Wissensrepräsentation

Nebel/Rintanen/Wölfl
WS 2004/2005

Universität Freiburg
Institut für Informatik

Projekt P2

Abgabe: Freitag, 21. Januar 2005

Abgabe per E-Mail an `helmert@informatik.uni-freiburg.de`

Dieses Projekt widmet sich dem qualitativen zeitlichen Schließen in Allens Intervallalgebra. Es umfasst drei Teile:

- Im ersten Teil wird ein einfacher Backtracking-Algorithmus zum Lösen von Intervall-CSPs implementiert.
- Im zweiten Teil wird die Erweiterung dieses Algorithmus auf effizienteres Branching mit Hilfe der ORD-Horn-Klasse vorbereitet.
- Im dritten Teil wird diese Erweiterung implementiert.

Hinweis: Bei den Projektaufgaben geht es um die Implementierung, nicht so sehr um die theoretischen Aspekte. Wenn Sie an bestimmten Stellen nicht genau wissen, wie die Algorithmen und Definitionen aus der Vorlesung in die Praxis umgesetzt werden, können Sie daher gerne Fragen stellen — entweder per E-Mail oder in der Übungsgruppe.

Aufgabe P2.1 (Lösung von Intervall-CSPs, 2 Punkte)

Implementieren Sie ein Programm zum Lösen von CSPs in Allens Intervallalgebra. Eine Lösung zu einem Eingabe-CSP Θ ist ein konsistentes *Basis-CSP*, das heißt ein CSP, das für jedes Paar von Variablen aus Θ eine Basisrelation festlegt und mit Θ konsistent ist.
Verwenden Sie dabei folgenden Backtracking-Algorithmus:

1. Zunächst wird Pfadkonsistenz hergestellt.
2. Ergibt sich ein leerer Constraint, wird dieser Suchzweig abgebrochen.
3. Ergibt sich ein Basis-CSP, wird die Lösung ausgegeben und die Suche beendet.
4. Tritt keiner dieser Fälle ein, werden Variablen X und Y ausgewählt, so dass die Relation zwischen X und Y noch keine Basisrelation ist. Es wird immer ein Variablenpaar gewählt, für das der aktuelle Constraint zwischen X und Y *minimale Kardinalität* hat, und unter solchen Paaren immer das *lexikographisch kleinste*. Es werden nur Paare berücksichtigt, bei denen X im Alphabet vor Y steht, da für den umgekehrten Fall immer die konverse Relation gilt.
5. Nun wird nacheinander der Constraint zwischen X und Y mit den verschiedenen möglichen Basisrelationen instantiiert, und zwar in der Reihenfolge $\{\equiv, \prec, \succ, \mathsf{d}, \mathsf{d}^{-1}, \mathsf{s}, \mathsf{s}^{-1}, \mathsf{f}, \mathsf{f}^{-1}, \mathsf{m}, \mathsf{m}^{-1}, \mathsf{o}, \mathsf{o}^{-1}\}$. Die sich ergebenden CSPs werden rekursiv mit demselben Algorithmus gelöst, beginnend mit dem Herstellen der Pfadkonsistenz in Schritt 1.

Eingabe

Das zu implementierende Programm soll wie folgt aufgerufen werden:[1]

```
# ./allen <csp>
```

Dabei ist `<csp>` der Name einer Datei, die die Constraints enthält. Jede Zeile in dieser Datei bezeichnet einen Constraint, oder sie besteht nur aus Leerraum oder hat als erstes nicht-leeres Zeichen ein `#`, in welchem Falle sie ignoriert wird. Constraints werden wie folgt notiert:

```
<Variable 1> <Relation 1> ... <Relation k> <Variable 2>
```

Wir schreiben `<` für $\prec$, `>` für $\succ$, `m` für m, `mi` für m^{-1} und so weiter. Erlaubte Variablen sind die Großbuchstaben `A` bis `Z`. Enthält die Datei für ein Variablenpaar `X` und `Y` keinen Eintrag, so ist dies äquivalent zu

```
X = < > d di s si f fi m mi o oi Y
```

und sollte vom Programm auch so behandelt werden. Insbesondere muss die *Lösung* des CSPs auch dann die Relation zwischen `X` und `Y` angeben, wenn in der Eingabe kein Constraint spezifiziert wurde.

Ausgabe

Je nach dem, ob das CSP lösbar ist oder nicht, sollte die erste Zeile der Ausgabe entweder

```
Solution found.
```

oder

```
No solution.
```

lauten. Die folgende Zeile gibt die Zahl der betrachteten Suchknoten nach dem folgenden Muster an:

```
112 nodes searched.
```

Die ausgegebene Zahl sollte dabei gleich der Zahl der Aufrufe des Verfahrens zum Herstellen der Pfadkonsistenz sein. Bei lösbaren Instanzen sollte abschließend eine Leerzeile und dann die Lösung ausgegeben werden, und zwar indem für jedes Paar von Variablen `X` und `Y` des CSPs eine Zeile der Art

```
X mi Y
```

notiert wird, und zwar in alphabetischer Reihenfolge der Variablen (erst nach der ersten Variable sortiert, bei gleicher erster Variable nach der zweiten). Um redundante Ausgaben zu vermeiden, sollen nur Paare berücksichtigt werden, bei denen die erste Variable im Alphabet vor der zweiten steht.

Beispiel

Nehmen wir an, die Datei `example.csp` habe folgenden Inhalt:

```
A < m B
B d o f C
A d D
B < = > D
```

Dann sollte der Aufruf

```
./allen example.csp
```

die folgende Ausgabe erzeugen:

[1] Passen Sie hier und im Folgenden die Kommandos entsprechend an, wenn Sie eine interpretierte Programmiersprache wie Java verwenden.

```
Solution found.
4 nodes searched.

A < B
A < C
A d D
B d C
B > D
C > D
```

Hinweise

Achten Sie bitte darauf, dass die Zählung der Knotenzahl, die Auswahl des nächsten Variablenpaars und die Auswahl der nächsten einzusetzenden Relation **genau so** erfolgt wie in der Aufgabe verlangt, da Sie sonst keine vergleichbaren Ergebnisse erhalten werden.

Die für die Herstellung der Pfadkonsistenz benötigte Kompositionstabelle der Intervallalgebra ist auf der Übungsseite der Vorlesung verfügbar:
`http://www.informatik.uni-freiburg.de/~ki/teaching/ws0405/krr/uebungen_de.html`.

Aufgabe P2.2 (Splitting mit ORD-Horn, 1 Punkt)

Implementieren Sie ein Programm, das für jede Relation $R \in \mathcal{A}$ eine minimal große Menge von Relationen aus $\mathcal{H}$ ermittelt, deren Vereinigung R ergibt. Ihr Programm sollte als

```
# ./horncover
```

aufgerufen werden und insgesamt $2^{13} = 8192$ Zeilen der Form

```
<Relation>: <Horn-Relation 1>, ..., <Horn-Relation k>
```

ausgeben. Relationen werden dabei wie bisher als durch Leerzeichen getrennte Basisrelationen ausgegeben. Beispielsweise könnte eine Zeile der Ausgabe

```
= < > m mi o oi: = < m o, = > mi oi
```

lauten, da die ORD-Horn-Relationen $\{\equiv, \prec, \mathsf{m}, \mathsf{o}\}$ und $\{\equiv, \succ, \mathsf{m}^{-1}, \mathsf{o}^{-1}\}$ eine Überdeckung von $\{\equiv, \prec, \succ, \mathsf{m}, \mathsf{m}^{-1}, \mathsf{o}, \mathsf{o}^{-1}\}$ mit minimaler Kardinalität bilden. Die Basisrelationen innerhalb einer Relation geben Sie bitte in derselben Reihenfolge an wie in Aufgabe P2.1. Die Reihenfolge der Relationen der Überdeckung und die Reihenfolge der Ausgabezeilen ist beliebig.

Hinweise

Mit dem falschen Algorithmus kann man bei diesem Problem leicht zu einem Programm kommen, das nicht in vertretbarer Zeit terminiert. Fragen Sie nach, wenn Sie Hilfe beim Entwurf des Algorithmus benötigen.

Die Liste der Hornrelationen ist auf der Übungsseite der Vorlesung verfügbar:
`http://www.informatik.uni-freiburg.de/~ki/teaching/ws0405/krr/uebungen_de.html`.

Aufgabe P2.3 (Backtracking mit ORD-Horn, 1 Punkt)

Modifizieren Sie das Programm aus Aufgabe P2.1 so, dass es im Aufteilungsschritt die Relationen nicht in Basisrelationen auflöst, sondern in Elemente der Klasse $\mathcal{H}$. Das neue Programm sollte als

```
# ./allen-horn <csp>
```

aufgerufen werden, wobei Ein- und Ausgabe wie in Aufgabe P2.1 zu gestalten sind. Der modifizierte Algorithmus sieht dann wie folgt aus:

1. Zunächst wird Pfadkonsistenz hergestellt.
2. Ergibt sich ein leerer Constraint, wird dieser Suchzweig abgebrochen.
3. Ergibt sich ein CSP, bei dem alle Relationen in $\mathcal{H}$ liegen, wird die Lösung mit dem Algorithmus aus Aufgabe P2.1 berechnet. (Es sind dann nur noch linear viele Schritte nötig.)
4. Tritt keiner dieser Fälle ein, werden Variablen X und Y ausgewählt, so dass die Relation zwischen X und Y noch keine Hornrelation ist. Es wird immer ein Variablenpaar gewählt, für das die Überdeckung des aktuellen Constraints zwischen X und Y durch Hornrelationen minimale Kardinalität hat. Es werden wieder nur Paare berücksichtigt, bei denen X im Alphabet vor Y steht.
5. Nun wird nacheinander der Constraint zwischen X und Y mit den Hornrelationen aus der Überdeckung instantiiert. Die sich ergebenden CSPs werden rekursiv mit demselben Algorithmus gelöst, beginnend mit dem Herstellen der Pfadkonsistenz in Schritt 1.