Universität Ulm
Fakultät für Informatik
Prof. Dr. Thom Frühwirth

Constraint-Programmierung, WS 02/03
Übungsblatt 9

Abgabe bis Mi, 22.01.03, 10:00

Constraintsystem FD

Lösen Sie *eine* der folgenden Aufgaben. Geben Sie zu Ihrer Lösung *alle* Antworten bzw. mindestens vier Bespielsanfragen mit allen Antworten ab.

Ergänzen Sie zur Lösung Ihrer Aufgabe einen der ersten beiden Constraintlöser von CHR Online `webchr www.pms.informatik.uni-muenchen.de/~webchr/` den Sie unter dem Eintrag *Finite and infinite domains* finden.

Aufgabe 9-1

Erweiteren Sie den Intervallbereichs-Constraintlöser *Straightforward interval domains...* um weitere Regeln für Aufzählungsbereiche der Form `X::L`, wobei `L` eine Liste ist, damit die Constraints `le/2, eq/2, ne/2, add/3, mult/3` behandelt werden können. Orientieren Sie sich dabei am Löser *Enumeration list...* und erweiteren Sie ihn entsprechend.

Als Enumerationsverfahren zur Suche implementieren Sie zusätzliche Regeln für `label(X)` `und split(X)`, wobei ersteres die Variable `X` der Reihe nach mit Werten aus dem Aufzählungsbereich von `X` bindet, während `split` den Aufzählungsbereich in zwei möglichst gleich große Teile teilt.

Testen Sie ihren Constraintlöser an mindestens vier nicht-trivialen Beispielen.

Aufgabe 9-2

Modifizieren und erweiteren Sie den Intervallbereichs-Constraintlöser *Straightforward interval domains...*, indem Sie Constraints der Form $AA :: I$ zulassen, wobei AA ein arithmetischer Ausdruck mit den Operatoren $+$ sowie $*$ ist und I ein Intervall der Form $A : B$ (A, B sind Zahlen) ist.

1. Zerlegen Sie nun Constraints der Form $AA : I$, wo AA keine Variable ist, zusätzlich in Constraints der Teilausdrücke von AA, zum Beispiel:
`A+B::I ==> A::J, B::J.`
wobei `J` für das größtmögliche Zahlen-Intervall in Sicstus-Prolog steht.

2. Die Darstellung der Constraints `add/3` und `mult/3` wird geändert: Statt der Regel für `add`
`add @ add(X,Y,Z), X::A:B, Y::C:D, Z::E:F ==> X::E-D:F-C, Y::E-B:F-A, Z::A+C:B+D.`
schreiben Sie
`add @ X::A:B, Y::C:D, (X+Y)::E:F ==> X::E-D:F-C, Y::E-B:F-A, (X+Y)::A+C:B+D.`
Analog für die Regeln für `mult/3`.

3. Testen Sie ihren Constraintlöser an mindestens vier nicht-trivialen Beispielen.

Aufgabe 9-3 Krypto-arithmetische Rätsel I

Krypto-arithmetische Rätsel sind Darstellungen arithmetischer Berechnungen, bei denen die Ziffern durch Buchstaben verschlüsselt worden sind. Es gilt, diese Rechnungen zu rekonstruieren, wobei zu beachten ist, daß gleichen Buchstaben gleichen Ziffern und verschiedenen Buchstaben verschiedenen Ziffern entsprechen. Für einen Buchstaben darf nicht Null eingesetzt werden, wenn er an führender Position steht.

Beispiele:

		S	E	N	D
+		M	O	R	E
=	M	O	N	E	Y

		T	E	S	T	E
+		F	E	S	T	E
+		D	E	I	N	E
=	K	R	AE	F	T	E

Schreiben Sie ein Programm für einen Constraintlöser mit Aufzählungswertebereichen, das beliebige krypto-arithmetische Rätsel lösen kann.

Die übereinander stehenden Wörter sollen dabei in eine Gleichung transformiert werden. Dabei muß zwischen den einzusetzenden Ziffern und den Buchstaben eine eindeutige Beziehung bestehen. Folgende Struktur des Programmes wird empfohlen:

```
gen_diff_digits(Buchstabengesamtliste),
row_value(Zeilenbuchstabenliste_1,Zeilenzahl_1),
...
row_value(Zeilenbuchstabenliste_n,Zeilenzahl_n),
Zeilenzahl_n eq Zeilenzahl_1 + ... + Zeilenzahl_n-1,
enumeriere(Buchstabengesamtliste).
```

`gen_diff_digits` bewirkt, dass die Listenelemente (Buchstaben) voneinander paarweise verschiedene Ziffern sind.
`row_value` besagt dass die Zeilenbuchstabenliste, als Zahl gelesen den Wert `Zeilenzahl` hat, z.B. `row_value([H,U,G,O],3472)` wenn `H=3, U=4, G=7, O=2`. Das Constraint kann z.B. rekursiv implementiert werden.
`enumeriere` ist der Such-Teil des Programmes, es probiert für jeden Buchstaben in der Liste eine der möglichen Ziffern aus.

Hinweis: Sie brauchen wahrscheinlich ein Constraint der Form $10X + Y = Z$, geschrieben als `add10(X,Y,Z)`. Implementieren Sie dieses Constraint analog zu dem Constraint $X + Y = Z$, geschrieben als `add(X,Y,Z)`. Es reicht also aus, nur die Propagierung in Richtung $\widehat{Z}$ zu implementieren.

Aufgabe 9-4 Krypto-arithmetische Rätsel II

Folgendes Pyramiden-Rätsel soll gelöst werden mit einem Constraintlöser mit Aufzählungswertebereichen:

```
            A
         B     C
      D     E     F
   G     H     I     J
K     L     M     N     O
```

In diesem Schema sollen die Zahlen von 1 bis 15 so untergebracht werden, daß jede Zahl die absolute Differenz der beiden darunterstehenden Zahlen ergibt (also z. B. $D = |G - H|$).

Mit folgendem Programmschema soll das Rätsel gelöst werden:

```
L= [A,B,C,D,E,F,G,H,I,J,K,L,M,N,O],
wertebereiche(L),
alle_verschieden(L),
positionen(L),
enumeriere(L).
```

Das Prädikat `wertebereiche(L)` fügt für jedes Element `X` aus `L` ein Constraint der Form `X::D` in den Constraintspeicher einfügt, wobei `D` die Liste der Zahlen von 1 bis 15 ist. Das Prädikat `alle_verschieden(L)` stellt sicher, daß alle Elemente von `L` paarweise verschieden sind. `positionen(L)` fügt Constraints der Form `ueber(Oben,Untenlinks,Untenrechts)` in den Constraintspeicher ein. Diese Constraints spiegeln die Struktur der Pyramide wieder. `enumeriere(L)` bindet jede Variable von `L` an einen Wert aus ihrem Wertebereich.

Hinweise: Aus Effizienzgründen empfiehlt sich, daß`enumeriere(L)` die Variablen in der Liste von hinten nach vorne enumeriert, also hier beginnend mit Variable `O` und endend mit `A`. Es reicht aus, für das Constraint `ueber(D,G,H)` die Propagierung von `G` und `H` nach `D` zu implementieren, also nur in diese Richtung. Orientieren Sie sich an den Regeln für `add(G,H,D)`.