Model Expansion in the presence of Function Symbols using Constraint Programming

Broes De Cat – Bart Bogaerts – Jo Devriendt – Marc Denecker

KRR group - KU Leuven

Knowledge Representation and Reasoning

- Separate knowledge from computation
  - Study of knowledge involved in applications
  - And the general tasks for which it is used

- Representation: rich, declarative logic
- Reasoning: efficient inference engines

- Aims:
  - Recognize similar applications => reuse knowledge
  - Recognize similar tasks => reuse inference engines

IDP³

Knowledge

- Vocabulary
- Theory
- Structure

Inferences

- Model expansion
- Querying
- Deduction
- Visualization
- Model revision
- …

Procedural interface

Model expansion

- Model generation
  - Find models of a theory T
- Model expansion
  - Given a partial structure S

- Related to
  - Answer Set generation (ASP)
  - CSP solving (CP)
  - SAT

- State-of-the-art approach
  ground/unroll & search / BnB

Terminology

FO/SAT	CP	ASP
Sentence	Constraint	Rule/constraint
Theory	Set of constraints	Logic program
Structure	Data	Facts
Model	Solution	Answer set
Function	Variable	Function
Variable	(variable)	Variable

MX with Function Symbols using CP?

- Modelling <> encoding
  - KR languages => (more natural) modelling
  - SAT/ASP => clause/nogood learning
  - CP => concise, propagation

- Grounding to SAT/ASP explodes size

  => Function symbols give rise to CP constraints

  => Search combining SAT – ASP – CP

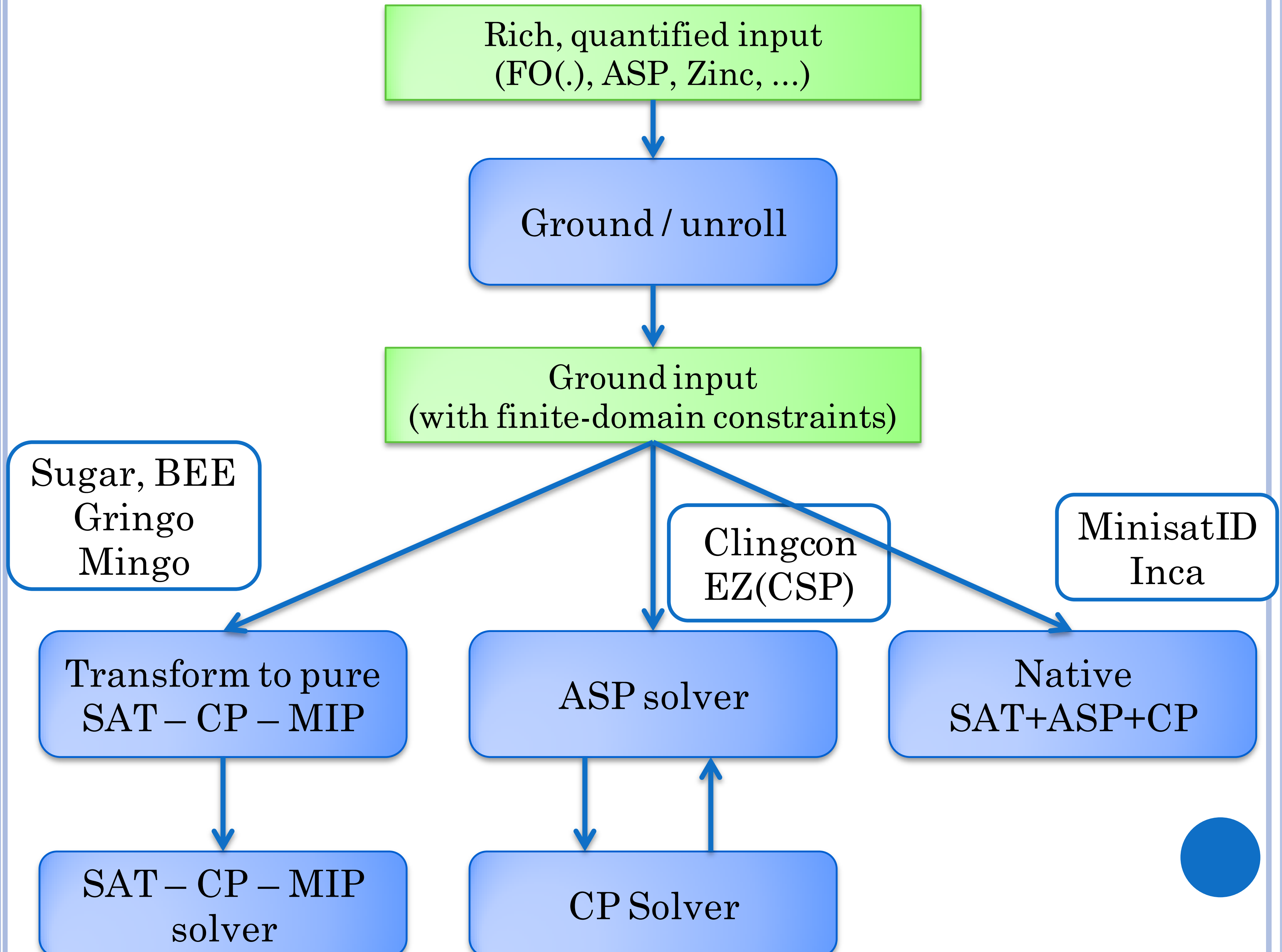
Take away message

Model (+ improve engine) <> encode

=> Function symbols give rise to CP constraints

=> Search combining SAT – ASP – CP

➔ **Maintain structure as long as possible**


Rich, quantified input
(FO(.), ASP, Zinc, …)
Ground / unroll
Ground input
(with finite-domain constraints)
Sugar, BEE
Gringo
Mingo
Clingcon
EZ(CSP)
MinisatID
Inca
Transform to pure
SAT – CP – MIP
ASP solver
Native
SAT+ASP+CP
SAT – CP – MIP
solver
CP Solver

The language FO(.)^{IDP}

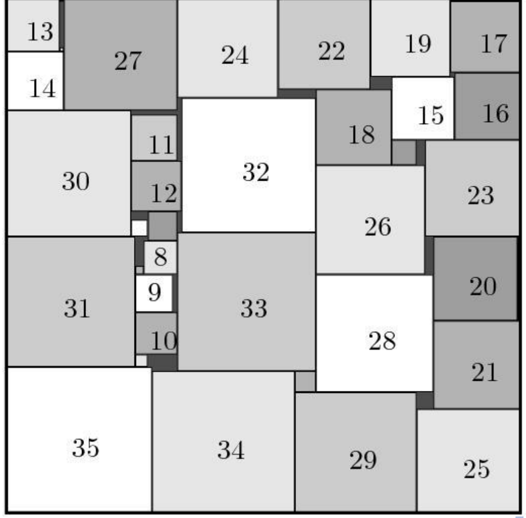
- Full first-order logic
- Type system (e.g. P($type_1$, ..., $type_n$))
  - Type = set of domain elements
  - Atoms outside type => false
- Definitions (set of rules)

  $\{\forall x[node] : r(x) \leftarrow start(x) \vee (\exists y[node] : edge(y, x) \wedge r(y))\}$
- Aggregates (weight/card rules)

  $sum(\{x[node]\ y[node] : edge(x, y) : weight(x, y)\})$
- Partial functions

  $\neg edge(x, y) \Rightarrow \neg denoting(weight(x, y))$

2-D Square Packing theory



$$\forall id_1\ id_2 : id_1 \neq id_2 \Rightarrow noOverlap(id_1, id_2)$$

$$\left\{ \begin{array}{ll} \forall id_1\ id_2 : & noOverlap(id_1, id_2) \leftarrow \\ & \quad leftof(id_1, id_2) \vee leftof(id_2, id_1) \\ & \quad \vee below(id_1, id_2) \vee below(id_2, id_1) \end{array} \right\}$$

$$\left\{ \begin{array}{l} \forall id_1\ id_2 : leftof(id_1, id_2) \leftarrow \\ \quad pos_x(id_1) + size(id_1) \leq pos_x(id_2) \end{array} \right\}$$

$$pos_x(largest) = 0 \wedge pos_y(largest) = 0$$

$$\{\forall id_1 : largest = id_1 \leftarrow \forall id_2 : size(id_2) \geq size(id_2)\}$$

Grounding

- Preprocessing:
  - **Unnest** functions:

    P(f(x)) ==> !y: f(x)=y => P(y)
  - **Graph** functions:

    f(x)=y ==> F(x,y), #{y: F(x,y)}=1

- Rewrite rules
  - **Instantiate** variable
    - Replace with values in the domain
  - **Evaluate** formula/term
    - Use structure as soon as possible
  - **Introduce** new atom/constant
    - To normalize on-the-fly

Grounding

- Priority on rules
  - Instantiate top-down, depth-first
    - Less memory intensive
  - Evaluate as-soon-as-possible
  - Introduce atom/constant only on context change
  - Use subformula to reduce domain size

    !x[1,10]: x<3 => P(x) [Wittockx, 2010]
  - …

Supported symbols

- Allowing functions results “constraints”

  $pos_x(largest)=0$

  $pos_x(id) + size(id) < pos_x(id')$

- Symbols supported by the solver
  - Arithmetic
  - Element constraint
  - Binary comparison
  - Uninterpreted functions
  - …

- Solver provides

  list S of supported symbols + context

Grounding

- Preprocessing:
  - **Unnest** functions **not in S**

    P(f(x)) ==> !y: f(x)=y => P(y)
  - **Graph** functions **not in S**

    f(x)=y ==> F(x,y), #{y: F(x,y)}=1

- Rewrite rules
  - **Instantiate** variable
    - Replace with values in the domain
  - **Evaluate** formula/term
    - Use structure as soon as possible
  - **Introduce** new atom/constant
    - To normalize on-the-fly

Ground theory

- Grounding can be passed to any solver supporting S

- Now assume S are **all** functions symbols
  - Results in full ground FO(.)
    - Definitions
    - Aggregates
    - Nested functions

- Can we build a solver for this?

Ground FO(.)

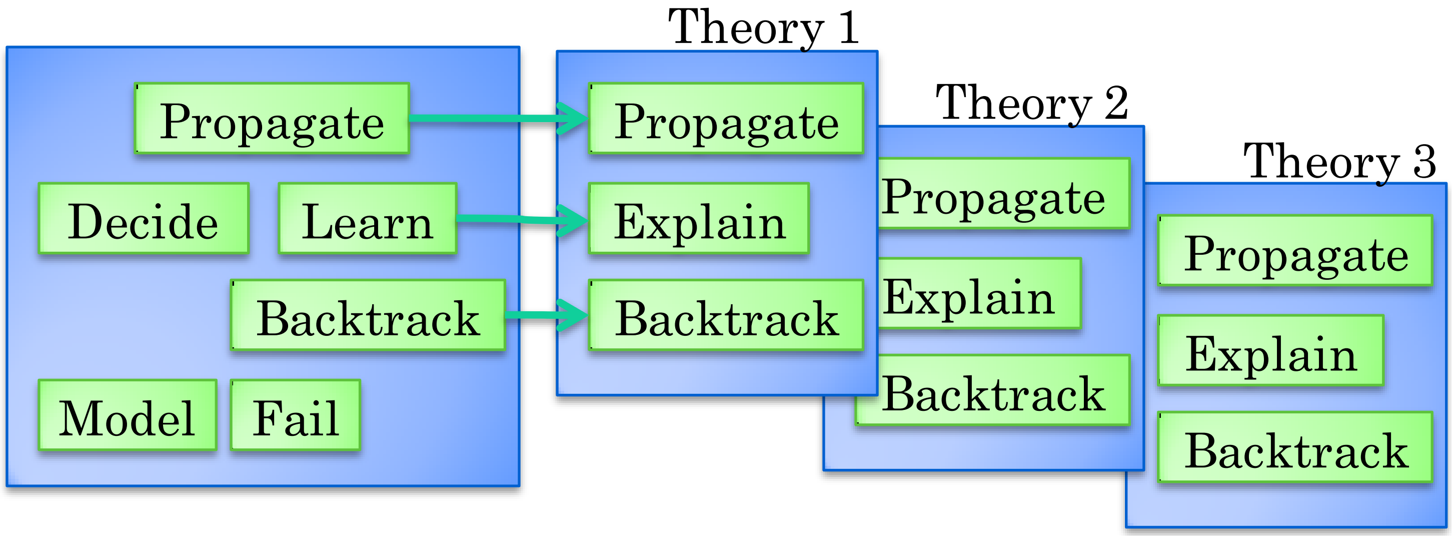
$$L_1 \vee \ldots \vee L_n.$$
$$Q(\bar{c}).$$
$$f(\bar{c}) \sim c'.$$
$$agg(\{L_1 : c_1\} \cup \cdots \cup \{L_n : c_n\}) \sim c'.$$

$$P(\bar{c}) \leftarrow L_1 \wedge \ldots \wedge L_n.$$
$$P(\bar{c}) \leftarrow L_1 \vee \ldots \vee L_n.$$
$$P(\bar{c}) \leftarrow Q(\bar{c}').$$
$$P(\bar{c}) \leftarrow f(\bar{c}) \sim c'.$$
$$P(\bar{c}) \leftarrow agg(\{L_1 : c_1\} \cup \cdots \cup \{L_n : c_n\}) \sim c'.$$

SAT Modulo Theories
(Nieuwenhuys et al.)

DPLL(T) architecture

Add new propagation mechanisms to SAT **with learning**



Lazy Clause Generation *[Stuckey et al., 2008]*

- Constraint = large set of implications
- Explanation = applied implications

- LCG:
  - Encode functions as Boolean atoms
  - Whenever a constraint would propagate
    instead, add a clause reprsenting it

E.g.: c1 = c2
  if c1<5, then would propagate c2<5
  instead, add clause $T_{c1<5} \Rightarrow T_{c2<5}$

  - Or lazier: build it when the explanation is requested

Lazy Grounding [De Cat et al., 2008]

- Module for constraint c
  - Set of (quantified) sentences
  - + intelligent watches on when to ground what parts

  Order encoding of c, domain D

$$\forall x[D - d_n] T_{c \leq x} \Rightarrow T_{c \leq next(x)}$$
$$\forall x[D - d_1] T_{c > x} \Rightarrow T_{c > prev(x)}$$

- Note: requires on-the-fly addition of atoms, variables and constraints!

Comparison constraint

- Comparison $P \Leftrightarrow c \leq c'$

$$
\begin{aligned}
&\forall x[D \cup D']T_{c \leq x} \wedge T_{c' \geq x} \Rightarrow P.\\
&\forall x[D \cup D']T_{c > x} \wedge T_{c' < x} \Rightarrow \neg P.\\
&\forall x[D]T_{c' \leq x} \wedge P \Rightarrow \Gamma_{c \leq x}.\\
&\forall x[D]T_{c' \geq x} \wedge \neg P \Rightarrow T_{c > x}.\\
&\forall x[D']T_{c \geq x} \wedge P \Rightarrow T_{c' \geq x}.\\
&\forall x[D']T_{c \leq x} \wedge \neg P \Rightarrow T_{c' < x}.
\end{aligned}
$$

Nested terms
Partial functions

- Nested terms $P \Leftrightarrow f(\bar{c}) \leq c'$

$$\forall \bar{x} \in dom_{\bar{c}} : T_{\bar{c}=\bar{x}} \Rightarrow (P \Leftrightarrow f(\bar{x}) \leq c')$$

- Partial functions
  - Encoding: one additional atom "denoting_c"
  - All other constraints:
    - As if conjoined with "denoting" of all their variables

Search

- **Decide**
- **UP**
- **Learn**
- **Encode_function**
- **Propagate_compare, Explain_compare**
- **Encode_aggregate** (bounds propagation)
- **Encode_nested**
- **Definition**

  Completion, Unfounded, Wellfounded

Search: SAT+ASP+CP

Choice
Learn
Backtrack
CDCL

Propagate | Explain

Clauses

Order encoding $a \in [1,10000]$

Completion

$P \Leftrightarrow a < b$

Unfounded sets

$P \Leftrightarrow Q(a,b,c)$

Wellfounded

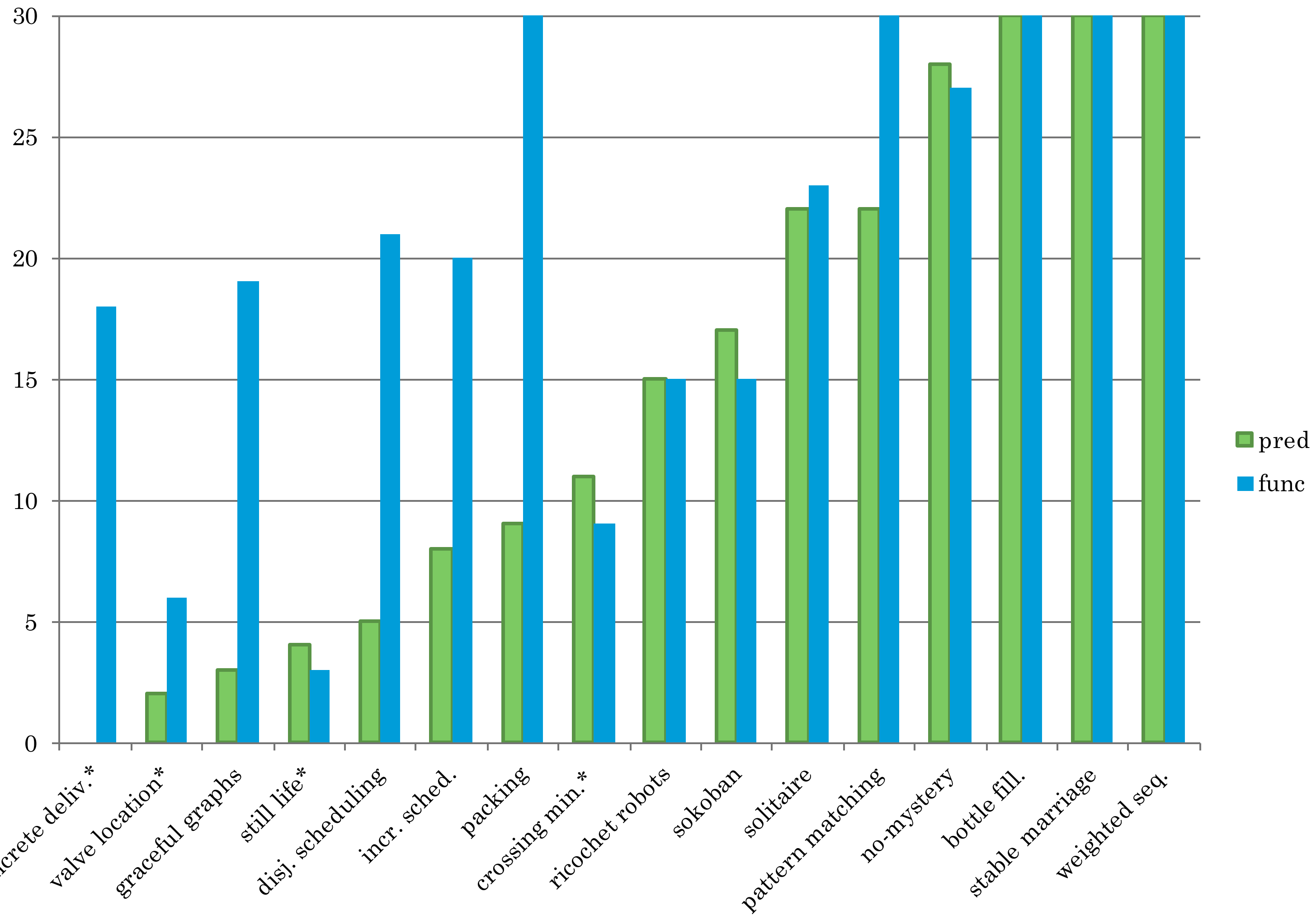
$P \Leftrightarrow a+b+c+d = n$

Pseudo-boolean
Aggregates

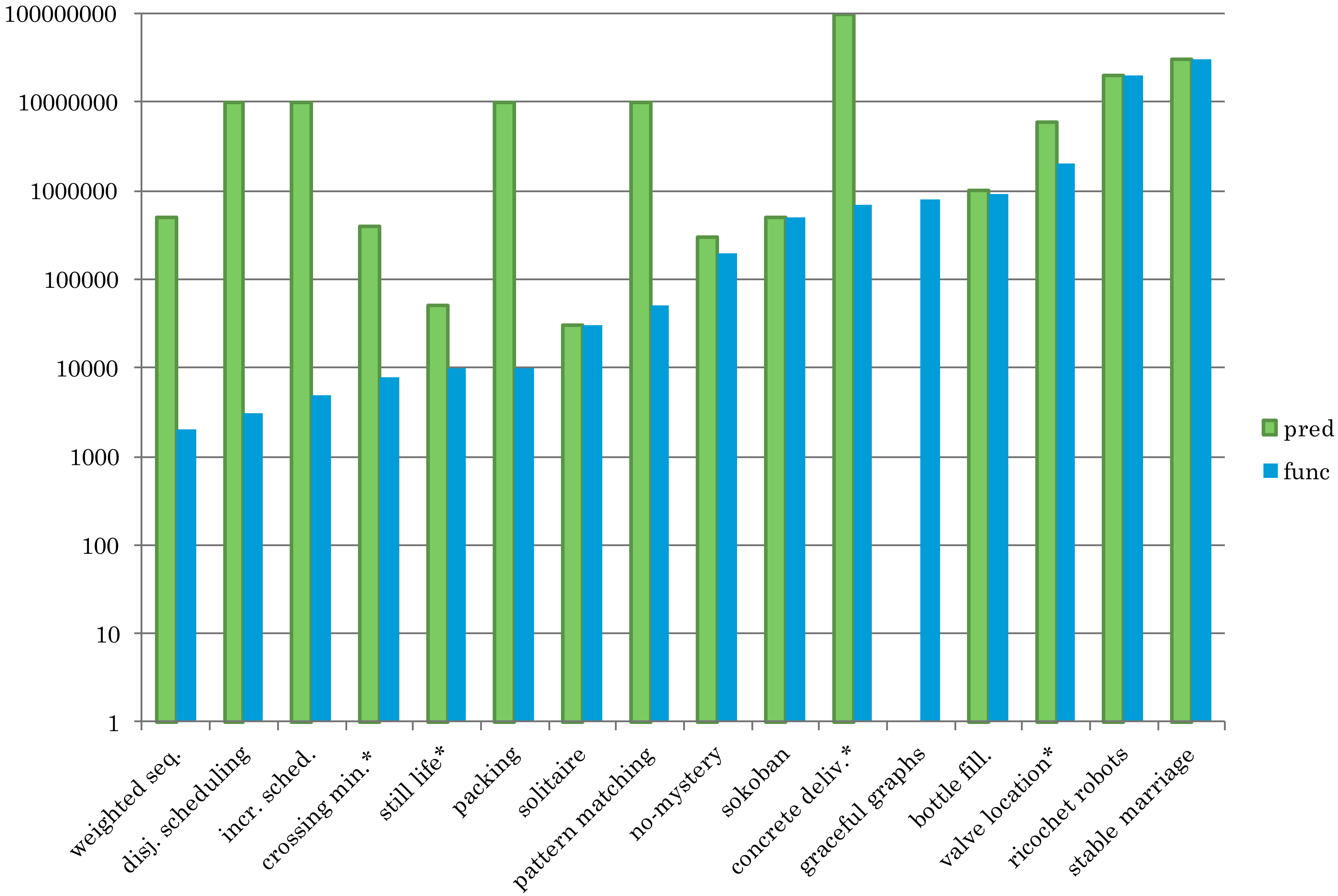
Symmetry propagation

Minimize

Results
instances solved within timeout



Results
Grounding size (#atoms)



Results

- MiniZinc
  - Solver-independent CSP language

- MiniZinc challenge
  - Performance?

- Single best solver in MiniZinc portfolio
  [Amadini, Arxiv 1308.0227]

Learning Deterministic Finite state automata (DFA)

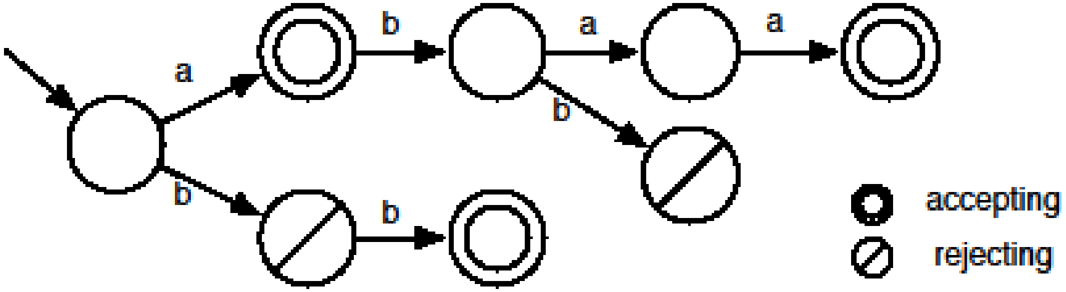
- Grammar learning
  - Given a sequence of finite labeled strings

ab
abb

abb
a
abaa

/
ab
aba

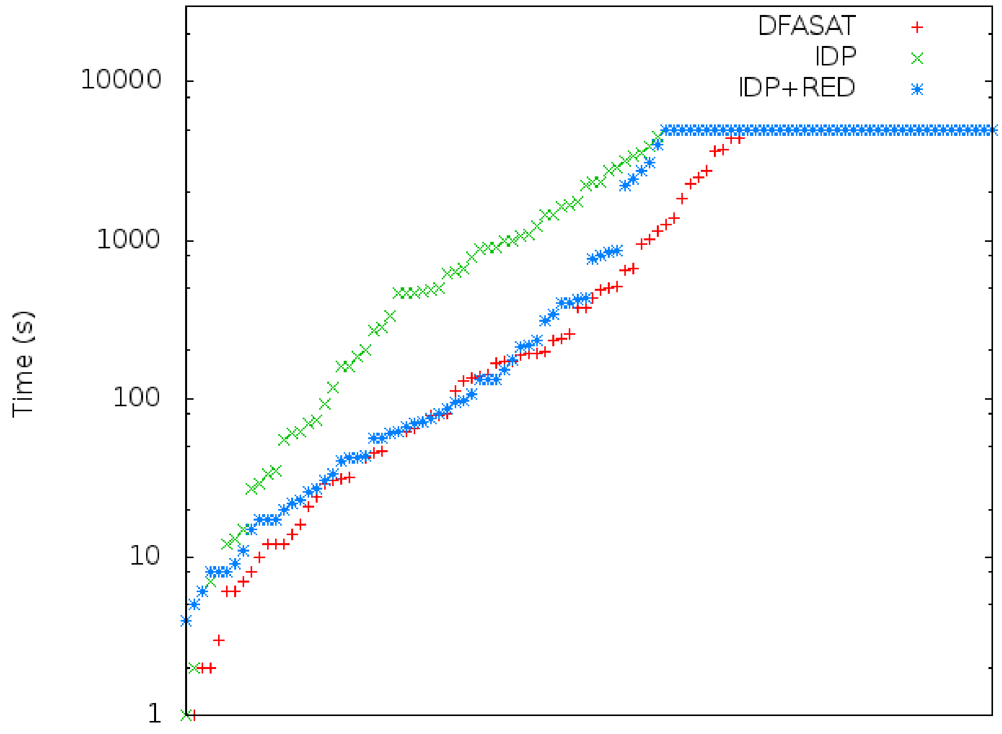
  - Derive matching automaton (trivial)



  - Improve it by reducing the number of nodes

DFA LEARNING DECISION PROBLEM

- 2000 lines C++ => 40 lines FO(.)^{IDP}
- Performance



Conclusion

- Configurable, efficient grounding algorithm
- Functions in logic give rise to constraint in the CP sense
- Search algorithm for full ground FO(.)
  - Combining SAT/ASP learning with CP propagation
  - First open-source LCG solver

- People might use function for modelling?

  Implicit function detection and rewriting

  [De Cat et al., ICLP 2013]

DTAI.CS.KULEUVEN.BE/KRR/SOFTWARE

IDP^3

MINISAT ID inside

IDP$_E$

IDP$_{DRAW}$

CONFIG ID