern lediglich deterministisch neue hinzufügen. Ferner prüft die Funktion **propagate**$_P$ alle ermittelten Bedingungen ab. Die gefundenen Lösungen sind daher korrekt.

Aus der Determiniertheit von **propagate**$_P$ ergibt sich, dass lediglich durch die nicht-deterministische Auswahl des nächsten zu bearbeitenden Knoten **choose**$_P$ in **color**$_P$ neue Färbungen induziert werden. In **color**$_P$ werden für jeden Knoten beide Markierungesmöglichkeiten getestet, so dass sichergestellt ist, dass alle Färbungen gefunden werden. Die Menge der gefundenen Lösungen ist daher vollständig.

# IV Evaluation

## *Benchmarks*

Das System NoMoRe wurde von den Entwicklern mit verschiedenen Benchmark-Funktionen getestet. Zum Einsatz kamen mehrere NP-vollständige Probleme, Planungsprobleme sowie das n-Damen-Problem. Es wurde sowohl die Laufzeit als auch die Zahl der nicht-deterministischen Entscheidungen gemessen und mit den Ergebnissen anderer Lösungssysteme für Antwortmengen verglichen. Dabei gibt die Zahl der getroffenen nicht-deterministischen Entscheidungen einen guten Einblick in die theoretische Leistungsfähigkeit der Implementierung, sofern man davon ausgeht, dass der deterministische Teil implementierungsunabhängig eine polynomielle Laufzeit benötigt. Für die Analyse der praktischen Anwendbarkeit ist dagegen eher die absolute Laufzeit in Interesse, die nicht in jedem Fall mit der Zahl der Entscheidungen abnimmt.

| | noMoRe | | | smodels | |
|---|---|---|---|---|---|
| backprop | no | yes | yes | with | (without) |
| jumping | no | no | yes | lookahead | |
| ham_k_7 | 14335 | 14335 | 2945 | 4814 | (34077) |
| ham_k_8 | 82200 | 82200 | 24240 | 688595 | (86364) |
| ind_cir_20 | 539 | 317 | 276* | 276 | (276) |
| ind_cir_30 | 9266 | 5264 | 4609* | 4609 | (4609) |
| p1_step4 | - | 464 | 176 | 7 | (69) |
| p2_step6 | - | 13654 | 3779 | 75 | (3700) |
| col4x4 | 27680 | 27680 | 7811 | 7811 | (102226) |
| col5x5 | - | - | 580985 | 580985 | (2.3 Mil) |
| queens4 | 84 | 84 | 5 | 1 | (11) |
| queens5 | 326 | 326 | 13 | 9 | (34) |

*Tabelle 1: Benchmarks für NoMoRe*
*Gemessen wurden die Zahl der nicht-deterministischen Entscheidungen bis zum Auffinden aller Lösungen.*

| | noMoRe | | | smodels | | dlv |
|---|---|---|---|---|---|---|
| backprop | no | yes | yes | with | (without) | |
| jumping | no | no | yes | lookahead | | |
| ham_k_7 | 17.6 | 3.25 | 3.92 | 0.38 | (0.16) | 0.1 |
| ham_k_8 | 328.86 | 35.86 | 44.23 | 7.69 | (3.57) | 0.71 |
| ind_cir_40 | 5.84 | 7.18 | 7.24 | 1.31 | (0.77) | 1.97 |
| ind_cir_50 | 106.15 | 127.45 | 128.99 | 23.87 | (16.81) | 40.62 |
| col4x4 | 1.92 | 2.51 | 1.96 | 0.1 | (0.27) | 0.36 |
| col5x5 | 179.38 | 223.42 | 180.35 | 11.54 | (41.17) | 24.79 |
| queens7 | 0.73 | 0.84 | 0.5 | 0.08 | (0.03) | 0.05 |
| queens8 | 3.82 | 4.3 | 1.86 | 0.32 | (0.09) | 0.12 |

*Tabelle 2: Zeit-Benchmarks für NoMoRe*
*Gemessen wurde die Laufzeit in Sekunden bis zum Auffinden aller Lösungen auf einem AMD 1.4 Ghz-Prozessor*

Der positive Einfluss von Backward propagation und Jumping auf die Zahl der Entscheidungen ist eindeutig zu erkennen. In den mit (*) markierten Fällen erreicht NoMoRe die minimale Anzahl an nötigen nicht-deterministischen Entscheidungen. Die Verwendung von Jumping führt dabei allerdings nur in einem Teil der Fälle zu einer Verbesserung der Laufzeit, wogegen es die Zahl der nicht-deterministischen Entscheidungen in jedem Fall deutlich reduziert.
Seit der Veröffentlichung der angegebenen Benchmarks wurde NoMoRe weiter verbessert und erreicht im Fall des 4-Damen-Problems und des 5-Damen-Problems nun auch die minimale Anzahl an Entscheidungen. Im Vergleich zu smodels [5] weisen die Entwickler von NoMoRe ausdrücklich darauf hin, dass ihr System zum Zeitpunkt der Veröffentlichung noch nicht über einen Lookahead verfügte. Ferner führen sie einige Leistungsunterschiede auf unterschiedliche Heuristiken beim Treffen der nicht-deterministischen Entscheidungen zurück und kündigen in [2] weitere Verbesserungen auf diesem Gebiet an.

### *Grenzen regelbasierter Systeme*

Die positiven Effekte der Erweiterung der Implementierung dürfen allerdings nicht darüber hinweg täuschen, dass auch dem Einsatz regelbasierter Systeme grundsätzliche Grenzen gesetzt sind, die sich am Beispiel des n-Damen-Problems gut beobachten lassen. Als Ergänzung zu Tabelle 1 sind dazu die folgenden Werte von Interesse:

4-Queens: 208 Regeln, 2 Lösungen, 1 Entscheidung
5-Queens: 405 Regeln, 10 Lösungen, 9 Entscheidungen
6-Queens: 700 Regeln, 4 Lösungen, 7 Entscheidungen

Der optimierten Anzahl von nötigen nicht-deterministischen Entscheidungen steht eine unvermeidbar hohe Anzahl von Regeln gegenüber, die in eine begründete Reihenfolge gebracht werden müssen und die die Knoten des Graphen darstellen. `NoMoRe` führt Backtracking auf diesem Graphen aus. Da sich die Zahl der nötigen Regeln zur Verhinderung ungültiger Lösungen von der Größenordnung her nicht unter die Zahl der in einer direkten Backtracking-Suche über alle möglichen Positionen auf dem Schachbrett nötigen Vergleiche drücken lässt, kann die Komplexität der Berechnung nicht unter die des klassischen direkten Backtracking-Verfahrens reduziert werden.
Das Auffinden einer einzelnen Lösung für das n-Damen-Problem ist allerdings ohnehin ohne Backtracking in linearer Laufzeit möglich.

## V Zusammenfassung

Für bestimmte Probleme stellt die Berechnung von Antwortmengen für logische Programme eine geeignete Methode dar, um das Ziehen sinnvoller Schlüsse als berechenbare Aufgabe durchzuführen.
Das System `NoMoRe` ergänzt die grundsätzliche Umsetzung einer graphbasierten Berechnung von Antwortmengen um sinnvolle Erweiterungen, die zu erhöhter Berechnungeffizienz und zu einer Verringerung von nicht-deterministischen Entscheidungen führen. `NoMoRe` erreicht damit im Bezug auf die Zahl der Entscheidungen die Qualität ähnlicher Systeme zur Berechnung von Antwortmengen für logische Programme (insbesondere `smodels`). Gleichwohl kann über die Vorzüge von regelbasierter Berechnung gegenüber atombasierter Berechnung noch keine Aussage getroffen werden. Der in der Backward propagation eingeführte Einsatz einer dritten Knotenmarkierung ähnelt dabei dem von `dlv` eingesetzten dritten Wert für Atome [4], wobei auch hier die Unterschiede zur atombasierten Berechnung noch nicht ausreichend analysiert sind.
Die gefundenen Antwortmengen werden grundsätzlich ohne weitere Bearbeitung als mögliche Lösungen ausgegeben. Eine von der graphischen Berechnung unabhängige Gewichtung von Regeln oder atomaren Aussagen, die eine Bewertung verschiedener Lösungen im Bezug auf ihre Qualität oder Plausibilität ermöglicht, ist allerdings ohne Umstände denkbar.

## VI Literatur


[1] C. Anger, K. Konczak, und T. Linke, Nomore: Non-monotonic reasoning with logic programs. In *Proceedings of the 8th European Conference on Logics in AI, JELIA 2002*, Seite 521-524, Cosenza, Italien, 2002. Springer, LNCS/2424

[2] C. Anger, K. Konczak, und T. Linke, More on Nomore. In *Proceedings of the 8th European Conference on Logics in AI, JELIA 2002*, Seite 468-480, Cosenza, Italien, 2002. Springer, LNCS/2424

[3] T. Linke: Graph theoretical characterization and computation of answer sets. In B. Nebel, editor, *Proc. of the IJCAI*, Seite 641-645. Morgan Kaufmann Publishers, 2001.

[4] W. Faber, N. Leone, und G. Pfeifer. Pushing goal derivation in dlp computations. In M. Gelfond, N. Leone, und G. Pfeifer, editors, *LPNMR*, vol. 1730 of LNAI, Seite 177-191, El Paso, Texas, USA, 1999, Springer Verlag.

[5] I. Niemelä und P. Simons. Smodels: An implementation of the stable model and well-founded semantics for normal logic programs. In J. Dix, U. Furbach, und A. Nerode, editors, *Proc. of the LPNMR*, Seite 420-429. Springer, 1997.