

# Logické programování s omezujícími podmínkami

## *Constraint Logic Programming: CLP*

## CP: elektronické materiály

- Dechter, R. **Constraint Processing**. Morgan Kaufmann Publishers, 2003.
  - <http://www.ics.uci.edu/~dechter/books/materials.html>  
průsvitky ke knize
- Barták R. **Přednáška Omezující podmínky na MFF UK, Praha**.
  - <http://kti.ms.mff.cuni.cz/~bartak/podminky/index.html>
- **SICStus Prolog User's Manual**. Kapitola o CLP(FD).
  - <http://www.fi.muni.cz/~hanka/sicstus/doc/html/>
- **Příklady v distribuci SICStus Prologu**: cca 60 příkladů, zdrojový kód
  - `lib/sicstus-*/library/clpfd/examples/`

## Probírané oblasti

- Obsah
  - úvod: od LP k CLP
  - základy programování
  - základní algoritmy pro řešení problémů s omezujícími podmínkami
- Příbuzné přednášky na FI
  - PA163 Programování s omezujícími podmínkami
    - viz interaktivní osnova IS
  - PA167 Rozvrhování
    - <http://www.fi.muni.cz/~hanka/rozvrhovani>
    - zahrnutý CP techniky pro řešení rozvrhovacích problémů

## Historie a současnost

- **1963** interaktivní grafika (Sutherland: Sketchpad)
- **Polovina 80. let**: logické programování omezujícími podmínkami
- **Od 1990**: komerční využití
- Už v roce **1996**: výnos řádově stovky milionů dolarů
- Aplikace – příklady
  - **Lufthansa**: krátkodobé personální plánování
    - reakce na změny při dopravě (zpoždění letadla, ...)
    - minimalizace změny v rozvrhu, minimalizace ceny
  - **Nokia**: automatická konfigurace sw pro mobilní telefony
  - **Renault**: krátkodobé plánování výroby, funkční od roku 1995

## Omezení (*constraint*)

- Dána
  - množina (**doménových**) **proměnných**  $Y = \{y_1, \dots, y_k\}$
  - **konečná** množina hodnot (**doména**)  $D = \{D_1, \dots, D_k\}$

**Omezení**  $c$  na  $Y$  je podmnožina  $D_1 \times \dots \times D_k$

- omezuje hodnoty, kterých mohou proměnné nabývat současně
- Příklad:
  - proměnné: A,B
  - domény:  $\{0,1\}$  pro A  $\{1,2\}$  pro B
  - omezení:  $A \neq B$  nebo  $(A,B) \in \{(0,1), (0,2), (1,2)\}$
- Omezení  $c$  definováno na  $y_1, \dots, y_k$  je **splněno**, pokud pro  $d_1 \in D_1, \dots, d_k \in D_k$  platí  $(d_1, \dots, d_k) \in c$ 
  - příklad (pokračování): omezení splněno pro (0,1), (0,2), (1,2), není splněno pro (1,1)

## Problém splňování podmínek (CSP)

- Dána
  - konečná množina **proměnných**  $X = \{x_1, \dots, x_n\}$
  - konečná množina hodnot (**doména**)  $D = \{D_1, \dots, D_n\}$
  - konečná množina **omezení**  $C = \{c_1, \dots, c_m\}$ 
    - omezení je definováno na podmnožině  $X$

**Problém splňování podmínek** je trojice  $(X, D, C)$   
**(constraint satisfaction problem)**

- Příklad:
  - proměnné: A,B,C
  - domény:  $\{0,1\}$  pro A  $\{1,2\}$  pro B  $\{0,2\}$  pro C
  - omezení:  $A \neq B, B \neq C$

## Řešení CSP

- **Částečné ohodnocení proměnných**  $(d_1, \dots, d_k), k < n$ 
  - některé proměnné mají přiřazenu hodnotu
- **Úplné ohodnocení proměnných**  $(d_1, \dots, d_n)$ 
  - všechny proměnné mají přiřazenu hodnotu
- **Řešení CSP**
  - úplné ohodnocení proměnných, které splňuje všechna omezení
  - $(d_1, \dots, d_n) \in D_1 \times \dots \times D_n$  je **řešení**  $(X, D, C)$ 
    - pro každé  $c_i \in C$  na  $x_{i_1}, \dots, x_{i_k}$  platí  $(d_{i_1}, \dots, d_{i_k}) \in c_i$
- Hledáme: jedno nebo
  - všetchna řešení nebo
  - optimální řešení (vzhledem k objektivní funkci)

## Příklad: jednoduchý školní rozvrh

- **proměnné:** Jan, Petr, ...
- **domény:**  $\{3, 4, 5, 6\}, \{3, 4\}, \dots$
- **omezení:** `all_distinct([Jan, Petr, ...])`
- **částečné ohodnocení:** Jan=6, Anna=5, Marie=1
- **úplné ohodnocení:**  
Jan=6, Petr=3, Anna=5, Ota=2, Eva=4, **Marie=6**
- **řešení CSP:**  
Jan=6, Petr=3, Anna=5, Ota=2, Eva=4, Marie=1
- **všetchna řešení:** ještě Jan=6, **Petr=4**, Anna=5, Ota=2, **Eva=3**, Marie=1
- **optimalizace:** ženy učí co nejdříve  
Anna+Eva+Marie  $\neq$  Cena minimalizace hodnoty proměnné Cena
- **optimální řešení:** Jan=6, **Petr=4**, Anna=5, Ota=2, **Eva=3**, Marie=1

učitel	min	max
Jan	3	6
Petr	3	4
Anna	2	5
Ota	2	4
Eva	3	4
Marie	1	6

## CLP(*FD*) program

### ▪ Základní struktura CLP programu

1. definice proměnných a jejich domén
2. definice omezení
3. hledání řešení

### ▪ (1) a (2) deklarativní část

- **modelování** problému
- vyjádření problému splňování podmínek

### ▪ (3) řídicí část

- **prohledávání** stavového prostoru řešení
- procedura pro hledání řešení (enumeraci) se nazývá **labeling**
- umožní nalézt jedno, všechna nebo optimální řešení

## Kód CLP(*FD*) programu

% základní struktura CLP programu

```
solve( Variables ) :-  
    declare_variables( Variables ),          domain([Jan],3,6), ...  
    post_constraints( Variables ),          all_distinct([Jan,Petr,...])  
    labeling( Variables ).
```

% triviální labeling

```
labeling( [] ).
```

```
labeling( [Var|Rest] ) :-
```

```
    fd_min(Var,Min),                        % výběr nejmenší hodnoty z domény  
    ( Var#=Min, labeling( Rest )  
    ;  
      Var#>Min, labeling( [Var|Rest] )  
    ).
```

## Příklad: algebrogram

▪ Přiřad'te cifry 0, ... 9 písmenům S, E, N, D, M, O, R, Y tak, aby platilo:

- SEND + MORE = MONEY
- různá písmena mají přiřazena různé cifry
- S a M nejsou 0

▪ `domain([E,N,D,O,R,Y], 0, 9), domain([S,M],1,9)`

```
1000*S + 100*E + 10*N + D  
+  
1000*M + 100*O + 10*R + E  
#= 10000*M + 1000*O + 100*N + 10*E + Y
```

▪ `all_distinct( [S,E,N,D,M,O,R,Y] )`

▪ `labeling( [S,E,N,D,M,O,R,Y] )`

## Od LP k CLP I.

▪ CLP: rozšíření logického programování o omezující podmínky

▪ CLP systémy se liší podle typu domény

- **CLP(*A*)** generický jazyk
- **CLP(*FD*)** domény proměnných jsou konečné (*Finite Domains*)
- **CLP(*R*)** doménou proměnných je množina reálných čísel

▪ Cíl

- využít syntaktické a výrazové přednosti LP
- dosáhnout větší efektivity

▪ **Unifikace v LP je nahrazena splňováním podmínek**

- unifikace se chápe jako **jedna** z podmínek
- $A = B$
- $A \#< B, A \text{ in } 0..9, \text{ domain}([A,B],0,9), \text{ all\_distinct}([A,B,C])$

## Od LP k CLP II.

- Pro řešení podmínek se používají **konzistenční techniky**
  - *consistency techniques, propagace omezení (constraint propagation)*
  - omezení:  $A \text{ in } 0..2, B \text{ in } 0..2, B \#< A$   
domény po propagaci omezení  $B \#< A$ :  $A \text{ in } 1..2, B \text{ in } 0..1$
- Podmínky jsou deterministicky vyhodnoceny v okamžiku volání podmínky
- **Prohledávání doplněno konzistenčními technikami**
  - $A \text{ in } 1..2, B \text{ in } 0..1, B \#< A$
  - po provedení  $A \# = 1$  se z  $B \#< A$  se odvodí:  $B \# = 0$
- **Podmínky jako výstup**
  - kompaktní reprezentace nekonečného počtu řešení, výstup lze použít jako vstup
  - dotaz:  $A \text{ in } 0..2, B \text{ in } 0..2, B \#< A$   
výstup:  $A \text{ in } 1..2, B \text{ in } 0..1, B \#< A$

## Syntaxe CLP

- Výběr jazyka omezení
  - CLP klauzule  
jako LP klauzule, ale její tělo může obsahovat omezení daného jazyka  
 $p(X,Y) :- X \#< Y+1, q(X), r(X,Y,Z).$
  - Rezoluční krok v LP
    - kontrola existence nejobecnějšího unifikátoru (MGU) mezi cílem a hlavou
  - Krok odvození v CLP také zahrnuje
    - kontrola konzistence aktuální množiny omezení s omezeními v těle klauzule
- ⇒ Vyvolání dvou řešičů: unifikace + řešič omezení

## Operační sémantika CLP

- CLP výpočet cíle  $G$ 
  - *Store* množina aktivních omezení  $\equiv$  **prostor omezení (constraint store)**
  - inicializace  $Store = \emptyset$
  - seznamy cílů v  $G$  prováděny v obvyklém pořadí
  - pokud narazíme na cíl s omezením  $c$ :  $NewStore = Store \cup \{c\}$
  - snažíme se splnit  $c$  vyvoláním jeho řešiče
    - při neúspěchu se vyvolá backtracking
    - při úspěchu se podmínky v  $NewStore$  zjednoduší propagací omezení
  - zbývající cíle jsou prováděny s upraveným  $NewStore$
- CLP výpočet cíle  $G$  je úspěšný, pokud se dostaneme z iniciálního stavu  $\langle G, \emptyset \rangle$  do stavu  $\langle G', Store \rangle$ , kde  $G'$  je prázdný cíl a  $Store$  je splnitelná.

## CLP(FD) v SICStus Prologu

# Systemy a jazyky pro CP

- **IBM ILOG CP** 1987
  - omezující podmínky v C++, Java nebo generickém modelovacím jazyku OPL
  - implementace podmínek založena na objektově orientovaném programování
  - špičkový komerční sw, vznikl ve Francii, nedávno zakoupen IBM
  - nyní nově volně dostupný pro akademické použití
- **Swedish Institute of Computer Science: SICStus Prolog** 1985
  - silná CLP(FD) knihovna, komerční i akademické použití
- **IC-PARC, Imperial College London, Cisco Systems: ECLiPSe** 1984
  - široké možnosti kooperace mezi různými řešiči: konečné domény, reálná čísla, repair
  - od 2004 vlastní Cisco Systems volně dostupné pro akademické použití, rozvoj na IC-PARC, platformy: Windows, Linux, Solaris
- Mnoho dalších systémů: Choco, Gecode, Minion, Oz, SWI Prolog, ...

# CLP(FD) v SICStus Prologu

- Vestavěné predikáty jsou dostupné v separátním modulu (knihovně) :- use\_module(library(clpfd)).
- Obecné principy platné všude nicméně standardy jsou nedostatečné
  - stejné/podobné vestavěné predikáty existují i jinde
  - CLP knihovny v SWI Prologu i ECLiPSe se liší

## Příslušnost k doméně: Range termy

- `?- domain( [A,B], 1,3).` `domain(+Variables, +Min, +Max)`  
`A in 1..3`  
`B in 1..3`
- `?- A in 1..8, A #\= 4.` `?X in +Min..+Max`  
`A in (1..3) \ (5..8)`
- Doména reprezentována jako posloupnost intervalů celých čísel
- `?- A in (1..3) \ (8..15) \ (5..9) \ {100}.` `?X in +Range`  
`A in (1..3) \ (5..15) \ {100}`
- Zjištění domény Range proměnné Var: `fd_dom(?Var, ?Range)`
  - `A in 1..8, A #\= 4, fd_dom(A, Range).` `Range=(1..3) \ (5..8)`
  - `A in 2..10, fd_dom(A, (1..3) \ (5..8)).` `no`
- Range term: reprezentace nezávislá na implementaci

## Příslušnost k doméně: FDSet termy

- FDSet term: reprezentace závislá na implementaci
- `?- A in 1..8, A #\= 4, fd_set(A, FDSet).` `fd_set(?Var, ?FDSet)`  
`A in (1..3) \ (5..8)`  
`FDSet = [[1|3], [5|8]]`
- `?- A in 1..8, A #\= 4, fd_set(A, FDSet), B in_set FDSet.` `?X in_set +FDSet`  
`A in (1..3) \ (5..8)`  
`FDSet = [[1|3], [5|8]]`  
`B in (1..3) \ (5..8)`
- FDSet termy představují nízko-úrovňovou implementaci
- FDSet termy nedoporučeny v programech
  - používat pouze predikáty pro manipulaci s nimi
  - omezit použití `A in_set [[1|2], [6|9]]`
- Range termy preferovány

## Další fd... predikáty

- `fdset_to_list(+FDset, -List)` vrací do seznamu prvky FDset
- `list_to_fdset(+List, -FDset)` vrací FDset odpovídající seznamu
- `fd_var(?Var)` je Var doménová proměnná?
- `fd_min(?Var, ?Min)` nejmenší hodnota v doméně
- `fd_max(?Var, ?Max)` největší hodnota v doméně
- `fd_size(?Var, ?Size)` velikost domény
- `fd_degree(?Var, ?Degree)` počet navázaných omezení na proměnné
  - mění se během výpočtu: pouze aktivní omezení, i odvozená aktivní omezení

## Aritmetická omezení

- `Expr RelOp Expr`      `RelOp -> #= | #\= | #< | #=< | #> | #>=`
  - `A + B #=< 3, A #\= (C - 4) * (D - 5), A/2 #= 4`
  - POZOR: neplést `#=< a #>=` s operátory pro implikaci: `#<=` `#=>`
- `sum(Variables, RelOp, Suma)`
  - `domain([A,B,C,F],1,3), sum([A,B,C],#=,F)`
  - `Variables` i `Suma` musí být doménové proměnné nebo celá čísla
- `scalar_product(Coeffs, Variables, RelOp, ScalarProduct)`
  - `domain([A,B,C,F],1,6), scalar_product([1,2,3],[A,B,C],#=,F)`
  - `Variables` i `Value` musí být doménové proměnné nebo celá čísla, `Coeffs` jsou celá čísla
  - POZOR na pořadí argumentů, nejprve jsou celočíselné koeficienty, pak dom. proměnné
  - `scalar_product(Coeffs, Variables, #=, Value, [consistency(domain)])`
    - silnější typ konzistence
    - POZOR: domény musí mít konečné hranice

## Základní globální omezení

- `all_distinct(List)`
  - všechny proměnné různé
- `cumulative(...)`
  - disjunktivní a kumulativní rozvrhování
- `cumulatives(...)`
  - kumulativní rozvrhování na více zdrojů

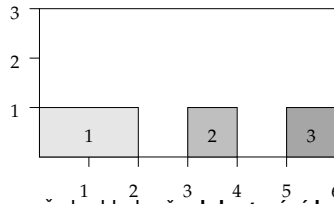
## Všechny proměnné různé

- `all_distinct(Variables), all_different(Variables)`
- Proměnné v seznamu `Variables` jsou různé
- `all_distinct` a `all_different` se liší úrovní propagace
  - `all_distinct` má úplnou propagaci
  - `all_different` má slabší (neúplnou) propagaci
- Příklad: učitelé musí učit v různé hodiny
  - `all_distinct([Jan,Petr,Anna,Ota,Eva,Marie])`  
`Jan = 6, Ota = 2, Anna = 5,`  
`Marie = 1, Petr in 3..4, Eva in 3..4`
  - `all_different([Jan,Petr,Anna,Ota,Eva,Marie])`  
`Jan in 3..6, Petr in 3..4, Anna in 2..5,`  
`Ota in 2..4, Eva in 3..4, Marie in 1..6`

učitel	min	max
Jan	3	6
Petr	3	4
Anna	2	5
Ota	2	4
Eva	3	4
Marie	1	6

## Disjunktivní rozvrhování (unární zdroj)

- `cumulative([task(Start, Duration, End, 1, Id) | Tasks])`
- Rozvržení úloh zadaných startovním a koncovým časem (Start,End), dobou trvání (nezáporné Duration) a identifikátorem (Id) tak, aby se nepřekrývaly
  - příklad s konstantami: `cumulative([task(0,2,2,1,1), task(3,1,4,1,2), task(5,1,6,1,3)])`



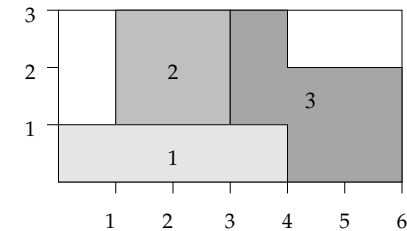
- příklad: vytvoření rozvrhu, za předpokladu, že **doba trvání hodin není stejná**  
`JanE#= Jan+1, PetrE#= Petr+1, AnnaE#= Anna+2,`  
`cumulative(task(Jan,1,JanE,1,1), task(Petr,1,PetrE,1,2), task(Anna,1,AnnaE,1,`  
`task(Ota,2,OtaE,1,4), task(Eva,2,EvaE,1,5), task(Marie,3,MarieE,1,6)])`

## Kumulativní rozvrhování

- `cumulative([task(Start,Duration,End,Demand,TaskId) | Tasks], [limit(Limit)])`
- Rozvržení úloh zadaných startovním a koncovým časem (Start,End), dobou trvání (nezáporné Duration), požadovanou kapacitou zdroje (Demand) a identifikátorem (Id) tak, aby se nepřekrývaly a aby celková kapacita zdroje nikdy nepřekročila Limit

- Příklad s konstantami:

`cumulative([task(0,4,4,1,1), task(1,2,3,2,2), task(3,3,6,2,3), task(4,2,6,1,4)], [limit(3)])`



## Kumulativní rozvrhování s více zdroji

- Rozvržení úloh tak, aby se nepřekrývaly a daná kapacita zdrojů nebyla překročena (limit zdroje chápán jako horní mez – `bound(upper)`)
- `cumulatives([task(Start,Duration,End,Demand,MachineId) | Tasks], [machine(Id,Limit) | Machines], [bound(upper)])`
- Úlohy zadány startovním a koncovým časem (Start,End), dobou trvání (nezáporné Duration), požadovanou kapacitou zdroje (Demand) a požadovaným typem zdroje (MachineId)
- Zdroje zadány identifikátorem (Id) a kapacitou (Limit)
- Příklad:  
`?- domain([B,C],1,2),`  
`cumulatives([task(0,4,4,1,1), task(3,1,4,1,B), task(5,1,6,1,C)],`  
`[machine(1,1), machine(2,1)],`  
`[bound(upper)]).` `C in 1..2, B=2`