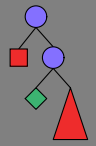


Constraintprogrammierung

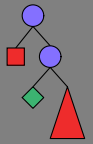
Eine Einführung

Niko Paltzer

Proseminar Programmiersysteme WS 03/04
Betreuer: Gert Smolka

Geschichte

- 1963: Das Constraint-Konzept wird in dem interaktiven Zeichen-Programm *SKETCHPAD* von I. Sutherland benutzt.

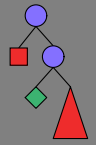








Geschichte



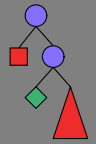
- 1963: Das Constraint-Konzept wird in dem interaktiven Zeichen-Programm *SKETCHPAD* von I. Sutherland benutzt.
- 70er: Verschiedene Algorithmen für die Lösung eines *constraint satisfaction problem* (CSP) werden in der KI-Forschung entwickelt. (z.B. Erkennung von dreidimensionalen Objekten: WALTZ, 1975)







Geschichte



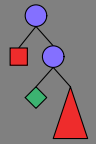
- 1963: Das Constraint-Konzept wird in dem interaktiven Zeichen-Programm *SKETCHPAD* von I. Sutherland benutzt.
- 70er: Verschiedene Algorithmen für die Lösung eines *constraint satisfaction problem* (CSP) werden in der KI-Forschung entwickelt. (z.B. Erkennung von dreidimensionalen Objekten: WALTZ, 1975)
- 80er: Erste Programmiersprachen für Constraintprogrammierung werden entwickelt. (um 1985: CHIP, Prolog II)





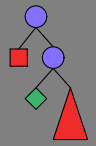


Geschichte



- 1963: Das Constraint-Konzept wird in dem interaktiven Zeichen-Programm *SKETCHPAD* von I. Sutherland benutzt.
- 70er: Verschiedene Algorithmen für die Lösung eines *constraint satisfaction problem* (CSP) werden in der KI-Forschung entwickelt. (z.B. Erkennung von dreidimensionalen Objekten: WALTZ, 1975)
- 80er: Erste Programmiersprachen für Constraintprogrammierung werden entwickelt. (um 1985: CHIP, Prolog II)
- 90er: Kombination von Forschungsergebnissen aus den Gebieten KI, Computer Algebra und Mathematische Logik (ab 1990: ILOG Solver, ab 1991: Oz)

Send More Money



Finden Sie eine eindeutige Ziffernbelegung für die Buchstaben S, E, N, D, M, O, R, Y, so dass folgende Gleichung erfüllt ist (führende Nullen sind nicht erlaubt):

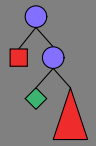
$$S\ E\ N\ D + M\ O\ R\ E = M\ O\ N\ E\ Y$$







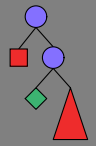
Send More Money



Finden Sie eine eindeutige Ziffernbelegung für die Buchstaben S, E, N, D, M, O, R, Y, so dass folgende Gleichung erfüllt ist (führende Nullen sind nicht erlaubt):

$$\begin{array}{cccccccccccccc} S & E & N & D & + & M & O & R & E & = & M & O & N & E & Y \\ 9 & 5 & 6 & 7 & + & 1 & 0 & 8 & 5 & = & 1 & 0 & 6 & 5 & 2 \end{array}$$

Safe



Knacken Sie den Safe! Der Code besteht aus einer Sequenz von 9 verschiedenen Ziffern $C_i \neq 0$ für die gilt:

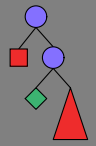
$$C_4 - C_6 = C_7$$
$$C_1 * C_2 * C_3 = C_8 + C_9$$
$$C_2 + C_3 + C_6 < C_8$$
$$C_9 < C_8$$
$$C_1 \neq 1, \ldots, C_9 \neq 9$$







Safe

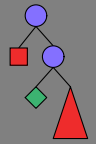


Knacken Sie den Safe! Der Code besteht aus einer Sequenz von 9 verschiedenen Ziffern $C_i \neq 0$ für die gilt:

$$C_4 - C_6 = C_7$$
$$C_1 * C_2 * C_3 = C_8 + C_9$$
$$C_2 + C_3 + C_6 < C_8$$
$$C_9 < C_8$$
$$C_1 \neq 1, \ldots, C_9 \neq 9$$

$$C_1 = 4,\ C_2 = 3,\ C_3 = 1$$
$$C_4 = 8,\ C_5 = 9,\ C_6 = 2$$
$$C_7 = 6,\ C_8 = 7,\ C_9 = 5$$

Map-Colouring



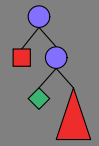
Gegeben eine Landkarte, färben Sie die Länder so ein, dass zwei benachbarte Länder unterschiedliche Farben haben und benutzten Sie dabei möglichst wenige verschiedene Farben.



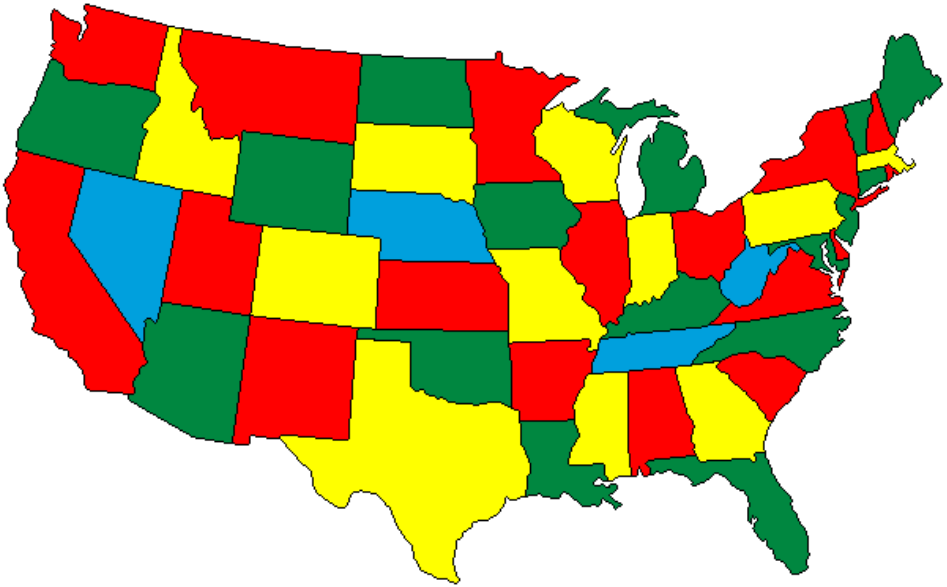




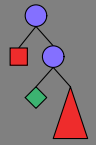
Map-Colouring



Gegeben eine Landkarte, färben Sie die Länder so ein, dass zwei benachbarte Länder unterschiedliche Farben haben und benutzten Sie dabei möglichst wenige verschiedene Farben.



Supermarkt



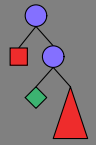
Sie haben im Supermarkt vier Artikel erstanden und dafür 7,11 € bezahlt. Erstaunt stellen Sie fest, dass das Produkt der Einzelpreise ebenfalls 7,11 € beträgt. Was kostet jeder der vier Artikel?







Supermarkt



Sie haben im Supermarkt vier Artikel erstanden und dafür 7,11 € bezahlt. Erstaunt stellen Sie fest, dass das Produkt der Einzelpreise ebenfalls 7,11 € beträgt. Was kostet jeder der vier Artikel?

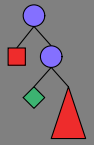
3,16 €

1,20 €

1,25 €

1,50 €

Damen-Problem



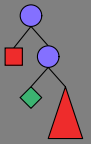
Platzieren Sie 8 Damen auf einem Schachbrett, ohne dass sie sich gegenseitig bedrohen.



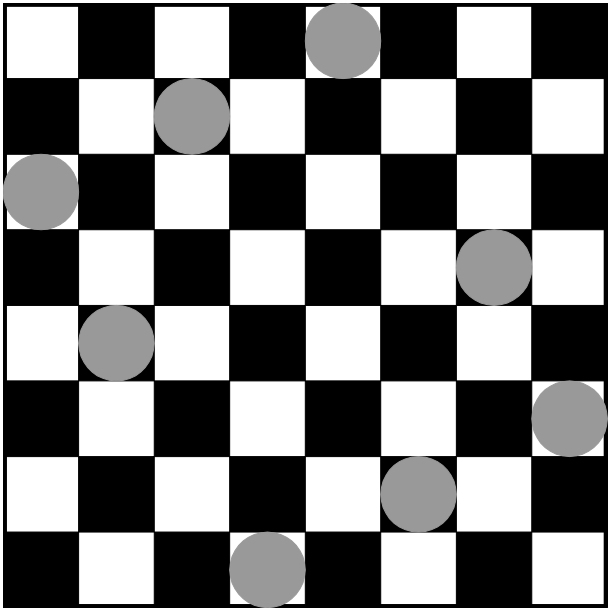




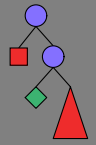
Damen-Problem



Platzieren Sie 8 Damen auf einem Schachbrett, ohne dass sie sich gegenseitig bedrohen.



Weitere Anwendungen



Abgesehen von logischen Spielereien:

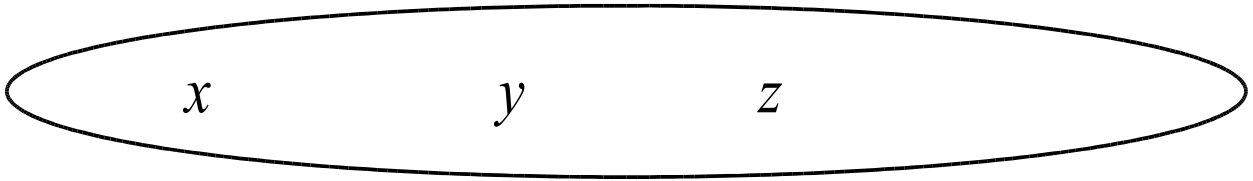
- Ablaufplanung
- Turnierplanung
- Personalplanung
- Zuschnittprobleme
- Sprachverarbeitung





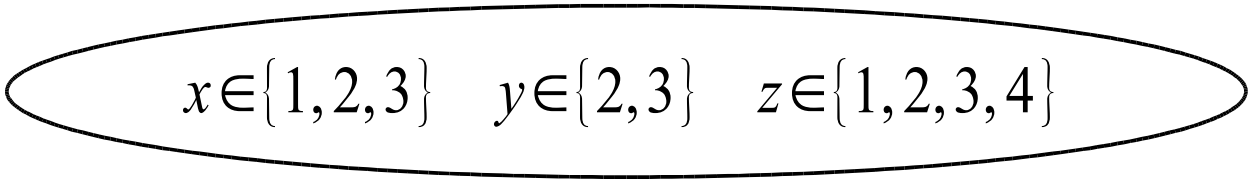


Variablenspeicher



Der Variablenspeicher enthält eine Menge von Variablen . . .

Variablenspeicher

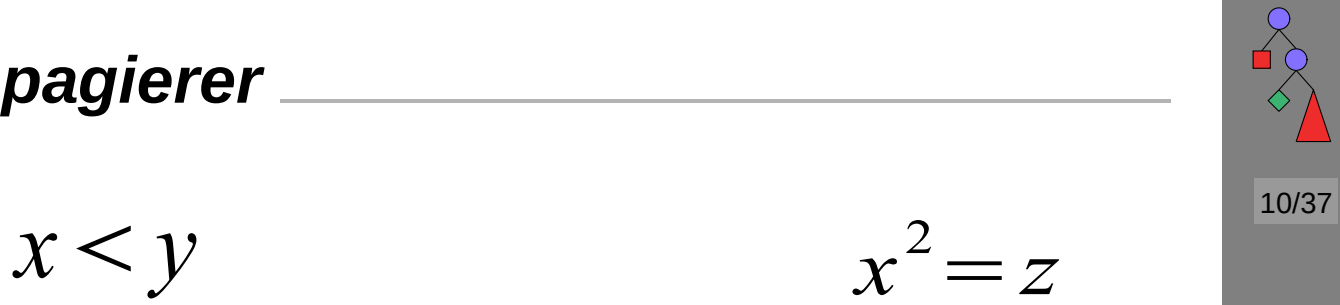


$$x \in \{1,2,3\} \quad y \in \{2,3\} \quad z \in \{1,2,3,4\}$$

Der Variablenspeicher enthält eine Menge von Variablen . . .

. . . und deren endliche Bereiche (*finite domains*).

Propagierer

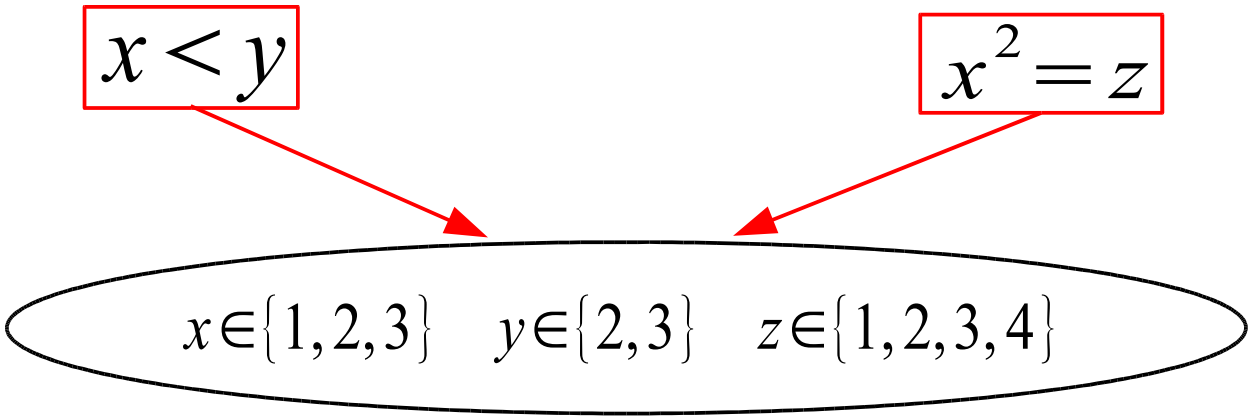


$$x < y \qquad x^2 = z$$

$$x \in \{1,2,3\} \quad y \in \{2,3\} \quad z \in \{1,2,3,4\}$$

Hinzu kommen Constraints . . .

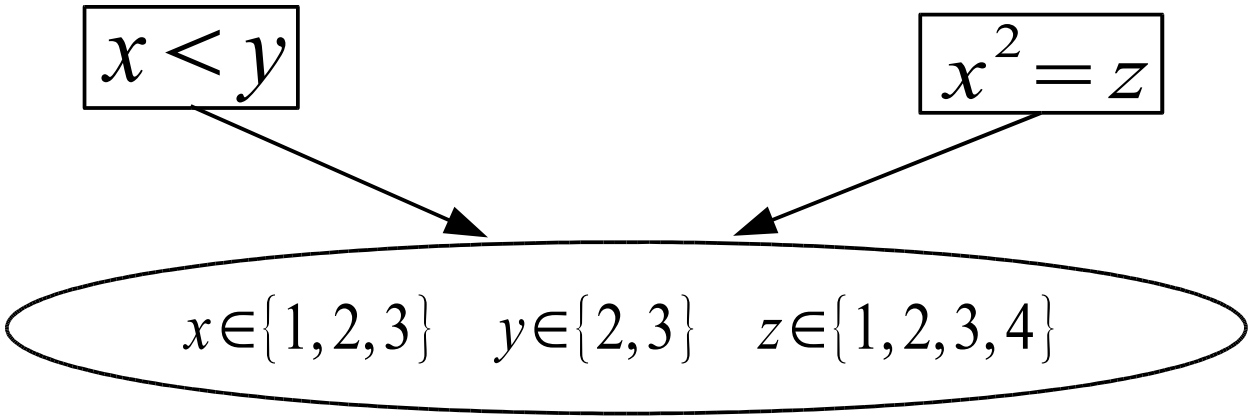
Propagierer



Hinzu kommen Constraints . . .

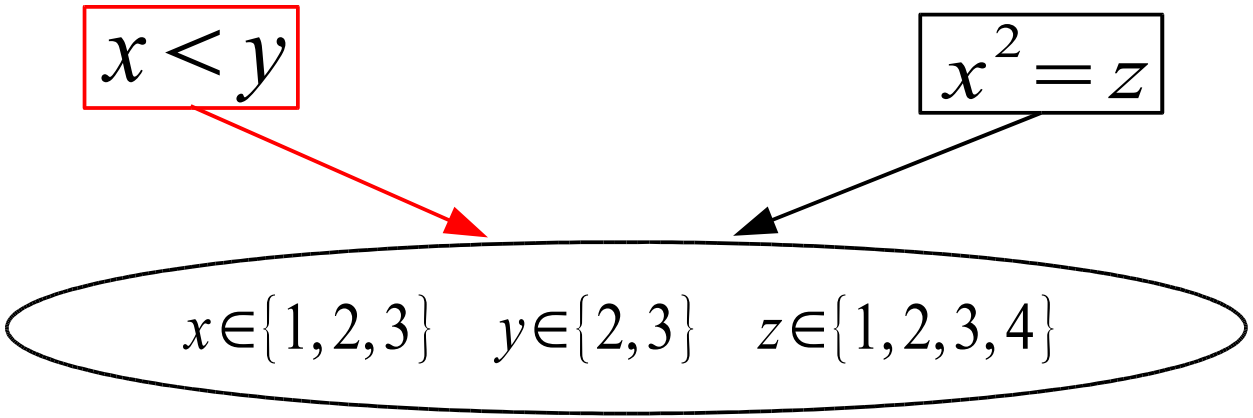
. . . die von Propagierern realisiert werden.

Konfiguration



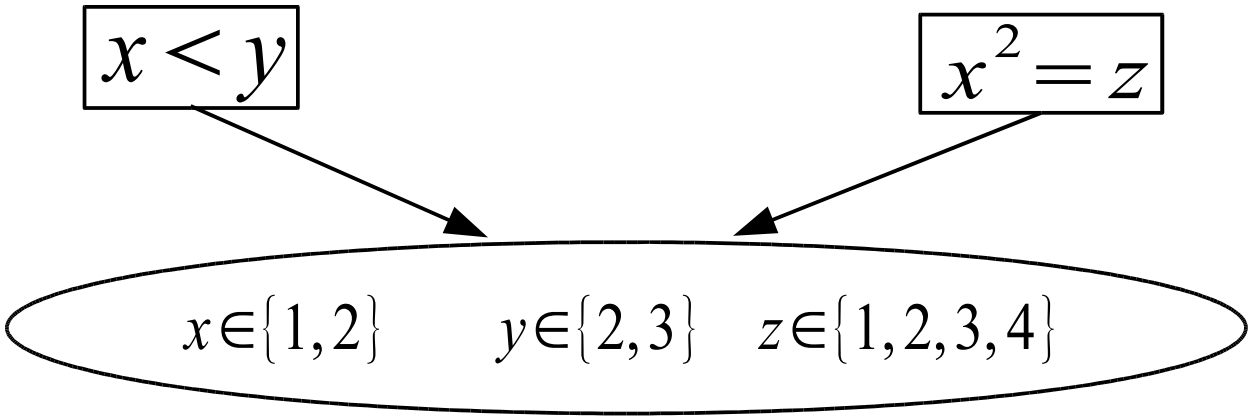
Zusammen ergibt es eine *Konfiguration* (*space*) des CSP.

Einengen der Bereiche



Der linke Propagierer beginnt . . .

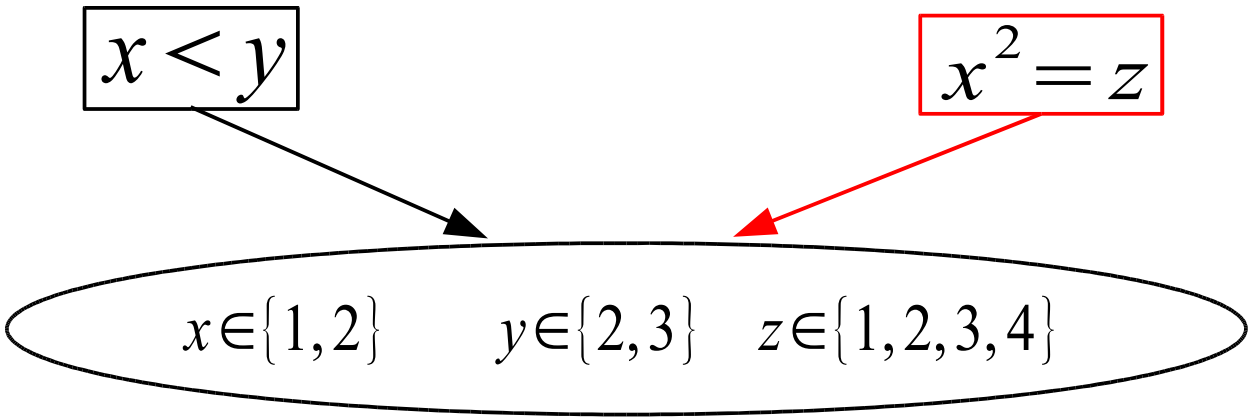
Einengen der Bereiche



Der linke Propagierer beginnt . . .

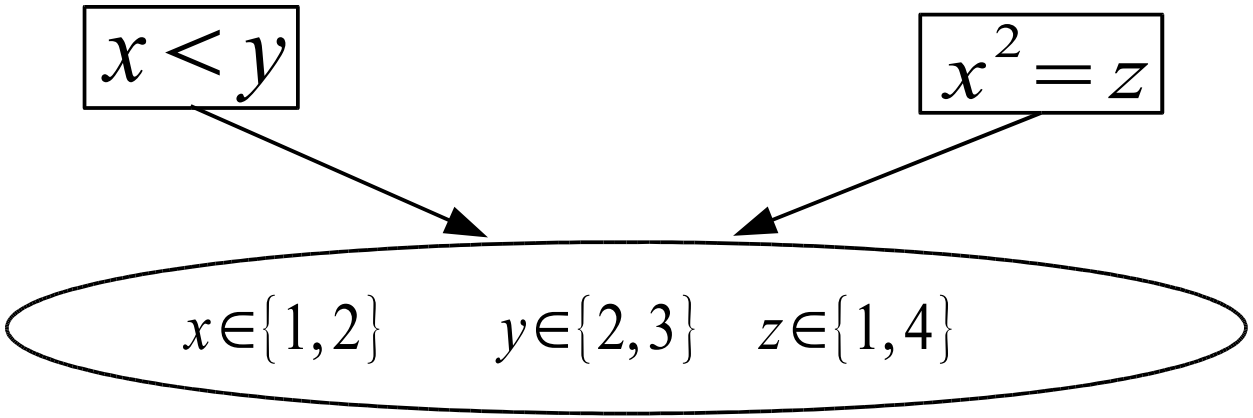
. . . und engt den Bereich von x ein.

Einengen der Bereiche



Der rechte Propagierer . . .

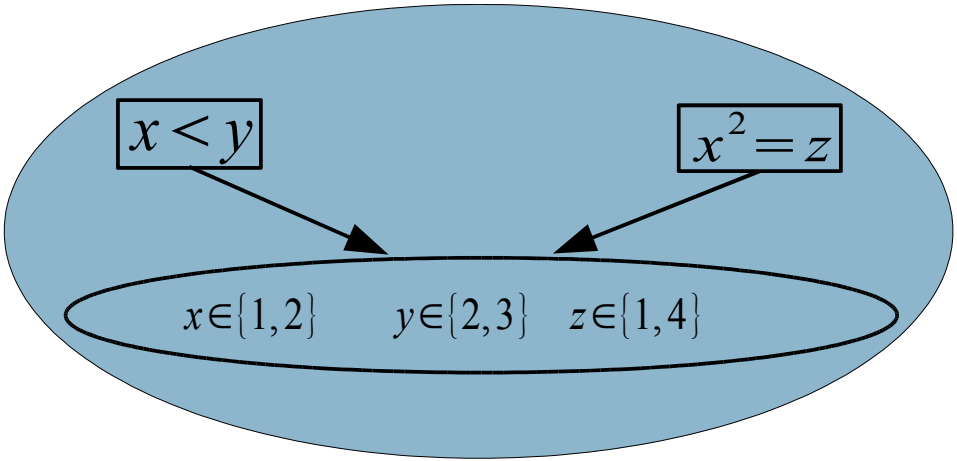
Einengen der Bereiche



Der rechte Propagierer . . .

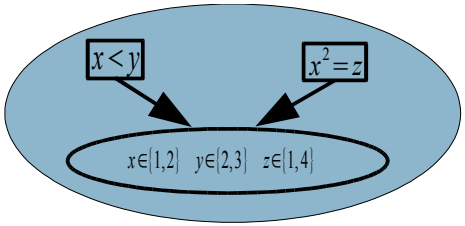
. . . kann den Bereich von z einengen.

Stabilität



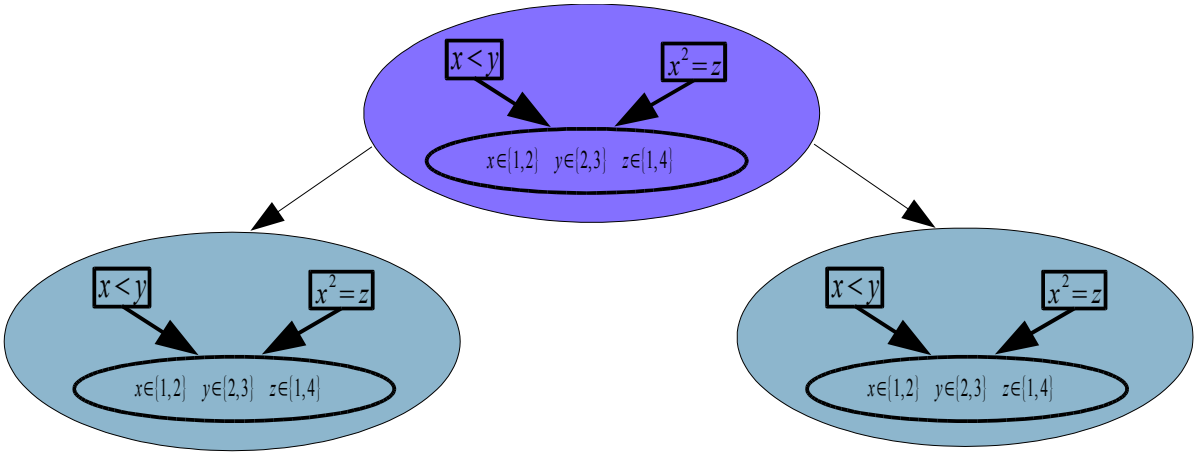
Die Konfiguration ist jetzt *stabil*, aber wir haben noch keine eindeutige Lösung gefunden.

Distribution



Also nehmen wir die Konfiguration . . .

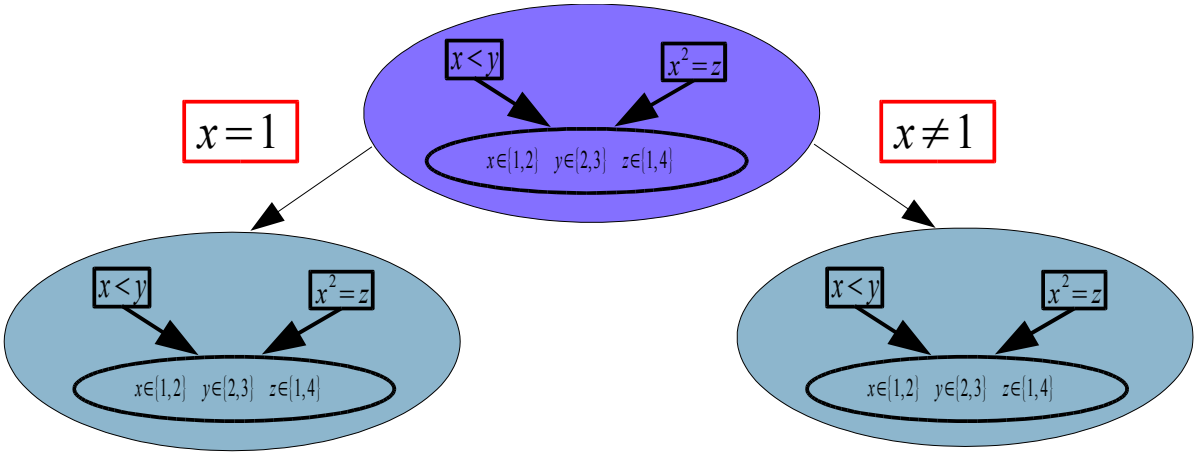
Distribution



Also nehmen wir die Konfiguration . . .

. . . kopieren sie zwei Mal . . .

Distribution



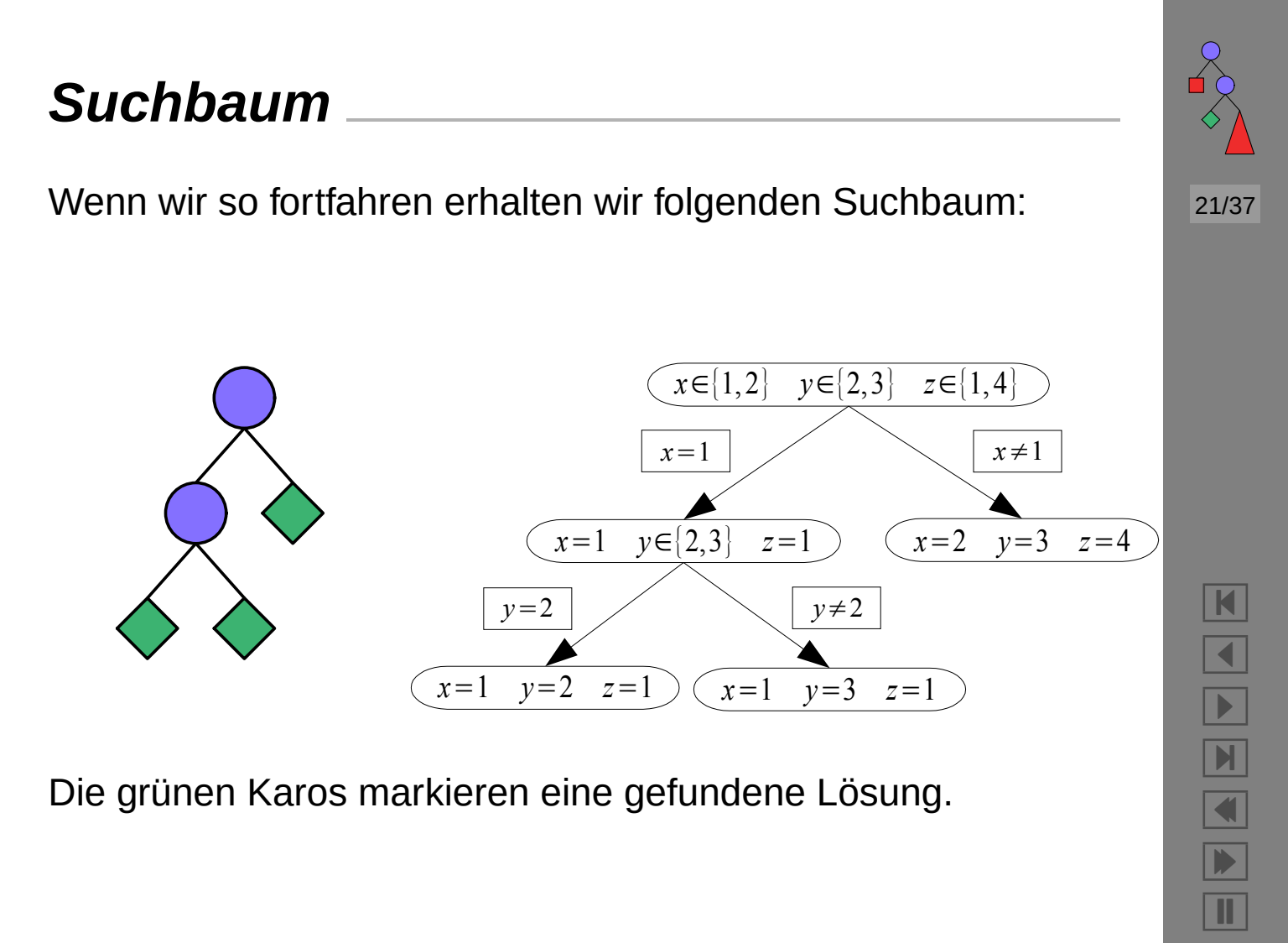
Also nehmen wir die Konfiguration . . .

. . . kopieren sie zwei Mal . . .

. . . und machen eine Fallunterscheidung.

Denn es gilt $A \Leftrightarrow (A \wedge C) \vee (A \wedge \neg C)$.

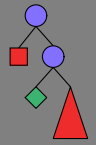
Suchbaum



Wenn wir so fortfahren erhalten wir folgenden Suchbaum:

Die grünen Karos markieren eine gefundene Lösung.

Distributions-Strategien



Bei den folgenden Standart-Strategien werden die Variablen als geordnete Sequenz betrachtet.

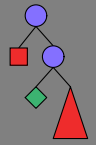
naive distribution

Distribution über die erste noch nicht determinisierte Variable der Sequenz

first-fail distribution

Distribution über die erste Variable, die noch nicht determinisiert ist und deren *domain* am kleinsten ist

Distributions-Strategien



Die Standart-Möglichkeiten der Distribution über einer Variablen x sind:

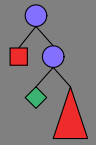
- $x = l$, wobei l der kleinste mögliche Wert für x ist
- $x = u$, wobei l der größte mögliche Wert für x ist
- $x = m$, wobei m ungefähr der mittlere Wert für x ist
- $x \leq m$, wobei m ebenfalls ungefähr in der Mitte der *domain* von x liegt (der mögliche Bereich für x wird geteilt)







Fragestellungen



Im Zusammenhang mit CSPs sind mehrere Fragestellungen möglich:

- Gibt es (mindestens) eine Lösung?
- Wie viele Lösungen gibt es?
- Wie sehen die Lösungen aus?
- Was ist die optimale Lösung?

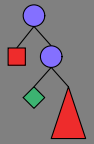
Um Antworten auf diese Fragen zu bekommen, verwendet man unterschiedliche Suchmaschinen.







Realisierung in Alice



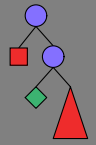
Constraints







Realisierung in Alice



Constraints

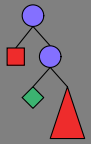
Fallunterscheider

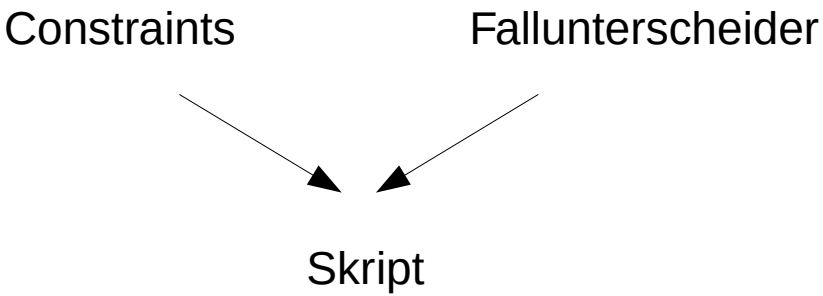






Realisierung in Alice



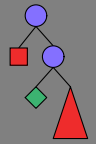


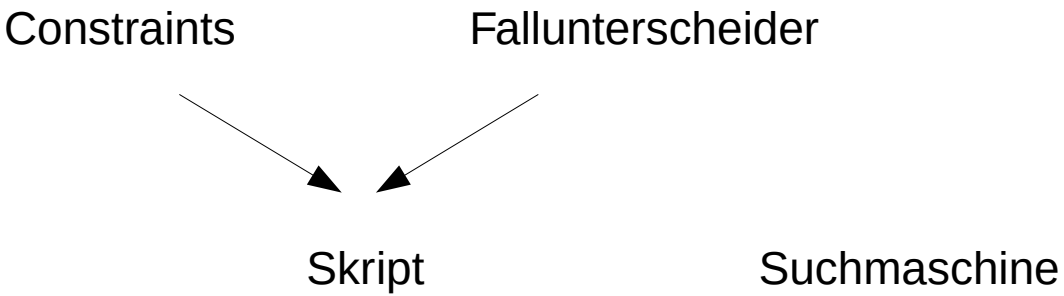




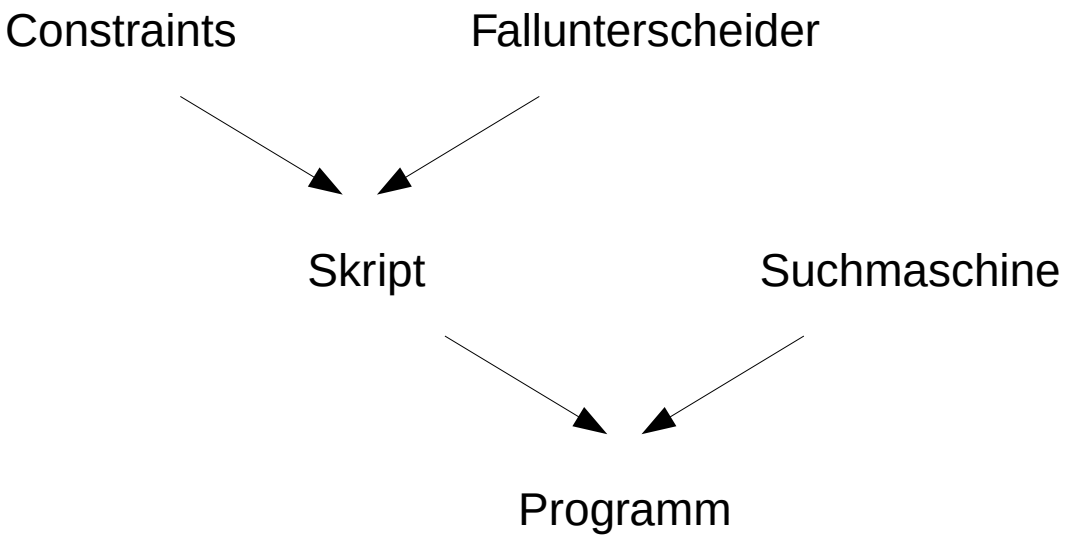


Realisierung in Alice

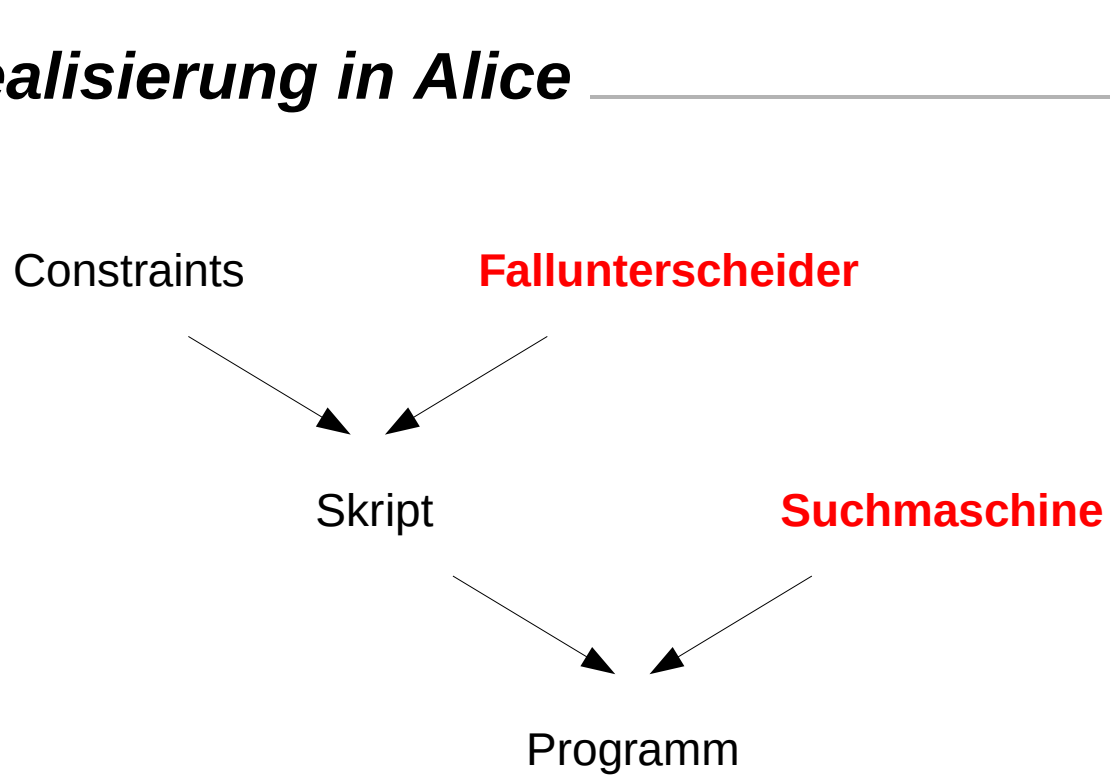




Realisierung in Alice

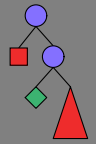


Realisierung in Alice



stellt Alice zur Verfügung

Constraints in Alice



Variablen haben den Typ `FD.fd`.

Initialisierung von Variablen z.B. mit:

```
FD.range : int * int -> FD.fd
FD.rangeVec : int * (int * int) -> FD.fd vector
```

Propagierer werden z.B. erzeugt mit:

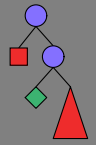
```
FD.equal : FD.fd * FD.fd -> unit
FD.plus : FD.fd * FD.fd * FD.fd -> unit
FD.distinct : FD.fd vector -> unit
```







Fallunterscheider in Alice



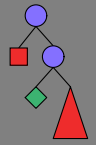
Fallunterscheider können angelegt werden mit:

```
FD.distribute :
        FD.dist_mode * FD.fd vector -> unit
```

Zu `FD.dist_mode` gehören z.B.

- `FD.NAIVE`
  Für die erste, nicht determinisierte Variable x mit einem Bereich $\{l, \ldots, u\}$ werden die Fälle $x = l$ und $x \neq l$ unterschieden.
- `FD.FIRSTFAIL`
  Wie `FD.NAIVE`, nur wird die erste Variable mit dem *kleinsten* Bereich ausgewählt.

Suchmaschinen in Alice



Es werden verschiedene Suchmaschinen zur Verfügung gestellt, z.B.

```
Search.searchOne : (unit -> 'a) -> 'a option
```

und

```
Search.searchAll : (unit -> 'a) -> 'a list
```

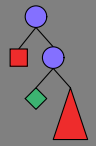
Sie finden für ein Skript die Lösung bzw. alle Lösungen, falls diese existieren.







Modellierung



Variablen

$$x \in \{1,2,3\} \quad y \in \{2,3\} \quad z \in \{1,2,3,4\}$$

Constraints

$$x < y \qquad x^2 = z$$

Fallunterscheider

First-Fail-Distribution nach dem kleinsten Wert





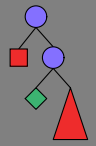


Skript

```
fun example () =
    let
       val x = FD.range (1, 3)
       val y = FD.range (2, 3)
       val z = FD.range (1, 4)
       val v = #[x, y, z]
    in
       FD.less (x, y);
       FD.times (x, x, z);
       FD.distribute (FD.FIRSTFAIL, v);
       v
    end
```

$x < y$

$x * x = z$

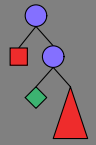








Parametrisierung



Die Lösung von parametrisierten Problemen gestaltet sich intuitiv:

- Erstellung eines parametrisierten Skriptes für eine Problem-Gruppe
- Instanzieren des einzelnen Problems durch Übergabe der Parameter

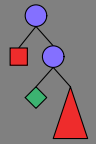
Als Beispiel betrachten wir eine abgewandelte Form des Damen-Problems, das *n-Damen-Problem*.







n-Damen-Problem: Modellierung



Variablen

Eine Variable für jede Reihe des Spielbretts:

$R_0, \ldots, R_{n-1} \in \{0, \ldots, n-1\}$

Constraints

...

Fallunterscheider

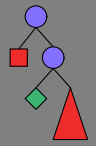
First-Fail-Distribution nach dem kleinsten Wert







n-Damen-Problem: Modellierung

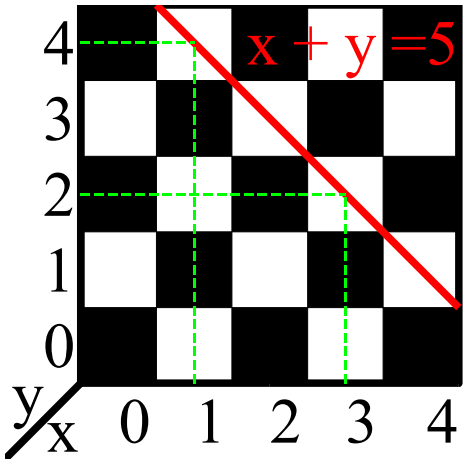


Constraints

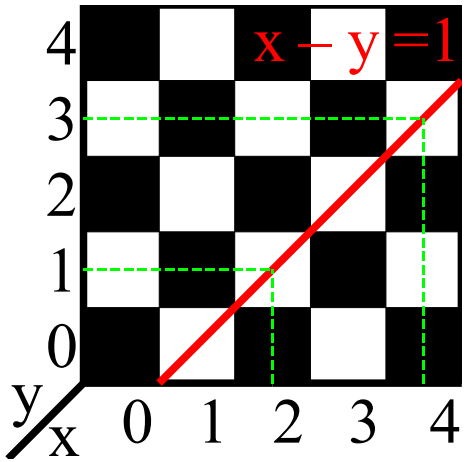
Nur eine Dame pro Spalte: $\forall i, j : R_i \neq R_j$

Nur eine Dame pro Diagonale:

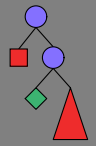
$\forall i, j : R_i + i \neq R_j + j$



$\forall i, j : R_i - i \neq R_j - j$



n-Damen-Problem: Werkzeug



Um dieses Modell umsetzen zu können benötigen wir eine neue Art von Propagierer:

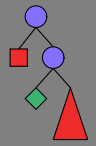
```
FD.distinctOffset : (FD.fd * int) vector -> unit
```

$$\nu = \begin{array}{|c|c|c|c|c|} \hline fd_0 & fd_1 & fd_2 & \ldots & fd_{n-1} \\ \hline i_0 & i_1 & i_2 & \ldots & i_{n-1} \\ \hline \end{array}$$

distinctOffset $v =$

distinct $(fd_0 + i_0,\ fd_1 + i_1,\ fd_2 + i_2,\ \ldots,\ fd_{n-1} + i_{n-1})$

n-Damen-Problem: Skript




```
fun nQueens n () =
    let
       val v  = FD.rangeVec (n, (0, n-1))
       val v1 = Vector.mapi (fn (i,x)=>(x, i)) v
       val v2 = Vector.mapi (fn (i,x)=>(x,~i)) v
    in
       FD.distinct v;
       FD.distinctOffset v1;
       FD.distinctOffset v2;
       FD.distribute(FD.FIRSTFAIL, v);
       v
    end
```

$R_i \neq R_j$

$R_i + i \neq R_j + j$

$R_i - i \neq R_j - j$

Literatur

[Apt03] Krzysztof R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[Sch00] Christian Schulte. *Programming Constraint Services*. Dissertation, 2000.

[SS03] Christian Schulte and Gert Smolka. *Finite Domain Constraint Programming in Oz. A Tutorial.* www.mozart-oz.org/documentation/fdt/, 2003.

[Tea03] The Alice Team. *The Alice System, Online Manual*. www.ps.uni-sb.de/alice/, 2003.