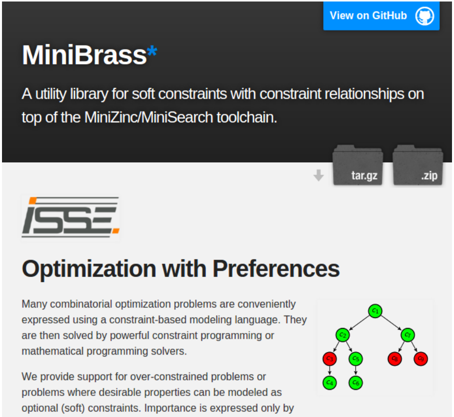
MiniBrass: Soft Constraint Programming

Alexander Schiendorfer et al.

Universität Augsburg
University

MiniBrass



View on GitHub

MiniBrass*

A utility library for soft constraints with constraint relationships on top of the MiniZinc/MiniSearch toolchain.

tar.gz .zip

Optimization with Preferences

Many combinatorial optimization problems are conveniently expressed using a constraint-based modeling language. They are then solved by powerful constraint programming or mathematical programming solvers.

We provide support for over-constrained problems or problems where desirable properties can be modeled as optional (soft) constraints. Importance is expressed only by

http://isse-augsburg.github.io/minibrass/

MiniBrass: HelloWorld

Institute for Software & Systems Engineering

Basismodell (MiniZinc)

```
include "hello_o.mzn";
include "soft_constraints/
   pvs_gen_search.mzn";
% the basic, "classic" CSP
set of int: DOM = 1..3;

var DOM: x; var DOM: y;
var DOM: z;
% add. *hard* constraints
% e.g. constraint x < y;

solve search pvs_BAB();
```

Präferenzmodell (MiniBrass)

```
PVS: cr1 =
  new ConstraintRelationships("cr1") {
   soft-constraint c1: 'x + 1 = y';
   soft-constraint c2: 'z = y + 2';
   soft-constraint c3: 'x + y <= 3';

   crEdges : '[| mbr.c2, mbr.c1 |
                 mbr.c3, mbr.c1 |]';
   useSPD: 'true' ;
};

solve cr1;
```

MiniBrass: HelloWorld

Basismodell (MiniZinc)

```
include "hello_o.mzn";
include "soft_constraints/
   pvs_gen_search.mzn";
% the basic, "classic" CSP
set of int: DOM = 1..3;

var DOM: x; var DOM: y;
var DOM: z;
% add. *hard* constraints
% e.g. constraint x < y;

solve search pvs_BAB();
```

Präferenzmodell (MiniBrass)

```
PVS: cr1 =
  new ConstraintRelationships("cr1") {
   soft-constraint c1: 'x + 1 = y';
   soft-constraint c2: 'z = y + 2';
   soft-constraint c3: 'x + y <= 3';

   crEdges : '[| mbr.c2, mbr.c1 |
                 mbr.c3, mbr.c1 |]';
   useSPD: 'true' ;
};

solve cr1;
```

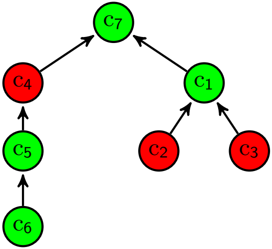
```
Solution:  x = 1; y = 2; z = 1
Valuations:  mbr_overall_cr1 = 2..2
----------
==========
```

Single-Predecessor-Dominance (SPD) Lifting

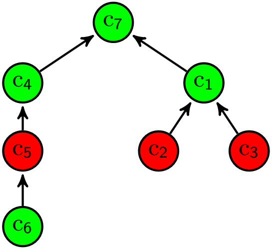
isWorseThan-Relation für Mengen verletzter Constraints (Schiendorfer et al., 2013)

$$V \uplus \{c\} \supset_{\mathrm{SPD}} V$$

$$V \uplus \{c_{\mathrm{imp}}\} \supset_{\mathrm{SPD}} V \uplus \{c_{\neg\mathrm{imp}}\} \quad \text{if } c_{\neg\mathrm{imp}} \to c_{\mathrm{imp}}$$



$\supset_{\mathrm{SPD}}$

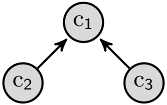


- Bekannt als *Smyth-Ordnung* (Powerdomains) (Amadio and Curien, 1998, Ch. 9)
- Entsteht aus *freier Konstruktion* über Constraint-Relationship.(Knapp et al., 2014)

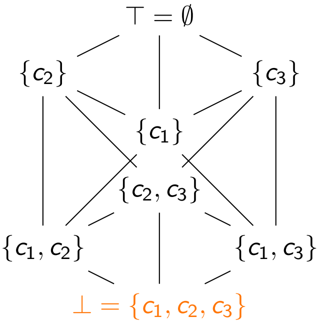
Constraint-Optimierung mit PVS

Institute for Software & Systems Engineering

Der partiell geordnete Bewertungsraum




```
c1: 'x + 1 = y';
c2: 'z = y + 2';
c3: 'x + y <= 3';
```

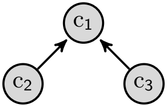



```
function ann: pvs_BAB() =
      repeat(
          if next() then
                print("Intermediate solution:") /\ print() /\
                commit() /\ postGetBetter()
          else break endif      );
```

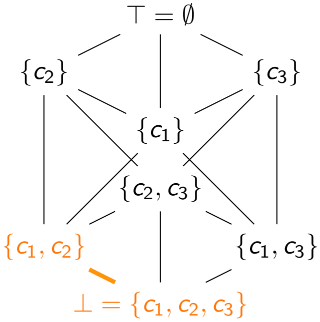
Constraint-Optimierung mit PVS

Institute for Software & Systems Engineering

Der partiell geordnete Bewertungsraum




```
c1: 'x + 1 = y';
c2: 'z = y + 2';
c3: 'x + y <= 3';
```

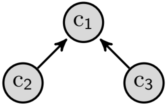



```
x = 1; y = 1; z = 1
Valuation = 1..2
```

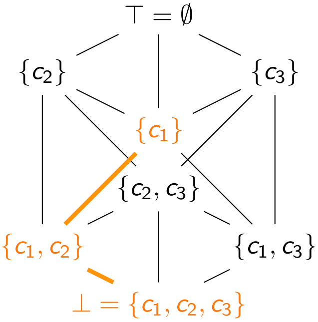
```
function ann: pvs_BAB() =
      repeat(
          if next() then
                print("Intermediate solution:") /\ print() /\
                commit() /\ postGetBetter()
          else break endif      );
```

Constraint-Optimierung mit PVS

Der partiell geordnete Bewertungsraum




```
c1: 'x + 1 = y';
c2: 'z = y + 2';
c3: 'x + y <= 3';
```

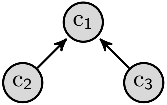



```
x = 1; y = 1; z = 1
Valuation = 1..2
----------
x = 1; y = 1; z = 3
Valuation = 1..1
```

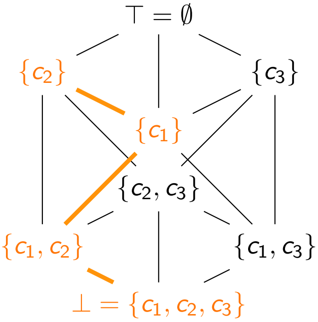
```
function ann: pvs_BAB() =
      repeat(
          if next() then
               print("Intermediate solution:") /\ print() /\
               commit() /\ postGetBetter()
          else break endif      );
```

Constraint-Optimierung mit PVS

Der partiell geordnete Bewertungsraum




```
c1: 'x + 1 = y';
c2: 'z = y + 2';
c3: 'x + y <= 3';
```

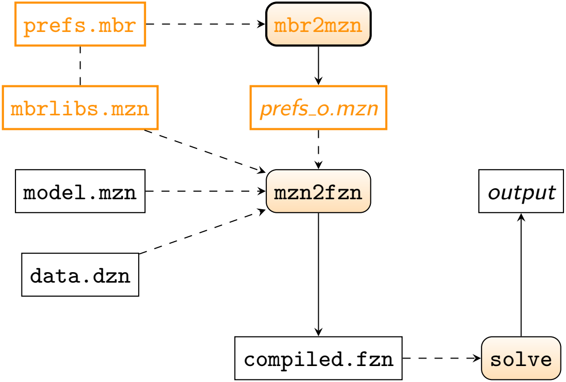



```
x = 1; y = 1; z = 1
Valuation = 1..2
----------
x = 1; y = 1; z = 3
Valuation = 1..1
----------
x = 1; y = 2; z = 1
Valuations = 2..2
----------
==========
```

```
function ann: pvs_BAB() =
      repeat(
          if next() then
               print("Intermediate solution:") /\ print() /\
               commit() /\ postGetBetter()
          else break endif      );
```

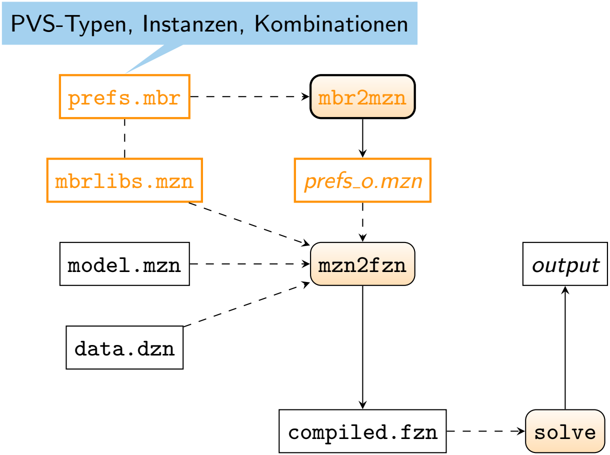
MiniBrass: Workflow

Architektur



MiniBrass: Workflow

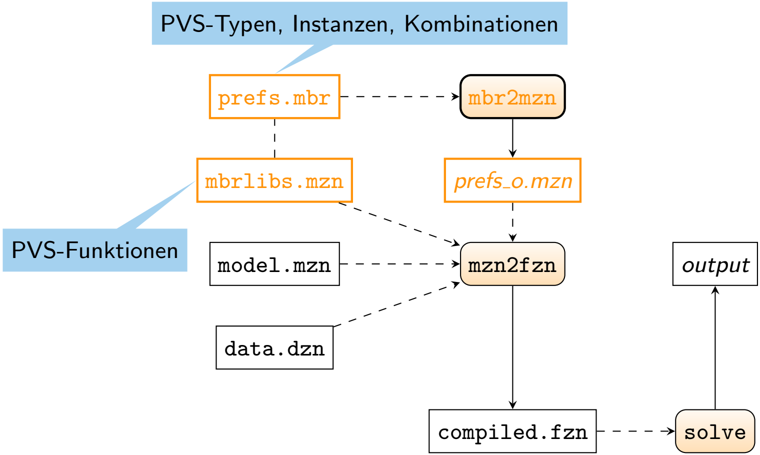
Architektur



MiniBrass: Workflow

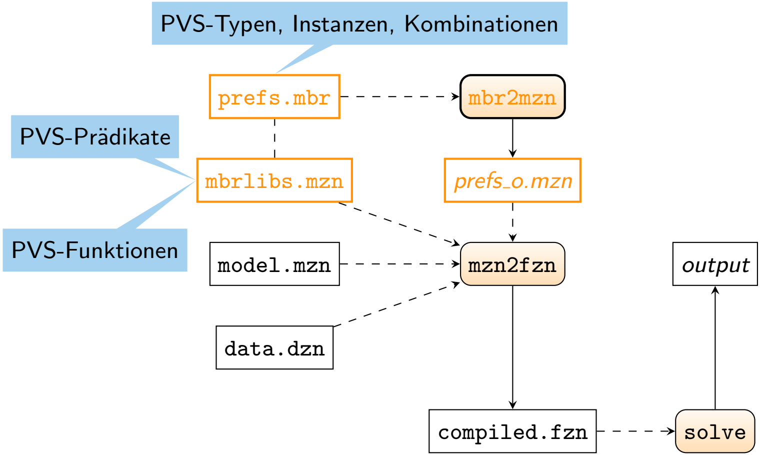
Institute for
Software & Systems
Engineering

Architektur



MiniBrass: Workflow

Architektur



PVS-Typdefinitionen

Institute for
Software & Systems
Engineering

```
type ConstraintRelationships = PVSType<bool, set of 1..nScs> =
  params {
    array[int, 1..2] of 1..nScs: crEdges; % adjacency matrix
    bool: useSPD;
  } in
  instantiates with "../mbr_types/cr_type.mzn" {
    times -> link_invert_booleans;
    is_worse -> is_worse_cr;
    top -> {};
  };
```

- `PVSType<S,E>` Unterscheidet zur einfacheren Verwendung zwischen
  Spezifikationstyp S
  Elementtyp E
- Kombinationsoperation: $\texttt{times} : S^n \rightarrow E$
- Ordnungsrelation: $\texttt{isWorse} \subseteq E \times E$

PVS-Definitionen

Innerhalb von `../mbr_types/cr_type.mzn`:

```
function var set of int:
  link_invert_booleans(array[int] of var bool: b...

% gives us access to constraint relationship predicates
include "soft_constraints/spd_worse.mzn";
include "soft_constraints/tpd_worse.mzn";

predicate is_worse_cr(var set of int: violated1,
                      var set of int: violated2,
                      par int: nScs, array[int, 1..2] of par int: crEdges,
                      par bool: useSPD) =
let { par set of int: softConstraints = 1..nScs; } in (
    if useSPD then
      spd_worse(violated1, violated2, softConstraints, crEdges)
    else
      tpd_worse(violated1, violated2, softConstraints, crEdges)
    endif);
```

PVS-Instanziierung

```
PVS: cr1 = new ConstraintRelationships("cr1") {
   soft-constraint c1: 'x + 1 = y';
   soft-constraint c2: 'z = y + 2';
   soft-constraint c3: 'x + y <= 3';

   crEdges : '[| mbr.c2, mbr.c1 | mbr.c3, mbr.c1 |]';
   useSPD: 'false' ;
};
```

- Jeder Soft-Constraint ein S-Ausdruck (hier z.B. `bool`)
- Mittels der Funktion `times` auf einen E-Wert abgebildet
- Ausdrücke in einfachen Anführungszeichen: MiniZinc-Code (nicht geparst, bis auf `mbr.`-Präfixe)
- Parameter aus `PVSType` müssen Wert erhalten

Weitere PVS-Typen

```
type WeightedCsp = PVSType<bool, int> =
  params {
    int: k;
    array[1..nScs] of 1..k: weights :: default('1');
  } in
  instantiates with "../mbr_types/weighted_type.mzn" {
    times -> weighted_sum;
    is_worse -> is_worse_weighted;
    top -> 0;
  };

type CostFunctionNetwork = PVSType<0..k> =
  params {
    int: k :: default('1000');
  } in instantiates with "../mbr_types/cfn_type.mzn" {
    times -> sum;
    is_worse -> is_worse_weighted;
    top -> 0;
 };
```

PVS-Instanziierung Weighted

Institute for Software & Systems Engineering

```
PVS: cr1 = new WeightedCsp("cr1") {
   soft-constraint c1: 'x + 1 = y' :: weights('2');
   soft-constraint c2: 'z = y + 2' :: weights('1');
   soft-constraint c3: 'x + y <= 3' :: weights('1');

   k : '20';
};
```

- Gewichte können direkt an Soft Constraints annotiert werden
- Oder direkt als Feld übergeben werden (`[2,1,1]`)

PVS-Instanziierung Weighted

```
PVS: cr1 = new WeightedCsp("cr1") {
   soft-constraint c1: 'x + 1 = y' :: weights('2');
   soft-constraint c2: 'z = y + 2' :: weights('1');
   soft-constraint c3: 'x + y <= 3' :: weights('1');

   k : '20';
};
```

- Gewichte können direkt an Soft Constraints annotiert werden
- Oder direkt als Feld übergeben werden (`[2,1,1]`)
- Aber können wir sie nicht auch berechnen? Aus Constraint Relationships?

PVS-Instanziierung Weighted

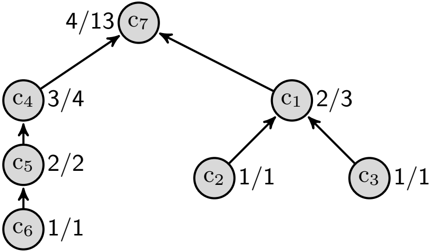
```
PVS: cr1 = new WeightedCsp("cr1") {
   soft-constraint c1: 'x + 1 = y' :: weights('2');
   soft-constraint c2: 'z = y + 2' :: weights('1');
   soft-constraint c3: 'x + y <= 3' :: weights('1');

   k : '20';
};
```

- Gewichte können direkt an Soft Constraints annotiert werden
- Oder direkt als Feld übergeben werden (`[2,1,1]`)
- Aber können wir sie nicht auch berechnen? Aus Constraint Relationships?

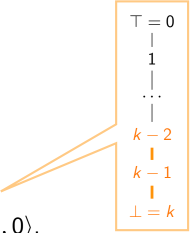
Gewichte für Constraint Relationships

Institute for Software & Systems Engineering



Beispiel mit errechneten Gewichten

$$w^{\mathrm{SPD}}(c) = 1 + \max_{c' \rightarrow^+ c} w^{\mathrm{SPD}}(c')$$
$$w^{\mathrm{TPD}}(c) = 1 + \sum_{c' \rightarrow^+ c} w^{\mathrm{TPD}}(c')$$



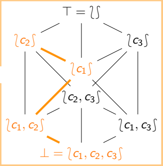
Für Ordnung P über Constraints: PVS $\mathrm{Weighted}(P) = \langle \mathbb{N}, +, \geq, 0 \rangle$.

(Schiendorfer et al., 2013)

PVS und Morphismen

Etwas systematischer ...

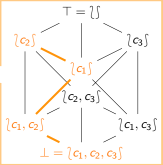
- $\text{PVS}_{\text{cr}} = \mathit{PVS}\langle P \rangle = \langle \mathcal{M}^{\text{fin}}(P), \uplus, \supseteq_{\text{SPD}}, \lbag \rbag \rangle$
- PVS-Homomorphismus $\varphi : \text{PVS}_{\text{cr}} \to \text{PVS}_{\text{weighted}}$



PVS und Morphismen

Etwas systematischer ...

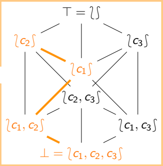
- $\mathrm{PVS}_{\mathrm{cr}} = \mathit{PVS}\langle P \rangle = \langle \mathcal{M}^{\mathrm{fin}}(P), \uplus, \supseteq_{\mathsf{SPD}}, \lbag\rbag \rangle$
- PVS-Homomorphismus $\varphi : \mathrm{PVS}_{\mathrm{cr}} \to \mathrm{PVS}_{\mathrm{weighted}}$
    - $\varphi(\top_{\mathrm{cr}}) = \top_{\mathrm{weighted}}$



PVS und Morphismen

Etwas systematischer ...

- $\mathrm{PVS}_{\mathrm{cr}} = PVS\langle P \rangle = \langle \mathcal{M}^{\mathrm{fin}}(P), \uplus, \supseteq_{\mathsf{SPD}}, \lbag\rbag \rangle$
- PVS-Homomorphismus $\varphi : \mathrm{PVS}_{\mathrm{cr}} \to \mathrm{PVS}_{\mathrm{weighted}}$
  - $\varphi(\top_{\mathrm{cr}}) = \top_{\mathrm{weighted}}$
  - $\varphi(m \cdot_{\mathrm{cr}} n) = \varphi(m) \cdot_{\mathrm{weighted}} \varphi(n)$

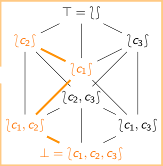


PVS und Morphismen

Institute for Software & Systems Engineering

Etwas systematischer ...

- $\mathrm{PVS}_{\mathrm{cr}} = \mathit{PVS}\langle P \rangle = \langle \mathcal{M}^{\mathrm{fin}}(P), \uplus, \supseteq_{\mathsf{SPD}}, \{\!\{\}\!\} \rangle$
- PVS-Homomorphismus $\varphi : \mathrm{PVS}_{\mathrm{cr}} \to \mathrm{PVS}_{\mathrm{weighted}}$
  - $\varphi(\top_{\mathrm{cr}}) = \top_{\mathrm{weighted}}$
  - $\varphi(m \cdot_{\mathrm{cr}} n) = \varphi(m) \cdot_{\mathrm{weighted}} \varphi(n)$
  - $m \leq_{\mathrm{cr}} n \to \varphi(m) \leq_{\mathrm{weighted}} \varphi(n)$

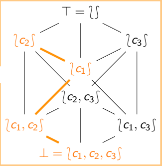


PVS und Morphismen

Institute for Software & Systems Engineering

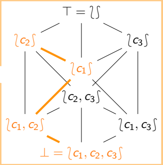
Etwas systematischer ...

- $\mathrm{PVS}_{\mathrm{cr}} = \mathit{PVS}\langle P\rangle = \langle \mathcal{M}^{\mathrm{fin}}(P), \uplus, \supseteq_{\mathrm{SPD}}, \lbag\rbag\rangle$
- PVS-Homomorphismus $\varphi : \mathrm{PVS}_{\mathrm{cr}} \to \mathrm{PVS}_{\mathrm{weighted}}$
  - $\varphi(\top_{\mathrm{cr}}) = \top_{\mathrm{weighted}}$
  - $\varphi(m \cdot_{\mathrm{cr}} n) = \varphi(m) \cdot_{\mathrm{weighted}} \varphi(n)$
  - $m \leq_{\mathrm{cr}} n \to \varphi(m) \leq_{\mathrm{weighted}} \varphi(n)$



- Beispiel: $\varphi(\lbag\rbag) = 0$, $\varphi(\lbag c\rbag \uplus C) = w^{\mathrm{SPD}}(c) + \varphi(C)$

PVS und Morphismen



Etwas systematischer ...

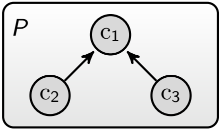
- $\mathrm{PVS}_{\mathrm{cr}} = \mathit{PVS}\langle P\rangle = \langle \mathcal{M}^{\mathrm{fin}}(P), \uplus, \supseteq_{\mathrm{SPD}}, \lbag\rbag \rangle$
- PVS-Homomorphismus $\varphi : \mathrm{PVS}_{\mathrm{cr}} \to \mathrm{PVS}_{\mathrm{weighted}}$
  - $\varphi(\top_{\mathrm{cr}}) = \top_{\mathrm{weighted}}$
  - $\varphi(m \cdot_{\mathrm{cr}} n) = \varphi(m) \cdot_{\mathrm{weighted}} \varphi(n)$
  - $m \leq_{\mathrm{cr}} n \to \varphi(m) \leq_{\mathrm{weighted}} \varphi(n)$

- Beispiel: $\varphi(\lbag\rbag) = 0$, $\varphi(\lbag c \rbag \uplus C) = w^{\mathrm{SPD}}(c) + \varphi(C)$

- Warum überhaupt Morphismen?
  - Ordnung soll bewusst totalisiert werden
  - Datentyp *xy* (z.B. Mengen) wird von Solver/Algorithmus nicht unterstützt
  - $\to$ Benutzer interessiert nicht konkrete Datenstruktur sondern nur die Erhaltung der gewünschten Ordnung

Looking for freedom . . .

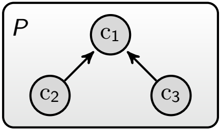
Institute for Software & Systems Engineering



Cat : POSet

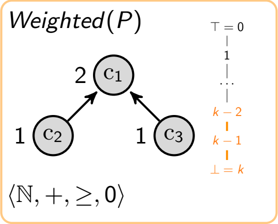
Cat : PVS

Looking for freedom . . .

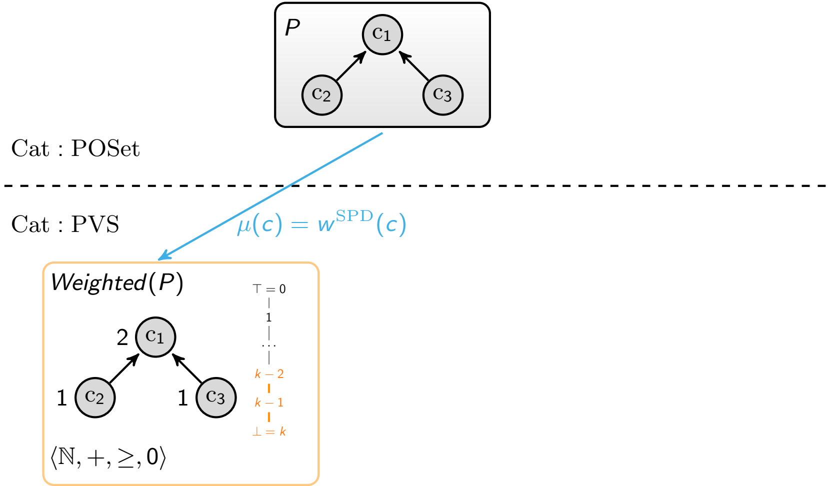


Cat : POSet

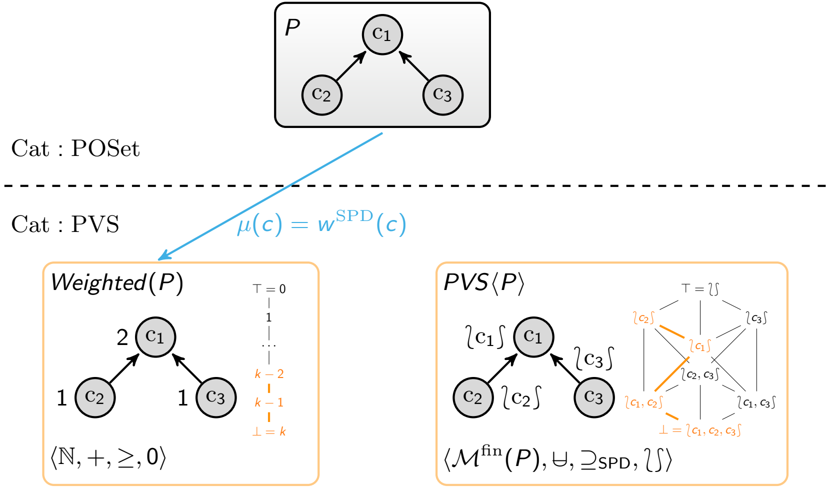
Cat : PVS



Looking for freedom . . .

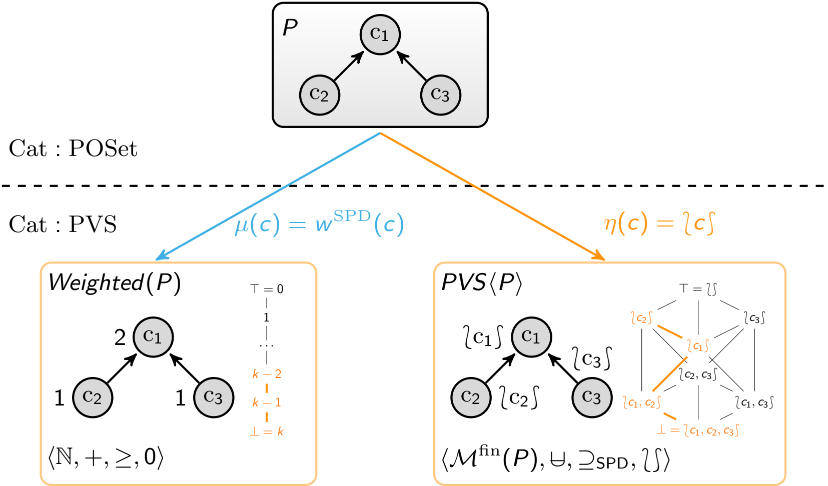


Looking for freedom . . .

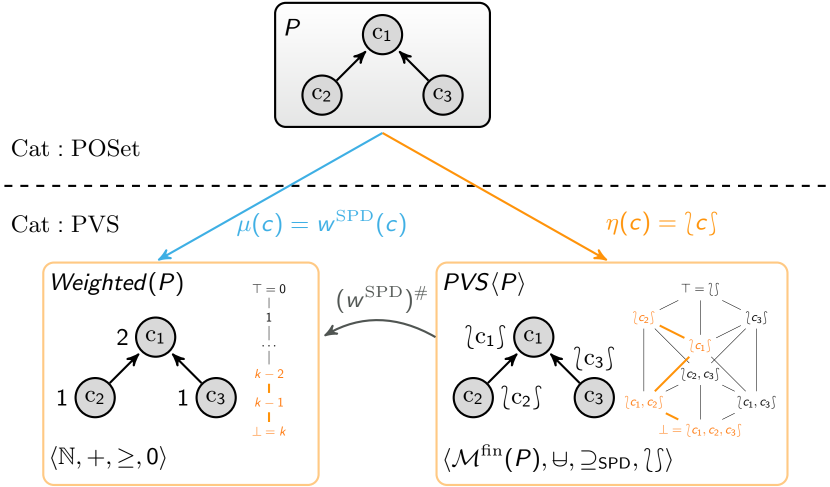


Looking for freedom . . .

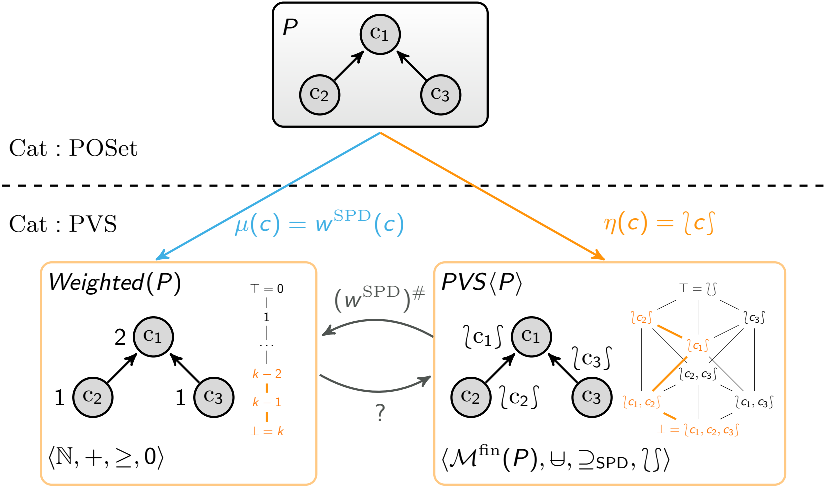
Institute for Software & Systems Engineering



Looking for freedom . . .



Looking for freedom ...



Looking for freedom ...

Institute for
Software & Systems
Engineering

Lemma (PVS-Freiheit (Knapp and Schiendorfer, 2014))

$PVS\langle P \rangle$ is the free partial valuation structure over the partial order P.

$$\begin{tikzcd} P \arrow[r, "\eta_P^{\mathrm{PVS}}"] \arrow[dr, "\varphi"'] & PO(PVS\langle P \rangle) \arrow[d, "PO(\varphi^{\#\mathrm{PVS}})"] \\ & PO(M) \end{tikzcd} \qquad \begin{tikzcd} PVS\langle P \rangle \arrow[d, dashed, "\varphi^{\#\mathrm{PVS}}"] \\ M \end{tikzcd}$$

Freie Konstruktionen

- no junk
- no confusion

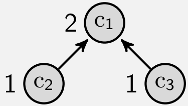
Morphismen in MiniBrass

```
morph ConstraintRelationships -> WeightedCsp: ToWeighted =
  params {
    k = 'mbr.nScs * max(i in 1..mbr.nScs) (mbr.weights[i]) ';
    weights = calculate_cr_weights;
  } in id; % "in" denotes the function applied to each soft constraint

PVS: cr1 = new ConstraintRelationships("cr1") {
   soft-constraint c1: 'x + 1 = y';
   soft-constraint c2: 'z = y + 2';
   soft-constraint c3: 'x + y <= 3';

   crEdges : '[| mbr.c2, mbr.c1 | mbr.c3, mbr.c1 |]';
   useSPD: 'false' ;
};

solve ToWeighted(cr1);
```




```
Solution: x = 1; y = 2; z = 1
Valuations: overall = 1
```

```
c1: 'x + 1 = y';
c2: 'z = y + 2';
c3: 'x + y <= 3';
```

PVS-Kombinationen: Pareto

Institute for Software & Systems Engineering

Mit PVSs M und N können wir das direkte Produkt $M \times N$

$$(m, n) \leq_{M \times N} (m', n') \leftrightarrow m \leq_M m' \land n \leq_N n'$$

bilden. Entspricht der *Pareto*-Ordnung

```
% in MZN-file: var bool: x; var bool: y;
PVS: wcsp1 = new WeightedCsp("wcsp1") {
   soft-constraint c1: 'y = false' ;
   k : '20';
};

PVS: wcsp2 = new WeightedCsp("wcsp2") {
   soft-constraint c1: 'x = false' ;
   k : '20';
};

solve wcsp1 pareto wcsp2; % returns x = false, y = false
```

PVS-Kombinationen: Lex

Außerdem das lexikographische Produkt $M \ltimes N$

$$(m, n) \leq_{M \ltimes N} (m', n') \leftrightarrow (m <_M m') \vee (m = m' \wedge n \leq_N n')$$

Ermöglicht strikte Hierarchien

```
% in MZN-file: var 1..3: x; var 1..3: y;
PVS: cr1 = new CostFunctionNetwork("cr1") {
   soft-constraint c1: 'x' ;
   soft-constraint c2: '3 - y' ;
   k : '20';
};
PVS: cr2 = new CostFunctionNetwork("cr2") {
   soft-constraint c1: 'y' ;
   soft-constraint c2: '3 - x' ;
   k : '20';
};
solve cr1 lex cr2; % returns x = 1, y = 3
% dually cr2 lex cr1 yields x = 3, y = 1
```

References I

Amadio, R. M. and Curien, P.-L. (1998).
Domains and Lambda-Calculi.
Cambridge Tracts in Theoretical Computer Science 46. Cambridge University Press.

Knapp, A. and Schiendorfer, A. (2014).
Embedding Constraint Relationships into C-Semirings.
Technical Report 2014-03, Institute for Software and Systems Engineering, University of Augsburg.
`http://opus.bibliothek.uni-augsburg.de/opus4/frontdoor/index/index/docId/2684`.

Knapp, A., Schiendorfer, A., and Reif, W. (2014).
Quality over Quantity in Soft Constraints.
In *Proc. 26th Int. Conf. Tools with Artificial Intelligence (ICTAI'2014)*, pages 453–460.

References II

Schiendorfer, A., Steghöfer, J.-P., Knapp, A., Nafz, F., and Reif, W. (2013).
Constraint Relationships for Soft Constraints.
In Bramer, M. and Petridis, M., editors, *Proc. 33rd SGAI Int. Conf. Innovative Techniques and Applications of Artificial Intelligence (AI'13)*, pages 241–255. Springer.