Endliche Wertebereiche FD

Constraintsystem (I)

Signatur:

- Funktionssymbole: Integer, Listen, + und ..
- Constraintsymbole: $=, \leq, <, >, \geq, \neq$ und ::

Erlaubte Constraints:

$$\begin{array}{rl} C ::= & \mathit{true} \mid \mathit{false} \mid \\ & X :: n..m \mid X :: [k_1, \ldots, k_l] \mid \\ & X \odot Y \mid X + Y = Z \mid C \wedge C \end{array}$$

wobei

- $n,\ m,\ k_1, \ldots, k_l$ sind Integer und $0 \leq l$
- $\odot \in \{=, <, \leq, >, \geq, \neq\}$
- $X,\ Y$ und Z sind Variable (oder Integer)

Stand 2000

Anmerkung

Idee der Constraintnetzwerke über endlichen Wertebreichen aus der Künstlichen Intelligenz in eine LP-Sprache integriert. Dies geschah erstmals in der Programmiersprache CHIP Ende der achtziger Jahre.

Signatur ohne Beschränkung der Allgemeinheit nur ganze Zahlen. Vorteil, daß uns auch arithmetische Operationen zu Verfügung stehen. Mit den erlaubten Constraints läßt sich jede lineare Gleichung bzw. Ungleichung darstellen: Zum Beispiel ist das Constraint $3X + Y \geq Z$ äquivalent zur Konjunktion der erlaubten Constraints $Z1 \geq Z \wedge Z1 =_{FD} X1 + Y \wedge X1 =_{FD} X + X2 \wedge X2 =_{FD} X + X$.

Endliche Wertebereiche *FD*

Constraintsystem (II)

Constrainttheorie:

Presburger Arithmetik ist das
lineare Fragment der ganzzahligen Arithmetik,
ganze Zahlen mit + und =.
Vollständig und entscheidbar.

$X :: Dom$ bedeutet $X \in Dom$:

- Enumerations-, (Aufzählungs-)Bereich
  $X :: [k_1, \ldots, k_l] \leftrightarrow X = k_1 \vee \ldots \vee X = k_l$
- Interval-Bereich
  $X :: n..m \leftrightarrow n \leq X \wedge X \leq m \wedge integer(X)$

Anmerkung

`X::`D_X ordnet einer Variablen X einen endlichen Wertebereich D_X zu. Der Wertebereich D_X entweder als *Aufzählungsbereich* $[k_1, ..., k_l]$ oder als *Intervallbereich* $n..m$. Es läßt sich zwar jeder Intervallbereich als Aufzählungsbereich ausdrücken, allerdings werden wir sehen, daß Intervallbereiche effizienter implementiert werden können.

Constraintnetzwerke

Binäre Constraintnetzwerke:

- Variablen
- Unäre Constraints über endliche Wertebereichen
- Binäre Constraints

Konjunktion solcher Constraints wird dargestellt als *gerichteter Graph*:

- Knoten = Variablen mit Wertebereichen
- Kanten = Binäre Constraints

Lösung eines Netzwerks:

Wertebelegung von Variablen, die alle Constraints erfüllt.

Anmerkung

Konsistenzverfahren

Lokale Konsistenz (Lokale Propagierung)

Löse wiederholt Teilprobleme:

- Betrachte kleine, fixe Anzahl von Variablen und Constraints
- Verwende sie um neue, redundante Constraints zu propagieren
- Vereinfache die alten Constraints mit Hilfe der neuen C.
- Betrachte Teilprobleme mit veränderten Constraints
- Stop wenn keine Änderung mehr (Fixpunkt)

Globale Konsistenz = Lokale Konsistenz + Suchverfahren (abwechselnd verzahnt)

Anmerkung

Suche durch Enumerationsverfahren (engl. labeling): Bindet die Variablen an ihre möglichen Werte.

Man filtert dabei inkompatible Teillösungen einer begrenzten Anzahl von Constraints aus dem Netzwerk, indem man neue, bisher implizite Constraints hinzufügt, die die lokale Inkonsistenz ausschließen. Wir sagen auch, daß die bisherigen Constraints die neuen Constraints *propagieren* und sprechen von Konsistenzverfahren als *lokaler Propagierung.* Die propagierten Constraints sind zwar redundant, können aber u. U. zu einer Vereinfachung der bisherigen Constraints führen. Dies wiederholt man solange, bis das Netzwerk keine neuen Constraints mehr produziert. Das so vereinfachte Netzwerk läßt sich leichter nach einer globalen Lösung durchssuchen. Um *globale Konsistenz* zu sichern, müssen die Konsistenzverfahren aber mit Suchverfahren kombiniert werden. Dabei wechseln Constraintbehandlung und Suche einander ab. Zur Steuerung der Suche wendet man meist ein Enumerationsverfahren (engl. labeling) an, daß die Variablen an ihre möglichen Werte bindet. Durch das Binden einer Variablen wird neue Information erzeugt, die der Constraintlöser zur weiteren Vereinfachung der Constraints verwenden kann.

Beispiel

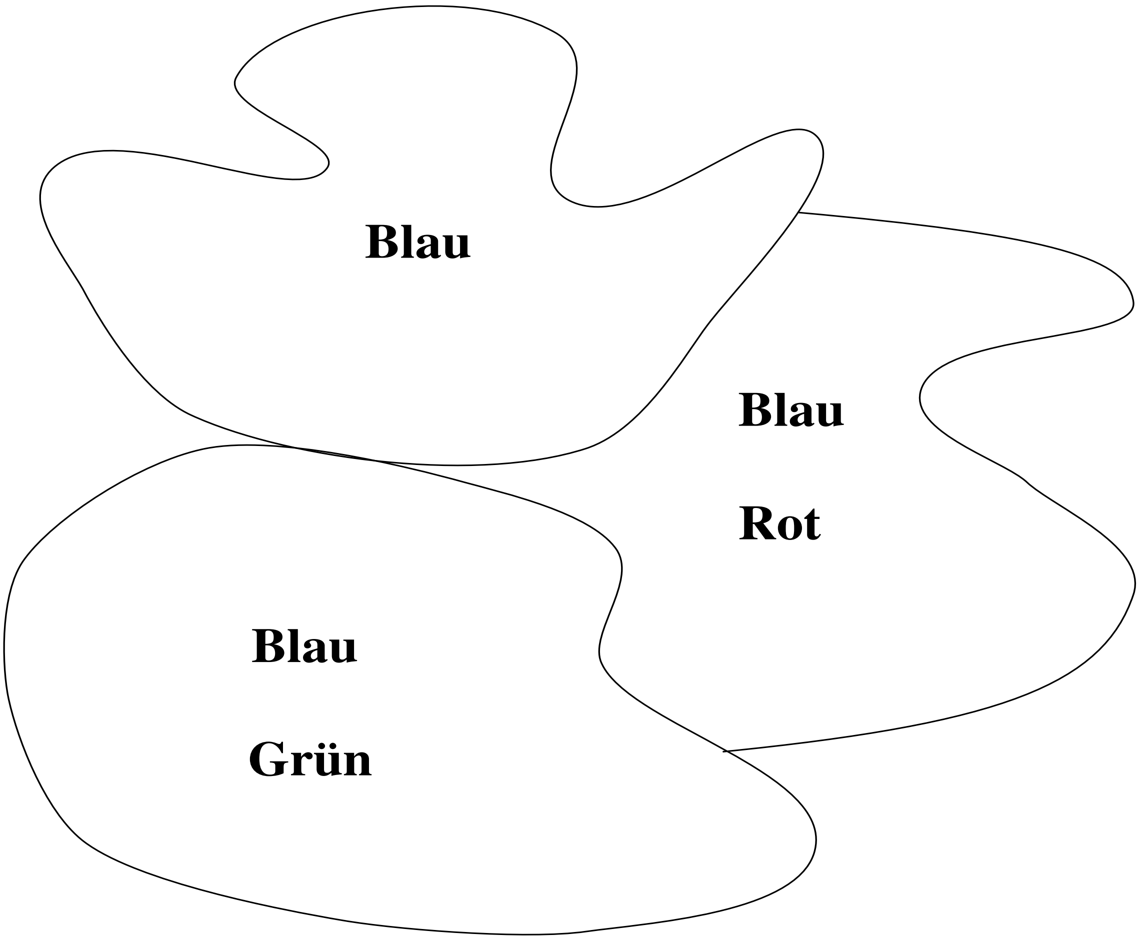


Abbildung 1: Das Färbeproblem dreier Länder

Anmerkung

Einfärben von Ländern auf einer Landkarte, wobei benachbarte Länder verschiedenfarbig sein müssen. Der zugehörige endliche *Wertebereich* ist die Menge der Farben Rot, Grün und Blau.

Beispiel, Fortsetzung

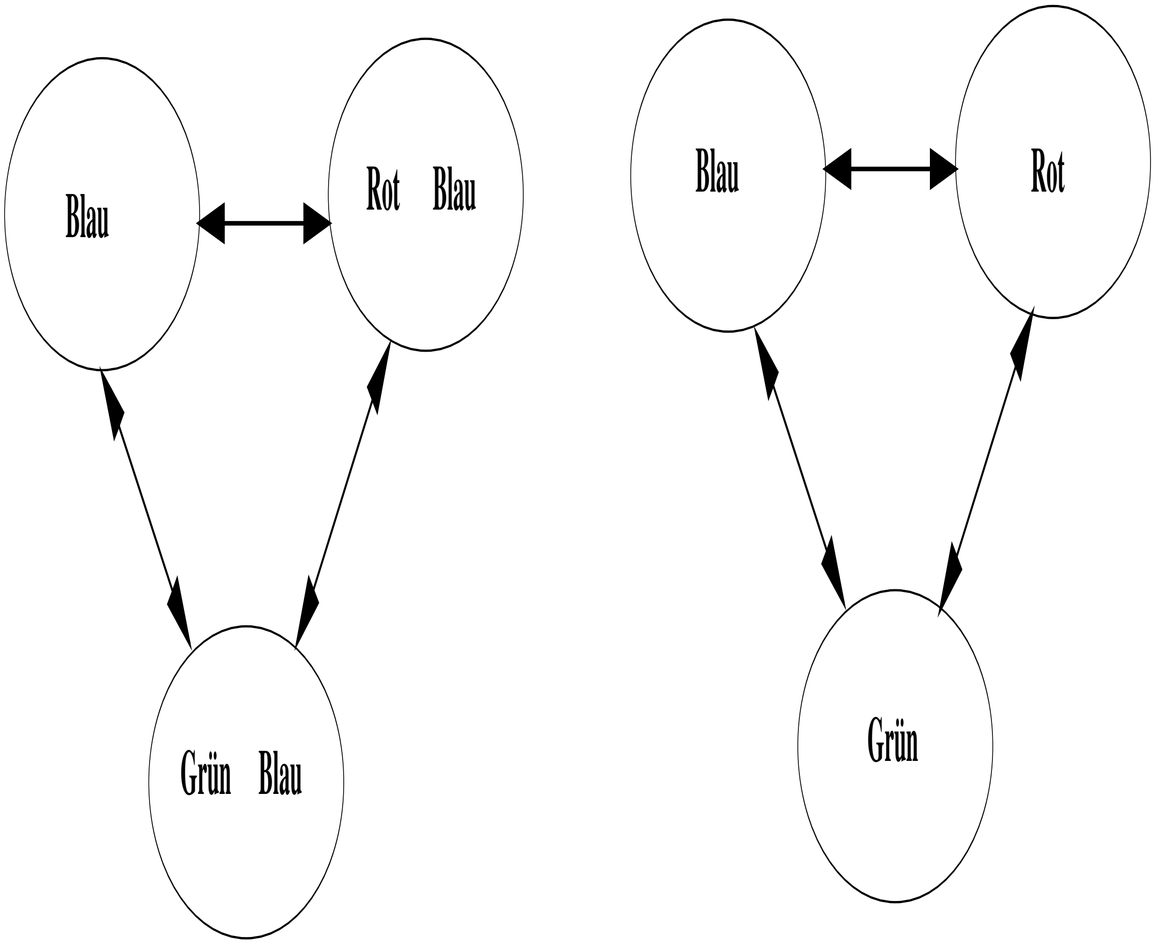


Abbildung 2: Vereinfachen eines Constraintsnetzwerkes

Anmerkung

Wenn man z. B. von drei benachbarten Ländern weiß, daß eines davon blau eingefärbt ist, und die beiden anderen rot oder blau bzw. grün oder blau eingefärbt werden können, ergibt sich durch Herausfiltern der inkompatiblen lokalen Teillösungen (Blau/Blau) sofort, daß man für die beiden letzteren Länder die Farben Rot bzw. Grün nehmen muß, will man das Constraint der Verschiedenfarbigkeit nicht verletzen. Damit erspart man sich, alle möglichen (d. h. exponentiell viele) Farbkombinationen nach einer Lösung durchzusuchen, wie das bei einer Formulierung im LP-Kalkül der Fall wäre.

Kantenkonsistenz in *FD*

Definition 1 Das Constraint $X_1 :: D_1 \ \wedge \ldots \wedge \ X_n :: D_n$ heißt *kantenkonsistent* (engl. arc-consistent) bezüglich eines Constraints $c(X_1, \ldots, X_n)$, wenn es für alle $i \in \{1, \ldots, n\}$ und für alle möglichen Wertebelegungen von X_i aus D_i Wertebelegungen für die Variablen X_j $(i \neq j)$ aus den jeweiligen Wertebereichen D_j gibt, sodaß das Constraint $c(X_1, \ldots, X_n)$ erfüllt ist.

Das heißt, jeder Wert aus jedem Wertebereich ist an mindestens einer Lösung von c beteiligt.

Beispiele:
$X :: [1, 2, 3], Y :: [1, 2, 3], X < Y$ nicht kantenkonsistent.
$X :: [1, 2], Y :: [2, 3], X < Y$ kantenkonsistent.
$X :: D, X \neq X$ kantenkonsistent für alle $|D| > 1$.

Anmerkung

Man betrachtet mehrfache Vorkommen derselben Variable als voneinander unabhängig. Das inkonsistente Constraint `X`$\neq_{FD}$`X` z. B. ist für jeden Wertebereich von `X` kantenkonsistent, der mehr als einen Wert enthält. Kantenkonsistenz allein garantiert also nicht immer globale Konsistenz.

Herstellung von Kantenkonsistenz

Ausgehend von der logischen Formel:
$X_1 :: D_1 \wedge \ldots \wedge X_i :: D_i \wedge \ldots \wedge X_n :: D_n$
$\wedge\ c(X_1, \ldots, X_n) \quad \rightarrow \quad X_i :: D_i',$
wobei D_i' sich nach Möglichkeit von D_i unterscheidet. Dann ist $D_i' \cap D_i$ ein neuer, kleinerer Wertebereich.

Prozedural: Entferne für jede Variable jene Werte aus ihrem Wertebereich, für die man keinen „Partner" in den Wertebereichen der restlichen Variablen von c finden kann, sodaß c erfüllbar ist.

Beispiel:
Wertebereiche: $X, Y, Z :: [1, 2, 3]$
Binäre Constraints: $X < Y \wedge Y < Z \wedge Z \leq 2$

1. $X < Y$ impliziert $X :: [1, 2]$ and $Y :: [2, 3]$
2. $Z \leq 2$ impliziert $Z :: [1, 2]$
3. $Y < Z$ impliziert $Y :: []$ and $Z :: []$

Anmerkung

Für die Intervallbereiche gilt die Definition der Kantenkonsistenz ebenfalls. Mit einer Ausnahme: Die Ungleichheit $\neq_{FD}$ kann Bereiche mit "Löchern" erzeugen und diese sind als Intervall nicht darstellbar, dann ist ein solches Intervall aber nicht kantenkonsistent, sondern bloß *grenzen-konsistent* (engl. bound-consistency), die Konsistenz ist dabei nur für die Intervallgrenzen gefordert, gilt also nicht unbedingt für Zahlen im Intervallbereich.

Effizienzsteigernd reicht es aus, bloß mit den Intervallgrenzen zu rechnen, um sicherzustellen, daß auch alle Werte im Intervall möglich sind (mit einer Ausnahme von $\neq_{FD}$). In der Praxis behilft man sich wegen $\neq_{FD}$ mit endlichen Vereinigungen von Intervallen, die eine kompaktere Darstellung von Aufzählungsbereichen sind.

Die Komplexität ist linear in der Anzahl der Constraints (d. h. im Fall von *FD* kubisch in der Anzahl der Variablen. bzw. quadratisch, wenn nur binäre Constraints erlaubt wären). Wenn die Größe der Aufzählungsbereiche nicht begrenzt ist, ist die Komplexität des Verfahrens auch kubisch in dieser Größe.

Intervallbereiche in *FD*

```
nicht_leer @ X::Min..Max ==> Min =< Max.

schnitt @ X::A..B, X::C..D <=> X::max(A,C):min(B,D).

kleiner_x @ X ≤ Y, X::A..B, Y::C..D ==> B > D | X::A..D.
kleiner_y @ X ≤ Y, X::A..B, Y::C..D ==> A > C | Y::A..D.
```

Anmerkung

Wir implementieren zuerst Intervallbereiche, wobei wir auf die vordefinierten Prädikate `<`, `=<` und `is` zurückgreifen. Die ersten beiden CHR-Regeln sind eine Konsequenz aus dem Axiomenschema *Intervall*.

Varianten der Regeln für `kleiner`:
Mit Simpagierungsregeln schreiben oder:

```
kleiner @ X ≤ Y, X::A..B, Y::C..D ==>
          X::A..D, Y::A..D.
```

Intervallbereiche in *FD*

Beispiel: `A::2..3, B::1..2,` $A \leq B$

$\mapsto_{kleiner_x}$ `A::2..3, B::1..2,` $A \leq B$`, A::2..2`

$\mapsto_{schnitt}$ `B::1..2,` $A \leq B$`, A::2..2`

$\mapsto_{kleiner_y}$ `B::1..2,` $A \leq B$`, A::2..2, B::2..2`

$\mapsto_{schnitt}$ $A \leq B$`, A::2..2, B::2..2`

Anmerkung

Um den Wert einer eindeutig bestimmten (determinierten) Variable wie `A::2..2` explizit zu machen, können wir die Variable an ihren Wert binden, indem wir auf das vordefinierte Gleichheitsconstraint = zurückgreifen:

```
binden @ X::Min..Min <=> X = Min.
```

Allerdings muss man dann zusätzliche Regeln für alle Constraints schreiben für die Fälle, in denen ein oder mehrere Argumente Werte und nicht Variablen sind. Z.B.:

```
kleiner_x @ X ≤ Y, Y::C..D ==> X > D | false.
```

Zusätzliche Propagierungen gibt es dadurch nicht.

Ungleichheit propagiert nur schwach, Z.B.:

```
X ≠ Y, X::A..B, Y::C..D ==> A=B,B=C | Y::C+1..D.
```

```
z @ X+Y = Z, X::A..B, Y::C..D ==>
    E is A+C, F is B+D, Z::E..F.
y @ X+Y = Z, X::A..B, Z::E..F ==>
    C is E-B, D is F-A, Y::C..D.
x @ Y+X = Z, X::A..B, Z::E..F ==>
    C is E-B, D is F-A, Y::C..D.
```


Beispiel:	`A::1..3, B::2..4, C::0..4, A+B=C`
$\mapsto_z$	`A::1..3, B::2..4, C::0..4, A+B=C, C::3..7`
$\mapsto_{schnitt}$	`A::1..3, B::2..4, A+B=C, C::3..4`
$\mapsto_x \mapsto_y$	`A::1..3, B::2..4, A+B=C, A::-1..2, B::0..3, C::3..4`
$\mapsto^2_{schnitt}$	`A+B=C, A::1..2, B::2..3, C::3..4`

Anmerkung

Abhängigkeit des Verfahrens von der Größe des Wertebereichs:

Beispiel: Das Constraint `X::1..1000, X+1=X` kann in einem Schritt nur zu `X::2..1000, X=X+1` vereinfacht werden, es sind also insgesamt 1000 Schritte nötig (d. h. genauso viele wie der Bereich groß ist), um die Inkonsistenz zu erkennen. I. a. sind Inkonsistenzen nicht so offensichtlich und daher nicht effizient erkennbar.

Wie jedes lokale Konsistenzverfahren führt die Kantenkonsistenz nicht unbedingt zu einer vollständigen Vereinfachung der Constraints. So führt das Constraint `X::1..1000, X+Y=X` bloß zum Wertebereich `Y::-999..999`, statt zu `Y≐0`.

Aufzählungsbereiche in *FD*

```
X::[] <=> false.

X::L1, X::L2 <=> liste(L1),liste(L2) |
    durchschnitt(L1,L2,L3), X::L3.

X ≤ Y, X::L1, Y::L2 ==> liste(L1),liste(L2),
    list_max(L1,B), list_max(L2,D),
    B > D |
    filter_max(L1,D,L3), X::L3.
X ≤ Y, X::L1, Y::L2 ==> liste(L1),liste(L2),
    list_min(L1,A), list_min(L2,C),
    A > C |
    filter_min(L2,A,L3), Y::L3.

X+Y = Z, X::L1, Y::L2 ==>
    alle_additionen(L1,L2,L3), Z::L3.
...
```

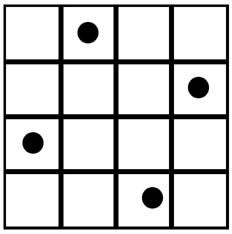
Aufzählungsbereiche in FD

Ziel: X $\leq$ Y, X :: [4,6,7], Y :: [3,7].
Antwort: X $\leq$ Y, X :: [4,6,7], Y = 7.

Ziel: X $\leq$ Y, X :: [2,3,4,5], Y :: [1,2,3].
Antwort: X $\leq$ Y, X::[2,3], Y::[2,3].

Ziel: X $\leq$ Y, X :: [2,3,4], Y :: [0,1].
Antwort: false.

n-Damen-Problem in *FD*



```
loese(N,Loesung):-
    wertebereich(N,Loesung),
    damen(N,Loesung), enum(Loesung).

damen(0,[]).
damen(N,[X|Xs]):- sicher(X,Xs,1),
    damen(N-1,Xs).

sicher(X,[],N).
sicher(X,[Y|T],N):- X≠Y, X+N≠Y, Y+N≠X,
    sicher(X,T,N+1).

enum([]).
enum([X|L]) :- indomain(X), enum(L).
```

Anmerkung

n Damen auf einem $n \times n$ Felder großem Schachbrett so zu positionieren, daß keine Dame eine andere bedroht. Zwei Damen bedrohen einander, wenn sie auf der gleichen Zeile, Spalte oder Diagonale eines Schachbrettes stehen.

Für das 2- und 3-Damen-Problem gibt es keine Lösung. Die Anzahl der Lösungen steigt mit n an. Das 4-Damen-Problem z. B. hat folgende Lösung. Seine zweite Lösung kann man durch Spiegelung erhalten.

Im folgenden ist eine Lösung des n-Damen-Problems `Lösung`, die mittels `loese(N,Loesung)` generiert wird, eine Liste mit `N` Elementen. Der Wert des i-ten Elements gibt die Zeile an, in der die Dame der Spalte i steht. Im obigen Beispiel lautet die Lösung also `[3,1,4,2]`.

Das Hilfsprädikat `wertebereich(N,Loesung)` ordnet jedem Element der Liste `Loesung` den Aufzählungsbereich `[1,2,...,N]` zu. Zum Enumerieren wird das häufig in *FD*-Implementierungen verwendete Prädikat `indomain/1` benutzt, das einer Variable der Reihe nach alle Werte aus ihrem Wertebereich zuordnet. Effizienter ist es meist, wenn Variablen mit kleinem Wertebereich zuerst enumeriert werden (engl. first failure principle).

Anwendungsbeispiel: Zeitplanung

FD-Variablen: Aufgabe A_i:

- S_i: Startzeit
- d_i: Dauer
- E_i: Endzeit ($S_i + d_i = E_i$)

Reihenfolge der Aufgaben: S_j kann erst beginnen, wenn S_i abgeschlossen ist ($S_i + d_i \leq S_j$).

Konkurrenz um ein Betriebsmittel:
$S_i + d_i \leq S_j \sqcup S_j + d_j \leq S_i$.

- S_i dauert d_i Zeiteinheiten,
- S_j dauert d_j Zeiteinheiten und
- S_i und S_j könnnen nicht gleichzeitig durchgeführt werden. Darstellung: `ct(Si,di,Sj,dj)`.

Anmerkung

Bei der Zeitplanung sind gewisse Aufgaben (engl. tasks, jobs) zu erledigen, die Vorbedingungen haben. Oft konkurrieren die Aufgaben auch um die Zuteilung von Betriebsmitteln (engl. resources). Ziel ist es, einen Zeitplan zu erstellen, bei dem jede Aufgabe erst dann beginnt, wenn ihre Vorbedingungen erfüllt sind, und bei dem die Betriebsmittelvergabe korrekt gelöst ist. Häufig möchte man noch ein Optimalitätskriterium, z. B. Minimalität der Gesamtdauer, erfüllen.

Anwendungsbeispiel: Zeitplanung

Ziel: `S1::1..6, S2::1..10, ct(S1,7,S2,6)`.

Die Disjunktion, die implizit im Constraint `ct(S1,7,S2,6)` steckt, legt eine Unterscheidung von zwei Fällen nahe:

- `S1` ist vor `S2`, dann Antwort `S1::1..3, S2::8..10`.
- `S2` ist vor `S1`, dann ist das Constraint inkonsistent.

Ziel: `S1::0..9, S2::4..9, S3::4..10, ct(S1,5,S2,5) ct(S1,5,S3,5), ct(S2,5,S3,5)`.

Abstand:	mind. 5 Zeiteinheiten
Frühestmögliche Zeitpunkt:	`0`
Letztmögliche Zeitpunkt:	`10`
Lösung:	`S1=0, S2=5, S3=10`

Anmerkung

Der Abstand zwischen den Aufgaben beträgt mindestens `5` Zeiteinheiten, der frühestmögliche Zeitpunkt ist `0`, der späteste `10`. Jede Lösung muß also die drei Aufgaben auf die Zeitpunkte `0, 5` und `10` legen. Aufgrund des Intervallbereiches von `S2` gibt es dafür nur eine Möglichkeit, nämlich `S1=0, S2=5, S3=10`. Um zu dieser Schlußfolgerung zu gelangen, bedurfte es jedoch einer globalen Betrachtung aller Constraints. Ein Constraintlöser, der auch global arbeitet, d.h. alle notwendigen Constraints auf einmal betrachtet, ist jedoch unter Umständen in der Zeitkomplexität teurer als einer, der immer nur eine beschränkte Anzahl von Constraints bearbeitet (d. h. mit lokalen Konsistenzverfahren arbeitet).