Constraint Relationships

Language Features

Preferences in Constraint Solving

Constraint problem (X, D, C)

- Variables X, Domains $D = (D_x)_{x \in X}$, Constraints C

How to deal with over-constrained problems?

$((\{x, y, z\}, D_x = D_y = D_z = \{1, 2, 3\}), \{c_1, c_2, c_3\})$ mit

$\qquad c_1 : x + 1 = y$

$\qquad c_2 : z = y + 2$

$\qquad c_3 : x + y \leq 3$

- Not all constraints can be satisfied simultaneously

Preferences in Constraint Solving

Constraint problem (X, D, C)

- Variables X, Domains $D = (D_x)_{x \in X}$, Constraints C

How to deal with over-constrained problems?

$$((\{x, y, z\}, D_x = D_y = D_z = \{1, 2, 3\}), \{c_1, c_2, c_3\}) \text{ mit}$$

$$c_1 : x + 1 = y$$
$$c_2 : z = y + 2$$
$$c_3 : x + y \leq 3$$

- Not all constraints can be satisfied simultaneously
  - e.g., c_2 forces $z = 3$ and $y = 1$, conflicting c_1
- We can choose between assignments satisfying $\{c_1, c_3\}$ or $\{c_2, c_3\}$.

Which assignments $v \in [X \to D]$ should be preferred by an agent/several agents?

Constraint Relationships

Approach (**?**)

- Define relation R over constraints C to denote which constraints are more important than others, e. g.
    - c_1 is more important than c_2
    - c_1 is more important than c_3
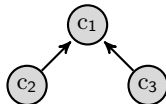
Benefits

- Qualitative formalism — easy to specify
- Graphical interpretation
    - Semantics (how much more important is a constraint) regulated by
    - dominance properties that are either "hierarchical" or "egalitarian"
    - Single-Predecessors-Dominance (SPD) vs. Transitive-Predecessors-Dominance (TPD)

SoftConstraints in MiniZinc

```
% X: {x,y,z} D_i = {1,2,3}, i in X
%    * c1: x + 1 = y * c2: z = y + 2 * c3: x + y <= 3
% (c) ISSE
% isse.uni-augsburg.de/en/software/constraint-relationships/
include "soft_constraints/minizinc_bundle.mzn";

var 1..3: x; var 1..3: y; var 1..3: z;

% read as "soft constraint c1 is satisfied iff x + 1 = y"
constraint x + 1 = y <-> satisfied[1];
constraint z = y + 2 <-> satisfied[2];
constraint x + y <= 3 <-> satisfied[3];

% soft constraint specific for this model
nScs = 3; nCrEdges = 2;
crEdges = [| 2, 1 | 3, 1 |]; % read c2 is less important than c1

solve minimize penSum; % minimize the sum of penalties
```
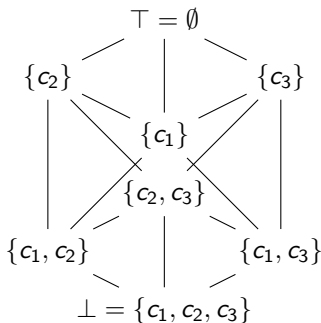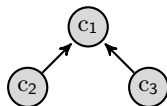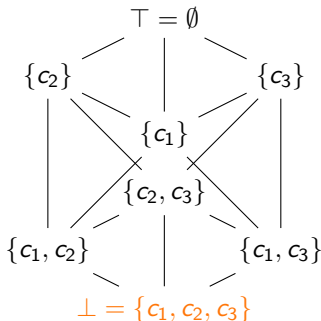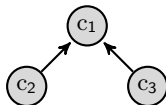
Search types

The whole valuation space (partially ordered)


```
%
% Typical Optimization Routine (Branch and Bound):
%
%  1. Look for the first feasible solution
%  2. Impose restrictions on the next feasible solution
%  3. Repeat
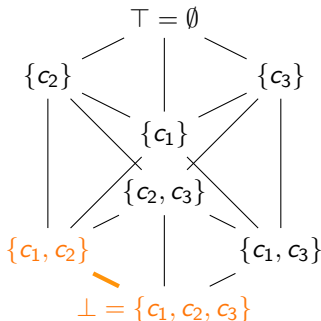```
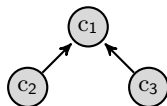
# Search types: Strictly better

The whole valuation space (partially ordered)



$$\top = \emptyset$$

$$\{c_2\} \qquad \{c_3\}$$

$$\{c_1\}$$

$$\{c_2, c_3\}$$

$$\{c_1, c_2\} \qquad \{c_1, c_3\}$$

$$\bot = \{c_1, c_2, c_3\}$$

```
function ann: strictlyBetterBAB(var set of SOFTCONSTRAINTS: vScs)
    = repeat(
        if next() then
          commit() /\
          post(isWorse(sol(vScs), vScs))
        else break endif   );
```
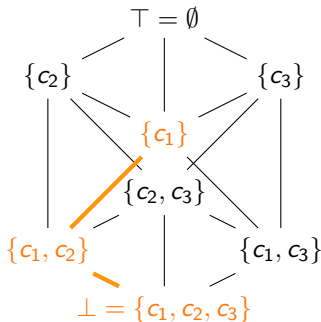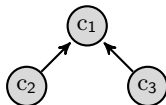
# Search types: Strictly better

The whole valuation space (partially ordered)



$\top = \emptyset$

$\{c_2\}$      $\{c_3\}$

$\{c_1\}$

$\{c_2, c_3\}$

$\{c_1, c_2\}$      $\{c_1, c_3\}$

$\bot = \{c_1, c_2, c_3\}$

```
function ann: strictlyBetterBAB(var set of SOFTCONSTRAINTS: vScs)
    = repeat(
          if next() then
            commit() /\
            post(isWorse(sol(vScs), vScs))
          else break endif      );
```

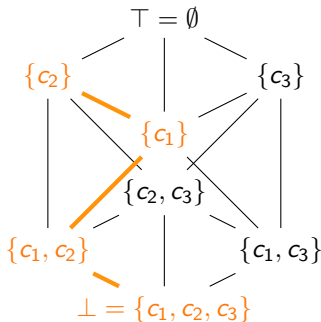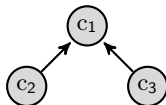# Search types: Strictly better
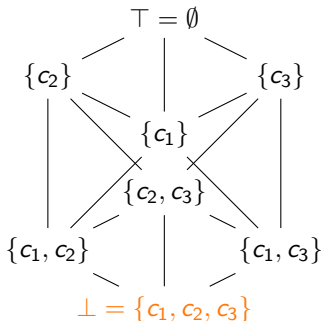
The whole valuation space (partially ordered)


```
function ann: strictlyBetterBAB(var set of SOFTCONSTRAINTS: vScs)
    = repeat(
        if next() then
          commit() /\
          post(isWorse(sol(vScs), vScs))
        else break endif      );
```

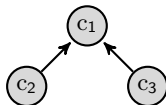# Search types: Strictly better

The whole valuation space (partially ordered)


```
function ann: strictlyBetterBAB(var set of SOFTCONSTRAINTS: vScs)
    = repeat(
         if next() then
           commit() /\
           post(isWorse(sol(vScs), vScs))
         else break endif      );
```
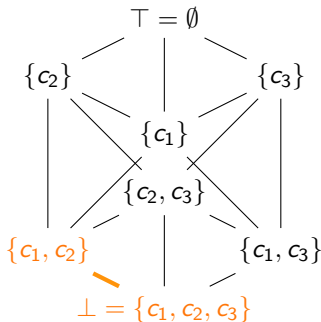
Search types: Only not dominated

The whole valuation space (partially ordered)

$$\top = \emptyset$$

$$\{c_2\} \qquad \{c_3\}$$

$$\{c_1\}$$

$$\{c_2, c_3\}$$

$$\{c_1, c_2\} \qquad \{c_1, c_3\}$$

$$\bot = \{c_1, c_2, c_3\}$$

```
function ann: onlyNotDominatedBAB(var set of SOFTCONSTRAINTS: vScs)
    = repeat(
        if next() then
          commit() /\
          post((isWorse(vScs, sol(vScs)) \/ vScs = sol(vScs)))
        else break endif    );
```

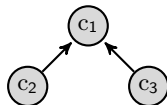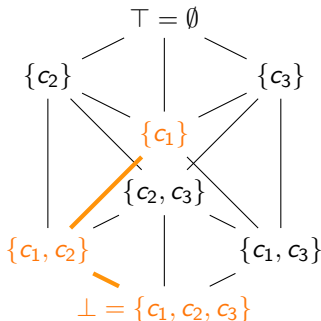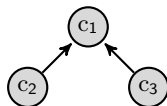# Search types: Only not dominated

The whole valuation space (partially ordered)


```
function ann: onlyNotDominatedBAB(var set of SOFTCONSTRAINTS: vScs)
    = repeat(
        if next() then
          commit() /\
          post((isWorse(vScs, sol(vScs)) \/ vScs = sol(vScs)))
        else break endif       );
```

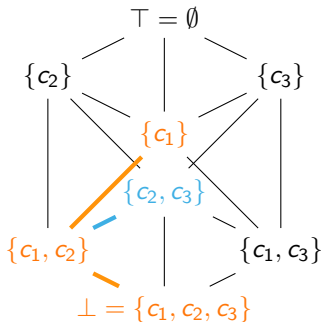# Search types: Only not dominated
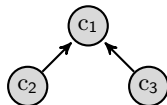
The whole valuation space (partially ordered)


```
function ann: onlyNotDominatedBAB(var set of SOFTCONSTRAINTS: vScs)
    = repeat(
        if next() then
          commit() /\
          post((isWorse(vScs, sol(vScs)) \/ vScs = sol(vScs)))
        else break endif      );
```

Search types: Only not dominated
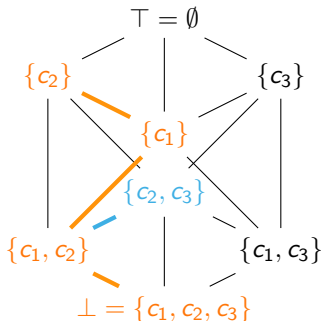
The whole valuation space (partially ordered)


```
function ann: onlyNotDominatedBAB(var set of SOFTCONSTRAINTS: vScs)
    = repeat(
        if next() then
          commit() /\
          post((isWorse(vScs, sol(vScs)) \/ vScs = sol(vScs)))
        else break endif    );
```

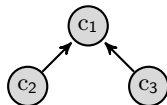# Search types: Only not dominated

The whole valuation space (partially ordered)


```
function ann: onlyNotDominatedBAB(var set of SOFTCONSTRAINTS: vScs)
    = repeat(
        if next() then
          commit() /\
          post((isWorse(vScs, sol(vScs)) \/ vScs = sol(vScs)))
        else break endif    );
```

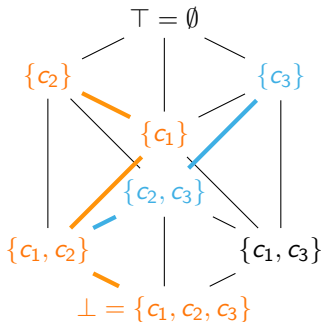# Search types: Only not dominated
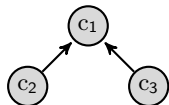
The whole valuation space (partially ordered)


```
function ann: onlyNotDominatedBAB(var set of SOFTCONSTRAINTS: vScs)
    = repeat(
        if next() then
          commit() /\
          post((isWorse(vScs, sol(vScs)) \/ vScs = sol(vScs)))
        else break endif      );
```

Language Features

a

Language Features: Suitable Weighting

a

a

Language Features: Variable Ordering

a

Language Features: Redundant Constraints

a

Language Features: Custom Search

a

Quellen I