Constraint satisfaction methods in embedded system design

Krzysztof Kuchcinski

Dept. of Computer Science,
Lund University, Sweden

Outline

1. Motivation an Example
2. CP Basics
3. Advanced Example- Sub-graph Isomorphism
4. Summary and Conclusions

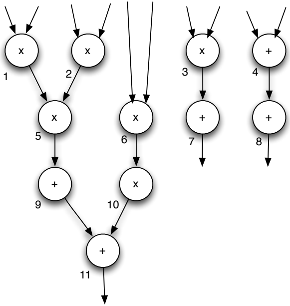
Outline

Why constraints?

- Examples of combinatorial optimization problems in embedded systems
  - Scheduling, allocation and assignment,
  - Partitioning,
  - Memory and register assignment,
  - Instruction selection.
- Different constraints:
  - timing,
  - resource,
  - power consumption, etc.
- Constraint programming over finite domain– combinatorial optimization problems!!!
- Constraint programming offers a *unified* approach to model and solve problems with *heterogeneous* constraints.

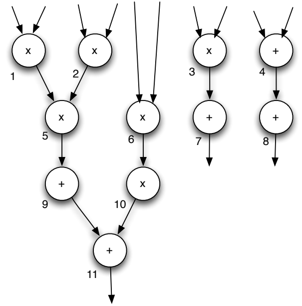
Scheduling example

Simple data-flow graph

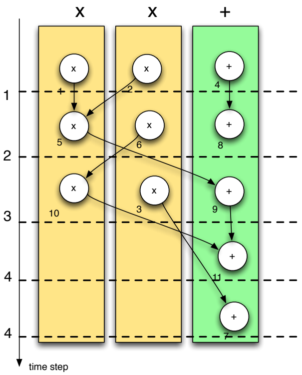


Scheduling example

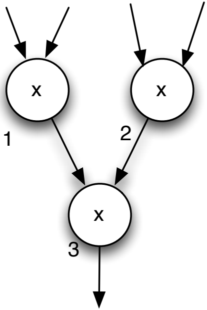
Simple data-flow graph



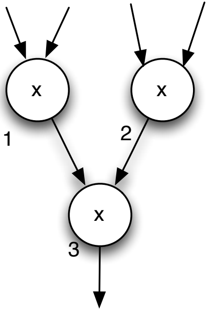
Simple schedule



Scheduling Constraints



Scheduling Constraints



Variables

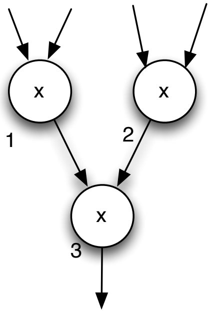
`Operation start`

$t_1 :: \{0..10\}, t_2 :: \{0..10\}, t_3 :: \{0..10\}$

`Assigned resource`

$r_1 :: \{1..2\}, r_2 :: \{1..2\}, r_3 :: \{1..2\}$

Scheduling Constraints



Variables

`Operation start`

$t_1 :: \{0..10\}, t_2 :: \{0..10\}, t_3 :: \{0..10\}$

`Assigned resource`

$r_1 :: \{1..2\}, r_2 :: \{1..2\}, r_3 :: \{1..2\}$

Constraints

`Precedence constraints`

$t_1 + d_1 \leq t_2 \wedge$
$t_2 + d_2 \leq t_3 \wedge$

`Resource constraints`

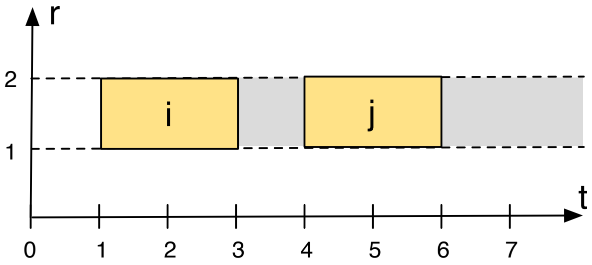
$(t_1 + d_1 \leq t_2 \vee t_2 + d_2 \leq t_1 \vee r_1 \neq r_2)$

Global Constraints

$\forall i, j$ `where` $i < j : t_i + d_i \leq t_j \vee t_j + d_j \leq t_i \vee r_i \neq r_j$

Global Constraints

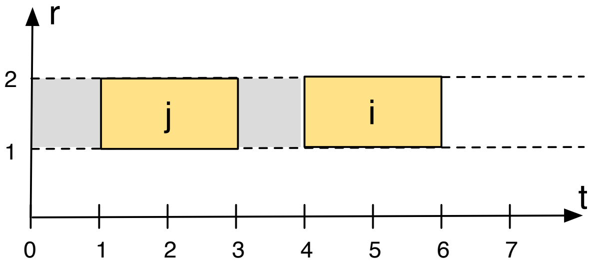
$$\forall i, j \ \texttt{where} \ i < j : t_i + d_i \leq t_j \vee t_j + d_j \leq t_i \vee r_i \neq r_j$$



Diff2 constraint (non-overlapping rectangles)

Global Constraints

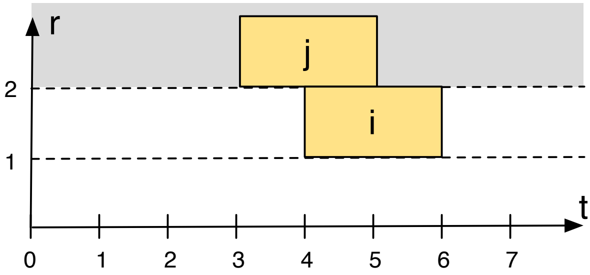
$$\forall i, j \texttt{ where } i < j : t_i + d_i \leq t_j \vee t_j + d_j \leq t_i \vee r_i \neq r_j$$



Diff2 constraint (non-overlapping rectangles)

Global Constraints

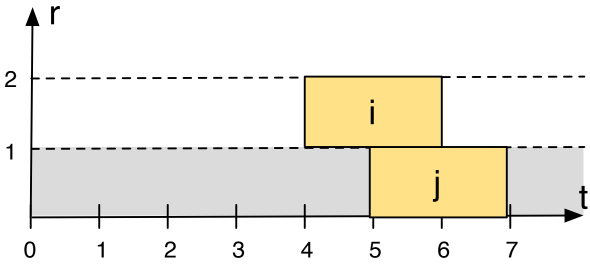
$$\forall i, j \;\texttt{where}\; i < j : t_i + d_i \le t_j \lor t_j + d_j \le t_i \lor r_i \ne r_j$$



Diff2 constraint (non-overlapping rectangles)

Global Constraints

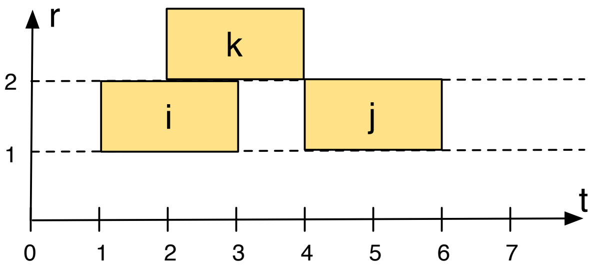
$$\forall i, j \texttt{ where } i < j : t_i + d_i \leq t_j \vee t_j + d_j \leq t_i \vee r_i \neq r_j$$



Diff2 constraint (non-overlapping rectangles)

Global Constraints

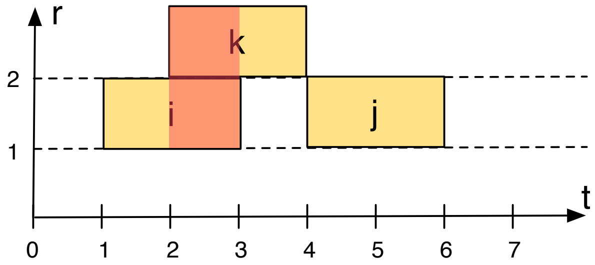
$\forall i, j$ where $i < j : t_i + d_i \leq t_j \vee t_j + d_j \leq t_i \vee r_i \neq r_j$



Diff2 constraint (non-overlapping rectangles)

Global Constraints

$$\forall i, j \ \texttt{where} \ i < j : t_i + d_i \leq t_j \vee t_j + d_j \leq t_i \vee r_i \neq r_j$$



Diff2 constraint (non-overlapping rectangles)

Final Model

```
array[1..n] of var 0..100 : t;
array[1..n] of var 1..2 : r;

% precedence constraints
constraint
   t[1] + 2 =< t[6] /\ t[2] + 2 =< t[6] /\ t[3] + 2 =< t[7] /\
   t[4] + 2 =< t[8] /\ t[5] + 1 =< t[9] /\ t[6] + 2 =< t[10] /\
   t[7] + 2 =< t[11] /\ t[10] + 1 =< t[11];

constraint
   % resource constraints for adders
   diff2([[t[5],r[5],1,1], [t[8],r[8],1,1], [t[9],r[9],1,1],
          [t[10],r[10],1,1], [t[11],r[11],1,1] ])
   /\
   % resource constraints for multipliers
   diff2([[t[1],r[1],2,1], [t[2],r[2],2,1], [t[3],r[3],2,1],
          [t[4],r[4],2,1], [t[6],r[6],2,1], [t[7],r[7],2,1]]);
```

Model Advantages

- Separation of a model and solving method
- Time-constrained and resource-constrained scheduling
- Easy to add new constraints
- Non-linear constraints
- Combination of consistency algorithms (e.g., diff2 and cumulative constraints)
- Standard and heuristic methods for solving the model

Outline

CP basics

- Finite domain variables, e.g., $t :: 0..10$
- Constraints; defined by their consistency methods (propagators)
- Primitive constraints
  - $a + b < c$, $x \cdot y = z$, $A \cup B = C$, etc.
  - bounds and domain consistency
- Global constraints
  - diff2, alldifferent, etc.
  - can be decomposed to primitive constraints BUT
  - specialized algorithms from operation research, graph theory, computational geometry, etc. are more efficient

Propagators

Propagator for $x + y = z$ (bounds consistency)

$$\begin{aligned} x \ \texttt{in} \ & \{\min(z) - \max(y) \ .. \ \max(z) - \min(y)\} \\ y \ \texttt{in} \ & \{\min(z) - \max(x) \ .. \ \max(z) - \min(x)\} \\ z \ \texttt{in} \ & \{\min(x) + \min(y) \ .. \ \max(x) + \max(y)\} \end{aligned}$$

Propagators

Propagator for $x + y = z$ (bounds consistency)

$$
\begin{aligned}
x \ \mathtt{in}\ & \{\min(z) - \max(y) \ .. \ \max(z) - \min(y)\} \\
y \ \mathtt{in}\ & \{\min(z) - \max(x) \ .. \ \max(z) - \min(x)\} \\
z \ \mathtt{in}\ & \{\min(x) + \min(y) \ .. \ \max(x) + \max(y)\}
\end{aligned}
$$

Example

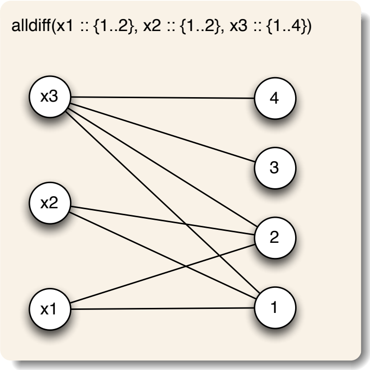
$x :: \{1..10\}$, $y :: \{1..10\}$ and $z :: \{1..10\}$
yields
$x :: \{1..9\}$, $y :: \{1..9\}$ and $z :: \{2..10\}$.

Global Constraints

- alldifferent, cumulative, table, etc.
- geometrical constraints: diff2, geost,
- combinatorial problems: binpacking, knapsack, network flow, etc.
- *graph constraints*: (sub-)graph isomorphism, clique, Hamiltonian path, simple path, connected components.

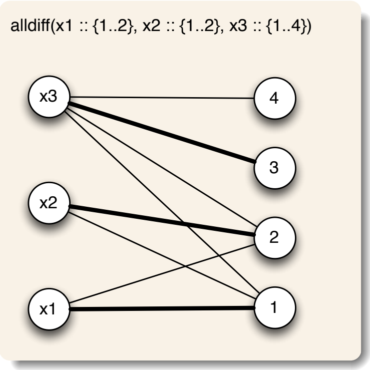
Global Constraints

- alldifferent, cumulative, table, etc.
- geometrical constraints: diff2, geost,
- combinatorial problems: binpacking, knapsack, network flow, etc.
- *graph constraints*: (sub-)graph isomorphism, clique, Hamiltonian path, simple path, connected components.



Global Constraints

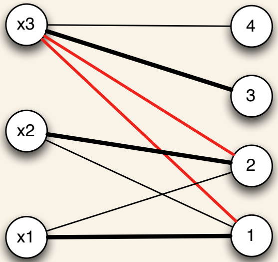
- alldifferent, cumulative, table, etc.
- geometrical constraints: diff2, geost,
- combinatorial problems: binpacking, knapsack, network flow, etc.
- *graph constraints*: (sub-)graph isomorphism, clique, Hamiltonian path, simple path, connected components.



Global Constraints

- alldifferent, cumulative, table, etc.
- geometrical constraints: diff2, geost,
- combinatorial problems: binpacking, knapsack, network flow, etc.
- *graph constraints*: (sub-)graph isomorphism, clique, Hamiltonian path, simple path, connected components.

alldiff(x1 :: {1..2}, x2 :: {1..2}, x3 :: {1..4})

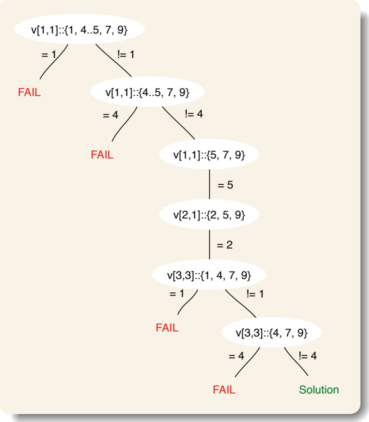


Berge, 1973

An edge belongs to a maximum matching iff for some maximum matching, it belongs to either an even alternating path which begins at a free node, or to an even alternating cycle.

Solving

- Systematically assign values to variables and check if the problem is still consistent
- Implemented usually as depth-first-search
- Other methods can be used instead of assigning values, i.e., constraints on tasks ordering
- Heuristics can be incorporated



Outline

Subgraph Isomorphism Constraint

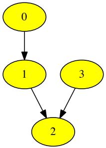
Definition (Subgraph isomorphism)

Target $G_t = (N_t, E_t)$ and pattern $G_p = (N_p, E_p)$ graphs are subgraph isomorphic iff there exist an injective function $f : N_p \to N_t$ respecting $(u, v) \in E_p \Leftrightarrow (f(u), f(v)) \in E_t$.

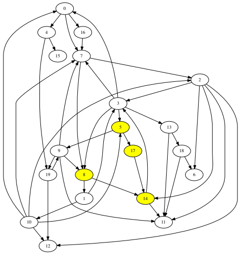
Subgraph Isomorphism Constraint

Definition (Subgraph isomorphism)

Target $G_t = (N_t, E_t)$ and pattern $G_p = (N_p, E_p)$ graphs are subgraph isomorphic iff there exist an injective function $f : N_p \rightarrow N_t$ respecting $(u, v) \in E_p \Leftrightarrow (f(u), f(v)) \in E_t$.

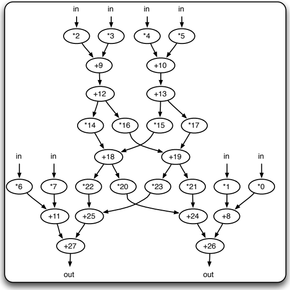


pattern graph

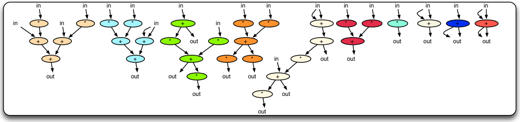


target graph with matching

Instruction Identification and Selection



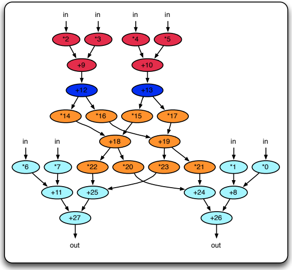
Data-flow graph



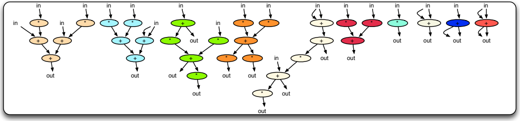
Computational patterns

- Computational patterns - connected components of the graph

Instruction Identification and Selection (cont'd)



Covered data-flow graph



Computational patterns

- Find sub-graph isomorphism that fulfills additional constraints (e.g., shortest schedule)

Outline

Our Solver

Java **Co**nstraint **P**rogramming

- constraint programming paradigm implemented in Java.
- provides different type of constraints
  - *primitive constraints*, such as arithmetical constraints (+, *, div, mod, etc.), equality (=) and inequalities (<, >, =<, >=, !=).
  - *logical, reified and conditional constraints*
  - *global constraints*.
  - *set constraints*, such as =, $\bigcup$, $\bigcap$.
  - *stochastic variables and constraints*.
- High-level language, minizinc, interface
- `http://www.jacop.eu`
- `http://sourceforge.net/projects/jacop-solver/`

Conclusions

- Easy way of modeling problems with heterogeneous constraints
- Easy to extend the problem with new constraints
- Can handle non-linear constraints
- Combination of different algorithms through global constraints
- Separation between modeling and solving
- Both complete and heuristic methods can be used for finding solutions