

**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/ics-275a/fall-2001/readings.html>
- Barták R. **Přednáška Omezující podmínky na MFF UK, Praha**.
 - <http://kti.ms.mff.cuni.cz/~bartak/podminky/prednaska.html>
- **SICStus Prolog User's Manual**, 2004. Kapitola o CLP(FD).
 - <http://www.fi.muni.cz/~hanka/sicstus/doc/html/>
- **Příklady v distribuci SICStus Prologu**: cca 25 příkladů, zdrojový kód
 - <aia:/software/sicstus-3.10.1/lib/sicstus-3.10.1/library/clpfd/examples/>
- **Constraint Programming Online**
 - <http://slash.math.unipd.it/cp/index.php>

Probírané oblasti

- Obsah
 - úvod: od LP k CLP
 - základy programování
 - základní algoritmy pro řešení problémů s omezujícími podmínkami

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

- <http://www.fi.muni.cz/~hanka/cp>

- PA167 Rozvrhování

- <http://www.fi.muni.cz/~hanka/rozvrhovani>

- zahrnuty 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

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 (*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 $(d_1, \dots, d_k) \in c$

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)

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

- **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 $(d_{i_1}, \dots, d_{i_k}) \in c_i$

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
všechna řešení nebo
optimální řešení (vzhledem k objektivní funkci)

Příklad: jednoduchý školní rozvrh

- **proměnné:** Jan, Petr, ...
- **domény:** {3, 4, 5, 6}, {3, 4}, ...
- **omezení:** `all_distinct([Jan, Petr, ...])`

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

Příklad: jednoduchý školní rozvrh

- **proměnné:** Jan, Petr, ...
- **domény:** {3, 4, 5, 6}, {3, 4}, ...
- **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**

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

Příklad: jednoduchý školní rozvrh

- **proměnné:** Jan, Petr, ...
- **domény:** {3, 4, 5, 6}, {3, 4}, ...
- **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

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

- **všetchna řešení:** ještě Jan=6, **Petr=4**, Anna=5, Ota=2, **Eva=3**, Marie=1

Příklad: jednoduchý školní rozvrh

- **proměnné:** Jan, Petr, ...
- **domény:** {3, 4, 5, 6}, {3, 4}, ...
- **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
- **všechna řešení:** ještě Jan=6, **Petr=4**, Anna=5, Ota=2, **Eva=3**, Marie=1
- **optimalizace:** ženy učí co nejdříve

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

Příklad: jednoduchý školní rozvrh

- **proměnné:** Jan, Petr, ...
- **domény:** {3, 4, 5, 6}, {3, 4}, ...
- **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

- **všechna 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

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

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 každého CLP programu

```
solve( Variables ) :-
```

```
    declare_variables( Variables ),          domain([Jan],3,6), ...
```

Kód CLP(*FD*) programu

% základní struktura každého CLP programu

```
solve( Variables ) :-
```

```
    declare_variables( Variables ),
```

```
    post_constraints( Variables ),
```

```
    domain([Jan],3,6), ...
```

```
    all_distinct([Jan,Petr,...])
```


Kód CLP(*FD*) programu

% základní struktura každého CLP programu

```
solve( Variables ) :-
```

```
    declare_variables( Variables ),
```

```
    post_constraints( Variables ),
```

```
    labeling( Variables ).
```

```
domain([Jan],3,6), ...
```

```
all_distinct([Jan,Petr,...])
```

Kód CLP(*FD*) programu

% základní struktura každého CLP programu

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

% triviální labeling

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

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

```
    fd_min(Var,Min),
```

```
    ( Var#=Min, labeling( Rest )
```

% výběr nejmenší hodnoty z domény

Kód CLP(*FD*) programu

% základní struktura každého CLP programu

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

% triviální labeling

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

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

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

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

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
- $\text{domain}([E,N,D,O,R,Y], 0, 9), \text{domain}([S,M], 1, 9)$

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

● $\text{domain}([E,N,D,O,R,Y], 0, 9), \text{domain}([S,M], 1, 9)$

$$\begin{array}{r} \bullet \\ + \\ \# = \end{array} \begin{array}{r} 1000*S + 100*E + 10*N + D \\ 1000*M + 100*O + 10*R + E \\ 10000*M + 1000*O + 100*N + 10*E + Y \end{array}$$

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

● $\text{domain}([E,N,D,O,R,Y], 0, 9), \text{domain}([S,M], 1, 9)$

●
$$\begin{array}{r} 1000*S + 100*E + 10*N + D \\ + \quad 1000*M + 100*O + 10*R + E \\ \# = 10000*M + 1000*O + 100*N + 10*E + Y \end{array}$$

● $\text{all_distinct}([S,E,N,D,M,O,R,Y])$

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

● $\text{domain}([E,N,D,O,R,Y], 0, 9), \text{domain}([S,M], 1, 9)$

●
$$\begin{array}{r} 1000*S + 100*E + 10*N + D \\ + \quad 1000*M + 100*O + 10*R + E \\ \# = 10000*M + 1000*O + 100*N + 10*E + Y \end{array}$$

● $\text{all_distinct}([S,E,N,D,M,O,R,Y])$

● $\text{labeling}([S,E,N,D,M,O,R,Y])$

Od LP k CLP I.

- CLP: rozšíření logického programování o omezující podmínky
- CLP systémy se liší podle typu domény
 - $\text{CLP}(\mathcal{A})$ generický jazyk
 - $\text{CLP}(FD)$ domény proměnných jsou konečné (*Finite Domains*)
 - $\text{CLP}(\mathbb{R})$ doménou proměnných je množina reálných čísel

Od LP k CLP I.

- CLP: rozšíření logického programování o omezující podmínky
- CLP systémy se liší podle typu domény
 - $\text{CLP}(\mathcal{A})$ generický jazyk
 - $\text{CLP}(FD)$ domény proměnných jsou konečné (*Finite Domains*)
 - $\text{CLP}(\mathbb{R})$ doménou proměnných je množina reálných čísel
- Cíl
 - využít syntaktické a výrazové přednosti LP
 - dosáhnout větší efektivity

Od LP k CLP I.

- CLP: rozšíření logického programování o omezující podmínky
- CLP systémy se liší podle typu domény
 - $\text{CLP}(\mathcal{A})$ generický jazyk
 - $\text{CLP}(FD)$ domény proměnných jsou konečné (*Finite Domains*)
 - $\text{CLP}(\mathbb{R})$ 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
 - $A = B$
 - $A \# < B, \quad A \text{ in } 0..9, \quad \text{domain}([A,B],0,9), \quad \text{all_distinct}([A,B,C])$

Od LP k CLP II.

- Pro řešení podmínek se používají **konzistenční techniky**
 - *consistency techniques, propagace omezení (constraint propagation)*
 - omezení: $A \text{ in } 0..2, B \text{ in } 0..2, B \neq A$

Od LP k CLP II.

- Pro řešení podmínek se používají **konzistenční techniky**
 - *consistency techniques, propagace omezení (constraint propagation)*
 - omezení: $A \in 0..2, B \in 0..2, B \neq A$
domény po propagaci omezení $B \neq A$: $A \in 1..2, B \in 0..1$

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

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**
 - $A \text{ in } 1..2, B \text{ in } 0..1, B \#< A$

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: $B \# = 0$

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

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: $A \text{ in } 1..2, B \text{ in } 0..1,$

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 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

Nejpoužívanější systémy a jazyky pro CP

- **Swedish Institute of Computer Science: SICStus Prolog** 1985
 - silná CLP(*FD*) knihovna, komerční i akademické použití pro širokou škálu platforem

Nejpoužívanější systémy a jazyky pro CP

- **Swedish Institute of Computer Science: SICStus Prolog** 1985
 - silná CLP(*FD*) knihovna, komerční i akademické použití pro širokou škálu platforem
- **IC-PARC, Imperial College London, Cisco Systems: ECLⁱPS^e** 1984
 - široké možnosti kooperace mezi různými řešičemi: konečné domény, reálná čísla, repair
 - od 2004 vlastní Cisco Systems volně dostupné pro akademické použití, rozvoj na IC-PARC, platformy: Windows, Linux, Solaris

Nejpoužívanější systémy a jazyky pro CP

- **Swedish Institute of Computer Science: SICStus Prolog** 1985
 - silná CLP(*FD*) knihovna, komerční i akademické použití pro širokou škálu platforem
- **IC-PARC, Imperial College London, Cisco Systems: ECLⁱPS^e** 1984
 - široké možnosti kooperace mezi různými řešičemi: konečné domény, reálná čísla, repair
 - od 2004 vlastní Cisco Systems volně dostupné pro akademické použití, rozvoj na IC-PARC, platformy: Windows, Linux, Solaris
- **ILOG, omezující podmínky v C++** 1987
 - komerční společnost, vznik ve Francii, nyní rozšířená po celém světě
 - implementace podmínek založena na objektově orientovaném programování

Nejpoužívanější systémy a jazyky pro CP

- **Swedish Institute of Computer Science: SICStus Prolog** 1985
 - silná CLP(*FD*) knihovna, komerční i akademické použití pro širokou škálu platforem
- **IC-PARC, Imperial College London, Cisco Systems: ECLⁱPS^e** 1984
 - široké možnosti kooperace mezi různými řešičemi: konečné domény, reálná čísla, repair
 - od 2004 vlastní Cisco Systems volně dostupné pro akademické použití, rozvoj na IC-PARC, platformy: Windows, Linux, Solaris
- **ILOG, omezující podmínky v C++** 1987
 - komerční společnost, vznik ve Francii, nyní rozšířená po celém světě
 - implementace podmínek založena na objektově orientovaném programování
- <http://www.fi.muni.cz/~hanka/bookmarks.html#tools>
 - cca 50 odkazů na nejrůznější systémy: Prolog, C++, Java, Lisp, ...
 - COSYTEC: CHIP, PrologIA: Prolog IV, Siemens: IF Prolog,
akademický: Mozart (objektově orientované, deklarativní programování, *concurrency*), ...

CLP(*FD*) v SICStus Prologu

- CLP není dostupné v SWI Prologu
- CLP knihovna v ECLiPSe se liší
- Vestavěné predikáty jsou dostupné v separátním modulu (knihovně)
`:- use_module(library(clpfd)).`
- Obecné principy platné všude
- Stejné/podobné vestavěné predikáty existují i jinde

Příslušnost k doméně: Range terms

● ?- domain([A,B], 1,3).

A in 1..3

B in 1..3

domain(+Variables, +Min, +Max)

Příslušnost k doméně: Range terms

● ?- domain([A,B], 1,3).

A in 1..3

B in 1..3

domain(+Variables, +Min, +Max)

● ?- A in 1..8, A #\= 4.

A in (1..3) \/ (5..8)

?X in +Min..+Max

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}

Příslušnost k doméně: Range terms

- `?- 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)`

Příslušnost k doméně: Range terms

- `?- 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

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]]`

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)`

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

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 \rightarrow #= | #\= | #< | #=< | #> | #>=

• A + B #=< 3,

A #\= (C - 4) * (D - 5), A/2 #= 4

Aritmetická omezení

• Expr RelOp Expr

RelOp \rightarrow #= | #\= | #< | #=< | #> | #>=

• A + B #=< 3,

A #\= (C - 4) * (D - 5), A/2 #= 4

• sum(Variables, RelOp, Suma)

• domain([A,B,C,F], 1, 3),

sum([A,B,C], #=, F)

Aritmetická omezení

Expr RelOp Expr

RelOp \rightarrow $\# =$ | $\# \backslash =$ | $\# <$ | $\# = <$ | $\# >$ | $\# > =$

$A + B \# = < 3,$

$A \# \backslash = (C - 4) * (D - 5), A/2 \# = 4$

sum(Variables, RelOp, Suma)

$\text{domain}([A, B, C, F], 1, 3),$

$\text{sum}([A, B, C], \# = , F)$

scalar_product(Coeffs, Variables, RelOp, ScalarProduct)

$\text{domain}([A, B, C, F], 1, 6),$

$\text{scalar_product}([1, 2, 3], [A, B, C], \# = , F)$

Výroková omezení, reifikace

● Výroková omezení

pozor na efektivitu

● $\backslash\#$ negace, $\#\backslash$ konjunkce, $\#\backslash/$ disjunkce, $\#\<=>$ ekvivalence, ...

● $X \#\> 4 \#\backslash Y \#\< 6$

Výroková omezení, reifikace

● Výroková omezení

pozor na efektivitu

● \neg negace, \wedge konjunkce, \vee disjunkce, \Leftrightarrow ekvivalence, ...

● $X > 4 \wedge Y < 6$

● za předpokladu $A \text{ in } 1..4$:

$A = 3, A = 4$

$A = 3 \wedge A = 4$

$A = 1 \vee A = 2$

$A \text{ in } (\text{inf}..2) \vee (5..sup)$

$A \text{ in } (\text{inf}..2) \vee (5..sup)$

$A \text{ in } \text{inf}..sup$

Výroková omezení, reifikace

● Výroková omezení

pozor na efektivitu

● \neg negace, \wedge konjunkce, \vee disjunkce, \Leftrightarrow ekvivalence, ...

● $X > 4 \wedge Y < 6$

● za předpokladu $A \in 1..4$:

$A = 3, A = 4$

$A = 3 \wedge A = 4$

$A = 1 \vee A = 2$

$A \in (\text{inf}..2) \vee (5..\text{sup})$

$A \in (\text{inf}..2) \vee (5..\text{sup})$

$A \in \text{inf}..\text{sup}$

tj. propagace disjunkce $A = 1 \vee A = 2$ je příliš slabá (propagační algoritmy příliš obecné)

Výroková omezení, reifikace

● Výroková omezení

pozor na efektivitu

● $\backslash\#$ negace, $\#\backslash$ konjunkce, $\#\backslash/$ disjunkce, $\#\<=>$ ekvivalence, ...

● $X \#\> 4 \#\backslash Y \#\< 6$

● za předpokladu $A \text{ in } 1..4$:

$A\#\backslash= 3, A\#\backslash= 4$

$A\#\backslash= 3 \#\backslash A\#\backslash= 4$

$A\#=1 \#\backslash/ A\#=2$

$A \text{ in } (\text{inf}..2)\backslash/ (5..sup)$

$A \text{ in } (\text{inf}..2)\backslash/ (5..sup)$

$A \text{ in } \text{inf}..sup$

tj. propagace disjunkce $A\#=1 \#\backslash/ A\#=2$ je příliš slabá (propagační algoritmy příliš obecné)

● Reifikace

pozor na efektivitu

● `Constraint #<=> Bool`

Bool in 0..1 v závislosti na tom, zda je Constraint splněn

● příklad: `A in 1..10 #<=> Bool`

Výroková omezení, reifikace

● Výroková omezení

pozor na efektivitu

● $\backslash\#$ negace, $\#\backslash$ konjunkce, $\#\backslash/$ disjunkce, $\#\<=>$ ekvivalence, ...

● $X \#\> 4 \#\backslash Y \#\< 6$

● za předpokladu $A \text{ in } 1..4$:

$A\#\backslash= 3, A\#\backslash= 4$

$A\#\backslash= 3 \#\backslash A\#\backslash= 4$

$A\#=1 \#\backslash/ A\#=2$

$A \text{ in } (\text{inf}..2)\backslash/ (5..sup)$

$A \text{ in } (\text{inf}..2)\backslash/ (5..sup)$

$A \text{ in } \text{inf}..sup$

tj. propagace disjunkce $A\#=1 \#\backslash/ A\#=2$ je příliš slabá (propagační algoritmy příliš obecné)

● Reifikace

pozor na efektivitu

● $\text{Constraint} \#\<=> \text{Bool}$

$\text{Bool} \text{ in } 0..1$ v závislosti na tom, zda je Constraint splněn

● příklad: $A \text{ in } 1..10 \#\<=> \text{Bool}$

● za předpokladu $X \text{ in } 3..10, Y \text{ in } 1..4, \text{Bool} \text{ in } 0..1$

porovnej rozdíl mezi $X\#\<Y$

$X\#\<Y \#\<=> \text{Bool}$

Výroková omezení, reifikace

● Výroková omezení

pozor na efektivitu

● $\backslash\#$ negace, $\#\backslash$ konjunkce, $\#\backslash/$ disjunkce, $\#\<=>$ ekvivalence, ...

● $X \#\> 4 \#\backslash Y \#\< 6$

● za předpokladu $A \text{ in } 1..4$:

$A\#\backslash= 3, A\#\backslash= 4$

$A\#\backslash= 3 \#\backslash A\#\backslash= 4$

$A\#=1 \#\backslash/ A\#=2$

$A \text{ in } (\text{inf}..2)\backslash/ (5..sup)$

$A \text{ in } (\text{inf}..2)\backslash/ (5..sup)$

$A \text{ in } \text{inf}..sup$

tj. propagace disjunkce $A\#=1 \#\backslash/ A\#=2$ je příliš slabá (propagační algoritmy příliš obecné)

● Reifikace

pozor na efektivitu

● $\text{Constraint} \#\<=> \text{Bool}$

$\text{Bool} \text{ in } 0..1$ v závislosti na tom, zda je Constraint splněn

● příklad: $A \text{ in } 1..10 \#\<=> \text{Bool}$

● za předpokladu $X \text{ in } 3..10, Y \text{ in } 1..4, \text{Bool} \text{ in } 0..1$

porovnej rozdíl mezi $X\#\<Y$

$X\#\<Y \#\<=> \text{Bool}$

$X = 3, Y = 4$

$X \text{ in } 3..10, Y \text{ in } 1..4, \text{Bool} \text{ in } 0..1$

Příklad: reifikace

- Přesně N prvků v seznamu S je rovno X : `exactly(X, S, N)`

Příklad: reifikace

● Přesně N prvků v seznamu S je rovno X: `exactly(X, S, N)`

`exactly(_, [], 0).`

Příklad: reifikace

- Přesně N prvků v seznamu S je rovno X: `exactly(X, S, N)`

```
exactly(_, [], 0).
```

```
exactly(X, [Y|L], N) :-
```

```
    X #= Y #<=> B,
```

```
% reifikace
```

Příklad: reifikace

- Přesně N prvků v seznamu S je rovno X: `exactly(X, S, N)`

```
exactly(_, [], 0).
```

```
exactly(X, [Y|L], N) :-
```

```
    X #= Y #<=> B,
```

% reifikace

```
    N #= M+B,
```

% doménová proměnná místo akumulátoru

```
    exactly(X, L, M).
```

Příklad: reifikace

- Přesně N prvků v seznamu S je rovno X: `exactly(X, S, N)`

```
exactly(_, [], 0).
```

```
exactly(X, [Y|L], N) :-
```

```
    X #= Y #<=> B,
```

% reifikace

```
    N #= M+B,
```

% doménová proměnná místo akumulátoru

```
    exactly(X, L, M).
```

- | ?- `domain([A,B,C,D,E,N],1,2), exactly(1,[A,B,C,D,E],N),`

Příklad: reifikace

- Přesně N prvků v seznamu S je rovno X: `exactly(X, S, N)`

```
exactly(_, [], 0).
```

```
exactly(X, [Y|L], N) :-
```

```
    X #= Y #<=> B,
```

% reifikace

```
    N #= M+B,
```

% doménová proměnná místo akumulátoru

```
    exactly(X, L, M).
```

- | ?- `domain([A,B,C,D,E,N],1,2), exactly(1,[A,B,C,D,E],N),A#< 2,`

Příklad: reifikace

- Přesně N prvků v seznamu S je rovno X: `exactly(X, S, N)`

```
exactly(_, [], 0).
```

```
exactly(X, [Y|L], N) :-
```

```
    X #= Y #<=> B,
```

% reifikace

```
    N #= M+B,
```

% doménová proměnná místo akumulátoru

```
    exactly(X, L, M).
```

- | ?- `domain([A,B,C,D,E,N],1,2), exactly(1,[A,B,C,D,E],N),A#< 2,B#< 2.`

Příklad: reifikace

- Přesně N prvků v seznamu S je rovno X: `exactly(X, S, N)`

```
exactly(_, [], 0).
```

```
exactly(X, [Y|L], N) :-
```

```
    X #= Y #<=> B,
```

% reifikace

```
    N #= M+B,
```

% doménová proměnná místo akumulátoru

```
    exactly(X, L, M).
```

- | ?- `domain([A,B,C,D,E,N],1,2), exactly(1,[A,B,C,D,E],N),A#< 2,B#< 2.`

A = 1, B = 1, C = 2, D = 2, E = 2, N = 2

Příklad: reifikace

- Přesně N prvků v seznamu S je rovno X: `exactly(X, S, N)`

```
exactly(_, [], 0).
```

```
exactly(X, [Y|L], N) :-
```

```
    X #= Y #<=> B,
```

% reifikace

```
    N #= M+B,
```

% doménová proměnná místo akumulátoru

```
    exactly(X, L, M).
```

- | ?- `domain([A,B,C,D,E,N],1,2), exactly(1, [A,B,C,D,E],N), A#< 2, B#< 2.`

A = 1, B = 1, C = 2, D = 2, E = 2, N = 2

- Vyzkoušejte si

- `greater(X, S, N)`: přesně N prvků v seznamu S je větší než X

- `atleast(X, S, N)`: alespoň N prvků v seznamu S je rovno X

- `atmost(X, S, N)`: nejvýše N prvků v seznamu S je rovno X

Základní kombinatorická omezení

● `element(N,List,X)`

- omezení na konkrétní prvek seznamu

● `all_distinct(List)`

- všechny proměnné různé

Základní kombinatorická omezení

● `element(N,List,X)`

- omezení na konkrétní prvek seznamu

● `all_distinct(List)`

- všechny proměnné různé

● `serialized(Starts,Durations)`

- disjunktivní rozvrhování

● `disjoint2(Rectangles)`

- nepřekrývání obdélníků

● `cumulative(Starts,Durations,Resources,Limit)`

- kumulativní rozvrhování

Výskyty prvků v seznamu

● `element(N,List,X)`

● N-tý prvek v seznamu `List` je roven `X`

● `| ?- A in 2..10, B in 1..3, element(N, [A,B], X),`

Výskyty prvků v seznamu

● `element(N,List,X)`

● N-tý prvek v seznamu `List` je roven `X`

● `| ?- A in 2..10, B in 1..3, element(N, [A,B], X), X#< 2.`

`B = 1, N = 2, X = 1, A in 2..10`

Výskyty prvků v seznamu

● `element(N,List,X)`

● N-tý prvek v seznamu `List` je roven `X`

● `| ?- A in 2..10, B in 1..3, element(N, [A,B], X), X#< 2.`

`B = 1, N = 2, X = 1, A in 2..10`

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

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

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

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

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

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

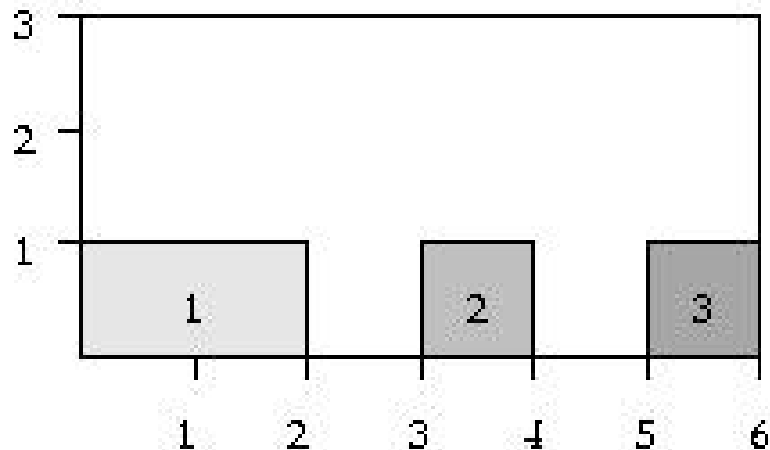
učitel	min	max
Jan	3	6
Petr	3	4
Anna	2	5
Ota	2	4
Eva	3	4
Marie	1	6

Disjunktivní rozvrhování

- `serialized(Starts, Durations)`
- Rozvržení úloh zadaných startovním časem (seznam `Starts`) a dobou trvání (seznam **nezáporných** `Durations`) tak, aby se nepřekrývaly

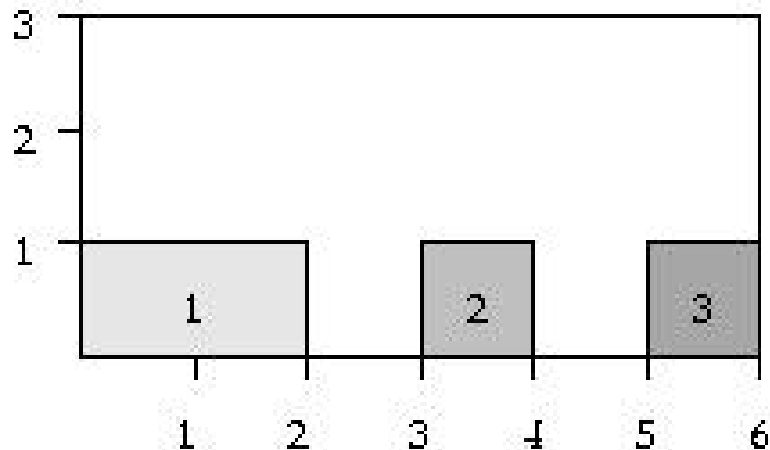
Disjunktivní rozvrhování

- `serialized(Starts,Durations)`
- Rozvržení úloh zadaných startovním časem (seznam Starts) a dobou trvání (seznam **nezáporných** Durations) tak, aby se nepřekrývaly
- příklad s konstantami: `serialized([0,3,5],[2,1,1])`



Disjunktivní rozvrhování

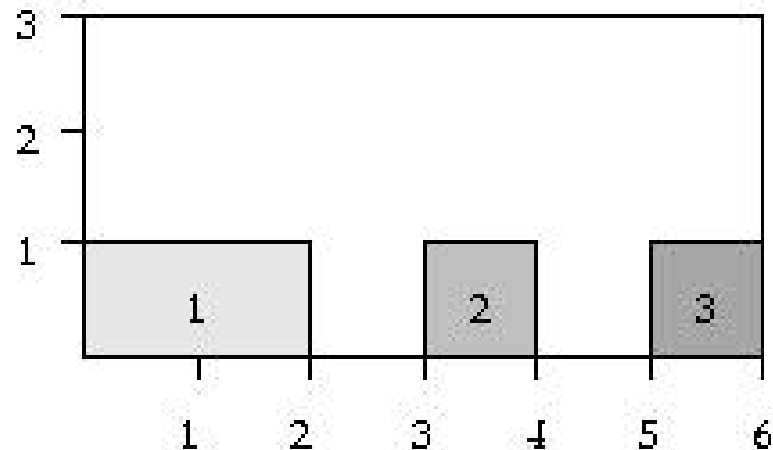
- `serialized(Starts, Durations)`
- Rozvržení úloh zadaných startovním časem (seznam Starts) a dobou trvání (seznam **nezáporných** Durations) tak, aby se nepřekrývaly
- příklad s konstantami: `serialized([0,3,5],[2,1,1])`



- příklad: vytvoření rozvrhu, za předpokladu, že **doba trvání hodin není stejná**

Disjunktivní rozvrhování

- `serialized(Starts,Durations)`
- Rozvržení úloh zadaných startovním časem (seznam Starts) a dobou trvání (seznam **nezáporných** Durations) tak, aby se nepřekrývaly
- příklad s konstantami: `serialized([0,3,5],[2,1,1])`



- příklad: vytvoření rozvrhu, za předpokladu, že **doba trvání hodin není stejná**
 - `D in 1..2, C = 3,`
`serialized([Jan,Petr,Anna,Ota,Eva,Marie], [D,D,D,C,C,C])`

Nepřekrývání obdélníků

- `disjoint2(Rectangles)` `disjoint1(Lines)`
`disjoint2([Name(X, Deřka, Y, Vyska) | _])`
- umístění obdélníků ve dvourozměrném prostoru
doménové proměnné `X, Y, Deřka, Vyska` mohou být z oboru **celých čísel**

Nepřekrývání obdélníků

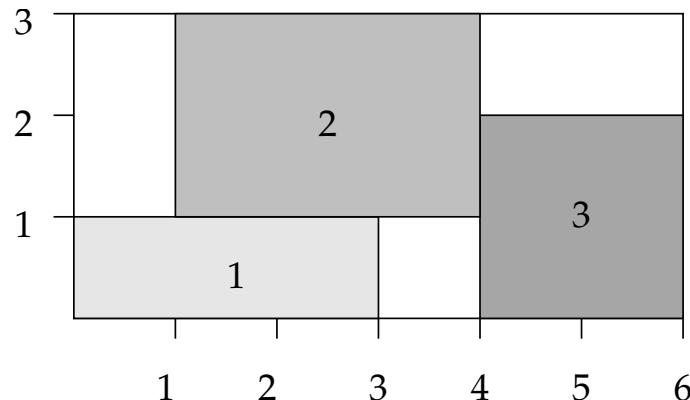
● `disjoint2(Rectangles)` `disjoint1(Lines)`

`disjoint2([Name(X, Deřka, Y, Vyska) | _])`

● umístění obdélníků ve dvourozměrném prostoru
doménové proměnné X,Y,Deřka,Vyska mohou být z oboru **celých čísel**

● příklad s konstantami:

`disjoint2([rect(0,3,0,1),rect(1,3,1,2),rect(4,2,2,-2)])`



Nepřekrývání obdélníků

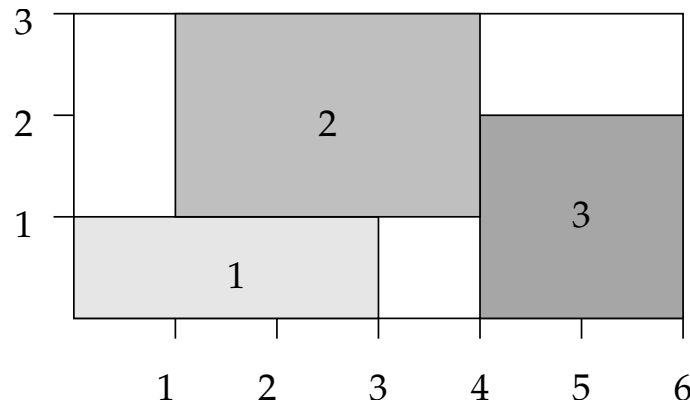
• `disjoint2(Rectangles)` `disjoint1(Lines)`

`disjoint2([Name(X, Deřka, Y, Vyska) | _])`

• umístění obdélníků ve dvourozměrném prostoru
doménové proměnné X,Y,Deřka,Vyska mohou být z oboru **celých čísel**

• příklad s konstantami:

`disjoint2([rect(0,3,0,1),rect(1,3,1,2),rect(4,2,2,-2)])`



• příklad: vytvoření rozvrhu za předpokladu, že **učitelé učí v různých místnostech**

`D in 1..2, C = 3,`

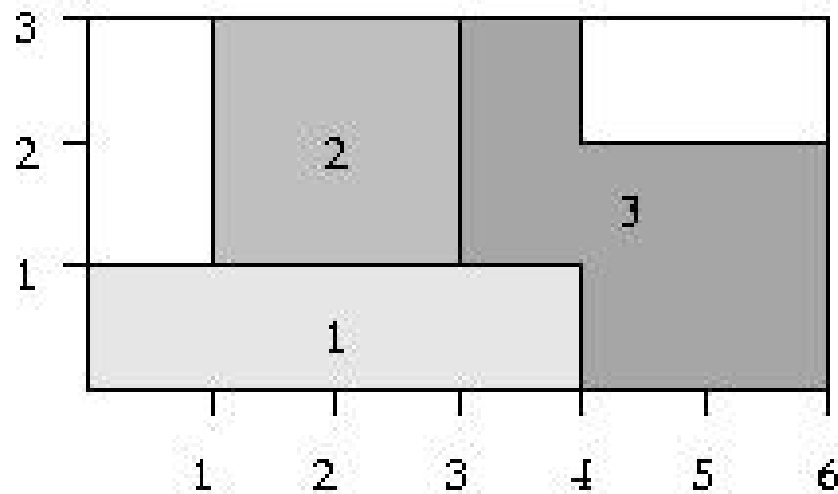
`disjoint2(class(Jan,D,M1,1), class(Petr,D,M2,1), class(Petr,D,M3,1), ...])`

Kumulativní rozvrhování

- `cumulative(Starts, Durations, Resources, Limit)`
- Úlohy jsou zadány startovním časem (seznam `Starts`), dobou trvání (seznam `Durations`) a požadovanou kapacitou zdroje (seznam `Resources`)
- Rozvržení úloh tak, aby celková kapacita zdroje nikdy nepřekročila `Limit`

Kumulativní rozvrhování

- `cumulative(Starts, Durations, Resources, Limit)`
- Úlohy jsou zadány startovním časem (seznam `Starts`), dobou trvání (seznam `Durations`) a požadovanou kapacitou zdroje (seznam `Resources`)
- Rozvržení úloh tak, aby celková kapacita zdroje nikdy nepřekročila `Limit`
- Příklad s konstantami: `cumulative([0,1,3],[4,2,3],[1,2,2],3)`



Příklad: kumulativní rozvrhování

- Vytvořte rozvrh pro následující úlohy, tak aby nebyla překročena kapacita 13 zdroje, a minimalizujte celkovou dobu trvání

úloha	doba trvání	kapacita
t1	16	2
t2	6	9
t3	13	3
t4	7	7
t5	5	10
t6	18	1
t7	4	11

Příklad: kumulativní rozvrhování

- Vytvořte rozvrh pro následující úlohy, tak aby nebyla překročena kapacita 13 zdroje, a minimalizujte celkovou dobu trvání

`schedule(Ss, End) :-`

úloha	doba trvání	kapacita
t1	16	2
t2	6	9
t3	13	3
t4	7	7
t5	5	10
t6	18	1
t7	4	11

Příklad: kumulativní rozvrhování

- Vytvořte rozvrh pro následující úlohy, tak aby nebyla překročena kapacita 13 zdroje, a minimalizujte celkovou dobu trvání

```
schedule(Ss, End) :-  
    length(Ss, 7),
```

úloha	doba trvání	kapacita
t1	16	2
t2	6	9
t3	13	3
t4	7	7
t5	5	10
t6	18	1
t7	4	11

Příklad: kumulativní rozvrhování

- Vytvořte rozvrh pro následující úlohy, tak aby nebyla překročena kapacita 13 zdroje, a minimalizujte celkovou dobu trvání

```
schedule(Ss, End) :-  
    length(Ss, 7),  
    Ds = [16, 6, 13, 7, 5, 18, 4],  
    Rs = [ 2, 9, 3, 7, 10, 1, 11],
```

úloha	doba trvání	kapacita
t1	16	2
t2	6	9
t3	13	3
t4	7	7
t5	5	10
t6	18	1
t7	4	11

Příklad: kumulativní rozvrhování

- Vytvořte rozvrh pro následující úlohy, tak aby nebyla překročena kapacita 13 zdroje, a minimalizujte celkovou dobu trvání

úloha	doba trvání	kapacita
t1	16	2
t2	6	9
t3	13	3
t4	7	7
t5	5	10
t6	18	1
t7	4	11

```
schedule(Ss, End) :-  
    length(Ss, 7),  
    Ds = [16, 6, 13, 7, 5, 18, 4],  
    Rs = [ 2, 9, 3, 7, 10, 1, 11],  
    domain(Ss, 0, 51),  
    domain([End], 0, 69),
```

Příklad: kumulativní rozvrhování

- Vytvořte rozvrh pro následující úlohy, tak aby nebyla překročena kapacita 13 zdroje, a minimalizujte celkovou dobu trvání

úloha	doba trvání	kapacita
t1	16	2
t2	6	9
t3	13	3
t4	7	7
t5	5	10
t6	18	1
t7	4	11

```
schedule(Ss, End) :-  
    length(Ss, 7),  
    Ds = [16, 6, 13, 7, 5, 18, 4],  
    Rs = [ 2, 9, 3, 7, 10, 1, 11],  
    domain(Ss, 0, 51),  
    domain([End], 0, 69),  
    after(Ss, Ds, End),      % koncový čas
```


Příklad: kumulativní rozvrhování

- Vytvořte rozvrh pro následující úlohy, tak aby nebyla překročena kapacita 13 zdroje, a minimalizujte celkovou dobu trvání

úloha	doba trvání	kapacita
t1	16	2
t2	6	9
t3	13	3
t4	7	7
t5	5	10
t6	18	1
t7	4	11

```
schedule(Ss, End) :-  
    length(Ss, 7),  
    Ds = [16, 6, 13, 7, 5, 18, 4],  
    Rs = [ 2, 9, 3, 7, 10, 1, 11],  
    domain(Ss, 0, 51),  
    domain([End], 0, 69),  
    after(Ss, Ds, End),    % koncový čas  
    cumulative(Ss, Ds, Rs, 13),
```

Příklad: kumulativní rozvrhování

- Vytvořte rozvrh pro následující úlohy, tak aby nebyla překročena kapacita 13 zdroje, a minimalizujte celkovou dobu trvání

úloha	doba trvání	kapacita
t1	16	2
t2	6	9
t3	13	3
t4	7	7
t5	5	10
t6	18	1
t7	4	11

```
schedule(Ss, End) :-  
    length(Ss, 7),  
    Ds = [16, 6, 13, 7, 5, 18, 4],  
    Rs = [ 2, 9, 3, 7, 10, 1, 11],  
    domain(Ss, 0, 51),  
    domain([End], 0, 69),  
    after(Ss, Ds, End),      % koncový čas  
    cumulative(Ss, Ds, Rs, 13),  
    append(Ss, [End], Vars),  
    labeling([minimize(End)], Vars).
```

Příklad: kumulativní rozvrhování

- Vytvořte rozvrh pro následující úlohy, tak aby nebyla překročena kapacita 13 zdroje, a minimalizujte celkovou dobu trvání

úloha	doba trvání	kapacita
t1	16	2
t2	6	9
t3	13	3
t4	7	7
t5	5	10
t6	18	1
t7	4	11

```
schedule(Ss, End) :-  
    length(Ss, 7),  
    Ds = [16, 6, 13, 7, 5, 18, 4],  
    Rs = [ 2, 9, 3, 7, 10, 1, 11],  
    domain(Ss, 0, 51),  
    domain([End], 0, 69),  
    after(Ss, Ds, End),      % koncový čas  
    cumulative(Ss, Ds, Rs, 13),  
    append(Ss, [End], Vars),  
    labeling([minimize(End)], Vars).  
  
after([], [], _).  
after([S|Ss], [D|Ds], E) :-  
    E #>= S+D, after(Ss, Ds, E).
```

Příklad: kumulativní rozvrhování

- Vytvořte rozvrh pro následující úlohy, tak aby nebyla překročena kapacita 13 zdroje, a minimalizujte celkovou dobu trvání

úloha	doba trvání	kapacita
t1	16	2
t2	6	9
t3	13	3
t4	7	7
t5	5	10
t6	18	1
t7	4	11

```
schedule(Ss, End) :-
    length(Ss, 7),
    Ds = [16, 6, 13, 7, 5, 18, 4],
    Rs = [ 2, 9, 3, 7, 10, 1, 11],
    domain(Ss, 0, 51),
    domain([End], 0, 69),
    after(Ss, Ds, End),      % koncový čas
    cumulative(Ss, Ds, Rs, 13),
    append(Ss, [End], Vars),
    labeling([minimize(End)], Vars).

after([], [], _).
after([S|Ss], [D|Ds], E) :-
    E #>= S+D, after(Ss, Ds, E).

| ?- schedule(Ss, End).
Ss = Ss = [0,16,9,9,4,4,0], End = 22 ?
```