Hochschule Bielefeld
University of Applied Sciences and Arts

Constraint Programming

CPL - Kurzvortrag - 09.12.2025

Einführung

Hochschule Bielefeld
University of Applied Sciences and Arts

- deklaratives Programmierparadigma
- Programmierer definiert, **was** gelöst werden muss und nicht **wie**
- Funktionen & Zuweisungen sind gerichtet
  - velocity(d,t) = d/t
  - distance(v,t) = v*t
- stattdessen Constraints (ungerichtet)
  - v == d/t

$$v = \frac{d}{t}$$

Constraints

Hochschule Bielefeld
University of Applied Sciences and Arts

- Constraints = Bedingungen / Einschränkungen
  - z.B. $X < 5$, $A \neq B$, $X+Z=Y$
- logische Aussage, die Werte von Variablen einschränken
- Beschreibt eine Relation zwischen Variablen, keinen Rechenschritt
  - $R \subseteq D_1 \times \ldots \times D_n$
- Definiert die Menge aller erlaubten Werte
  - $X,Y \in \mathbb{N}$, $X+Y=5$, $R = \{(1,4), (2,3), (3,2), (4,1)\}$

Constraint Satisfaction Problem

Hochschule Bielefeld
University of Applied Sciences and Arts

- **V:** endliche Menge von Variablen $V = \{x_1, \dots, x_n\}$
  D: Menge von Domänen $D = \{D_1, \dots, D_n\}$ D_i ist der Wertebereich für x_i
  C: Menge von Constraints $C = \{C_1, \dots, C_m\}$

- Eine Lösung ist ein Tupel $(d_1, \dots, d_n)$ von Werten, die alle Constraints erfüllen. $d_i \in D_i$ ist der Wert für x_i

CSP: Beispiel

Hochschule Bielefeld
University of Applied Sciences and Arts

Kartenfärbungsproblem:

Färbe die Regionen Mittelitaliens (Toscana, Marche, Umbria, Lazio, Abruzzo) mit den Farben Rot, Grün oder Blau so, dass keine zwei benachbarten Regionen dieselbe Farbe erhalten.

V = {T, M, U, L, A}

D = {D_T={red, green, blue}, D_M={red, green, blue}, … , D_A={red, green, blue}}

C = {T ≠ M, T ≠ U, T ≠ L, M ≠ L, M ≠ U, M ≠ A, U ≠ L, L ≠ A}

Keine Lösung

Constraint Optimization Problem

Hochschule Bielefeld
University of Applied Sciences and Arts

- CSP mit
  - Variablen $V = \{x_1, \dots, x_n\}$
  - Domänen $D = \{D_1, \dots, D_n\}$
  - Constraints $C = \{C_1, \dots, C_m\}$
- Zusätzlich Zielfunktion $f: D_1 \times \dots \times D_n \rightarrow D$
- Lösung für ein COP ist die Lösung vom CSP, bei der die Zielfunktion maximal bzw. minimal ist

Constraint Solver

Hochschule Bielefeld
University of Applied Sciences and Arts

- sucht Werte, die alle Constraints erfüllen
- Solver ist Domänen-spezifisch
- bereitgestellt von der Host-Programmiersprache
- Typische Arbeitsschritte:
  - Constraint Propagation: verkleinern der Domänen durch bestehende Constraints
  - Backtracking / Suche: ausprobieren von verschiedenen Kombinationen
  - Konsistenzprüfung: frühzeitiges Erkennen von Widersprüchen (spart Rechenzeit)

Constraint Logic Programming

Hochschule Bielefeld
University of Applied Sciences and Arts

- CLP = logische Programmierung + Constraint-Satisfaction
- In Prolog bestehen Constraints nur aus Gleichungen (Unifikation über dem Herbrand-Universum)
- CLP ersetzt/erweitert das Herbrand-Universum durch andere Domänen
- CLP fügt zu der Unifikation einen Constraint-Solver hinzu
  - ermöglicht effiziente Lösung von Gleichungen, Ungleichungen, linearen Problemen usw.

Definition CLP

Hochschule Bielefeld
University of Applied Sciences and Arts

- Signatur (Σ, Π) gegeben
  - Σ: Menge aller Funktionssymbole und Konstanten
  - Π: Menge aller Prädikatssymbole
  - $\Pi_c \subseteq \Pi$: Menge aller Constraintsymbole (Erweiterung von Π z.B. <, ≥, usw.)

- Seien $t_1, \ldots, t_n$ Terme über Σ und $c \in \Pi_c$ ein Constraintsymbol mit Arität n, dann ist $c(t_1, \ldots, t_n)$ ein Constraint
- true und false sind Constraints
- Wenn C und D Constraints sind, dann ist $C \wedge D$ ein Constraint

Definition CLP-Programm

Hochschule Bielefeld
University of Applied Sciences and Arts

- Sei $H, A_1, \ldots, A_n.$ eine atomare Formel über $(\Sigma, \Pi / \Pi_c)$ und C ein Constraint über (Σ, Π_c)
- Eine CLP Klausel ist eine Formel der Form: $H :- C, A_1, \ldots, A_n$
- Ein CLP Programm ist eine Menge von CLP Klauseln

Definition CLP-Übergangssystem

Hochschule Bielefeld
University of Applied Sciences and Arts

- Ein Zustand ist ein Paar $\langle G;D \rangle$
  - *G:* aktuelles Ziel (noch abzuarbeitenden Atome und Constraints)
  - *D:* alle bisher gesammelten Constraints

- Ein CLP-Übergangssystem ist eine Paar $(S;\rightarrow)$
  - *S*: Menge an Zuständen
  - $\rightarrow$: Übergangsrelation

- *CT* ist eine Constraint Theorie, die die Bedeutung von Constraintsymbolen definiert (z.B. was bedeutet > oder ≠)

Definition Übergangsrelation

Hochschule Bielefeld
University of Applied Sciences and Arts

- **Unfold:**
  - Wenn eine Klausel $H :- C, A_1, \ldots, A_n$ existiert und $CT \models \exists((A = H) \wedge C)$ gilt
  - dann $\langle A \wedge G; D\rangle \rightarrow \langle G \wedge C \wedge A_1 \wedge \ldots \wedge A_n ; (A = H) \wedge D\rangle$.

- **Failure:**
  - Wenn keine solche Klausel existiert gilt
  - dann $\langle A \wedge G; D\rangle \rightarrow \langle A \wedge G; \mathit{false}\rangle$.

- **Solve:**
  - Wenn $CT \models \forall((C \wedge D_1) \leftrightarrow D_2)$ gilt
  - dann $\langle C \wedge G; D_1\rangle \rightarrow \langle G; D_2\rangle$.

Definition CLP-Berechnungen

Hochschule Bielefeld
University of Applied Sciences and Arts

- Übergangssystem $(S;\rightarrow)$ gegeben

- Anfangszustand: $\langle G; true \rangle$
- Endzustand:
  - $\langle true; C \rangle$ (Erfolgreich)
  - $\langle G; false \rangle$(Fehlgeschlagen)

- endliche Berechnung = Sequenz $S_1, \ldots, S_n$, sodass gilt $S_1 \rightarrow S_{i+1}$ mit $i \in [1, n-1]$
- S_1 ist der Anfangszustand und S_n ist einer der Endzustände

Gen. & Test vs. Constraint & Gen.

Hochschule Bielefeld
University of Applied Sciences and Arts

Gen. & Test Prinzip

```
p(X):- Y =1, X is Y + 1.
=> 2

p(X):- X is Y + 1, Y =1.
=> Arguments are not sufficiently instantiated
```

Gen. & Test vs. Constraint & Gen.

Hochschule Bielefeld
University of Applied Sciences and Arts

Gen. & Test Prinzip

- Generierung einer Menge an Variablen
- Zuweisen der Variablen
- Testen ob Variablen die Constraints erfüllen
- “brute-force” Ansatz

Problem: großer Wertebereich

Gen. & Test vs. Constraint & Gen.

Hochschule Bielefeld
University of Applied Sciences and Arts

Ansatz: Constraint & Generate

- Wertemenge durch die Constraints beschränken
- Generierung einer Menge an Variablen aus der Menge an möglichen Werten welche durch Constraints reduziert sind

Nutzen

=> Deutlich verbesserte Effizienz, da der Suchraum drastisch verkleinert wird

Gen. & Test vs. Constraint & Gen.

Hochschule Bielefeld
University of Applied Sciences and Arts

Beispiel

- Aufgabe: Lösen der Gleichung
- Jeder Buchstabe steht für eine andere Zahl

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

Constraint & Gen.

Hochschule Bielefeld
University of Applied Sciences and Arts

Beispiel: Prolog (clpfd)

```
:- use_module(library(clpfd)).

send([S,E,N,D,M,O,R,Y]) :-
        gen_domains([S,E,N,D,M,O,R,Y],0..9),
        S #\= 0, M #\= 0,
        all_distinct([S,E,N,D,M,O,R,Y]),
        1000* S + 100* E + 10*N + D
        + 1000* M + 100* O + 10*R + E
        #= 10000* M + 1000* O + 100* N + 10*E + Y,
        labeling([],[S,E,N,D,M,O,R,Y]).

        gen_domains([] ,_).
        gen_domains([H|T],D) :- H in D, gen_domains(T,D).


=> S= 9, E=5, N=6, D=7, M=1, O=0, R=8, Y=2
```

Gen. & Test

Hochschule Bielefeld
University of Applied Sciences and Arts

Beispiel: Prolog (clpfd)

```
:- use_module(library(clpfd)).

send([S,E,N,D,M,O,R,Y]) :-
        gen_domains([S,E,N,D,M,O,R,Y],0..9),
        labeling([] ,[S,E,N,D,M,O,R,Y]),
        S #\= 0, M #\= 0,
        all_distinct([S,E,N,D,M,O,R,Y]),
        1000* S + 100* E + 10*N + D
        + 1000* M + 100* O + 10*R + E
        #= 10000* M + 1000* O + 100* N + 10*E + Y.

gen_domains([],_).
gen_domains([H|T],D) :- H in D, gen_domains(T,D).
```

```
=> Time limit exceeded
```

CP/CPL Sprachen

Hochschule Bielefeld
University of Applied Sciences and Arts

- Prolog (clpfd Extension)
- MiniZinc
- ChocoSolver
  - Gradle/Maven, Python
- OR-Tools
  - C++, C#, Java, Python

MiniZinc

Hochschule Bielefeld
University of Applied Sciences and Arts

- Open-Source constraint modeling language
- Unterstützt Modellierung von CSPs und COPs
- Modelle in High Level Language
- Kompilierung nach FlatZinc

Vorteil: Problem kann unabhängig vom Solver dargestellt werden

MiniZinc

Hochschule Bielefeld
University of Applied Sciences and Arts

Beispiel

```
include " globals.mzn";

var 1..9: S;
var 0..9: E;
var 0..9: N;
var 0..9: D;
var 1..9: M;
var 0..9: O;
var 0..9: R;
var 0..9: Y;

constraint 1000 * S + 100 * E + 10 * N + D
+ 1000 * M + 100 * O + 10 * R + E
= 10000 * M + 1000 * O + 100 * N + 10 * E + Y;

constraint alldifferent ([S,E,N,D,M,O,R,Y]);

solve satisfy;

output [" \(S)\(E)\(N)\(D)\n",
"+ \(M)\(O)\(R)\(E)\n",
"= \(M)\(O)\(N)\(E)\(Y)\n"];
```

Aufbau:

- Eingebundene Libraries
- Variablen und Parameter
- Constraints
- Solve-Ansatz
  - Satisfaction / Optimization

Anwendungsgebiete

Hochschule Bielefeld
University of Applied Sciences and Arts

Wann sollte man auf CP und CLP zurückgreifen?

- Kombinationsprobleme
- KI-Anwendungen
- Bioinformatik
- Industrieanwendungen

Literatur

Gabbrielli, M., und S. Martini. 2023. *Programming Languages: Principles and Paradigms*. Springer Cham. https://doi.org/10.1007/978-3-031-34144-1.