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
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

<table>
<thead>
<tr>
<th>FO/SAT</th>
<th>CP</th>
<th>ASP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sentence</td>
<td>Constraint</td>
<td>Rule/constraint</td>
</tr>
<tr>
<td>Theory</td>
<td>Set of constraints</td>
<td>Logic program</td>
</tr>
<tr>
<td>Structure</td>
<td>Data</td>
<td>Facts</td>
</tr>
<tr>
<td>Model</td>
<td>Solution</td>
<td>Answer set</td>
</tr>
<tr>
<td>Function</td>
<td>Variable</td>
<td>Function</td>
</tr>
<tr>
<td>Variable</td>
<td>(variable)</td>
<td>Variable</td>
</tr>
</tbody>
</table>
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

Transform to pure SAT – CP – MIP

SAT – CP – MIP solver

Clingcon, EZ(CSP)

ASP solver

CP Solver

Native SAT+ASP+CP

MinisatID, Inca
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)
  \[ \forall x[node] : r(x) \leftarrow \text{start}(x) \lor (\exists y[node] : \text{edge}(y, x) \land r(y)) \]  
- Aggregates (weight/card rules)
  \[ \text{sum} (\{ x[node] \ y[node] : \text{edge}(x, y) : \text{weight}(x, y) \}) \]
- Partial functions
  \[ \neg \text{edge}(x, y) \Rightarrow \neg \text{denoting}(\text{weight}(x, y)) \]
2-D Square Packing Theory

\[ \forall id_1 \ id_2 : id_1 \neq id_2 \Rightarrow noOverlap(id_1, id_2) \]

\[ \begin{align*}
\forall id_1 \ id_2 : & \quad noOverlap(id_1, id_2) \leftarrow \\
& \quad leftof(id_1, id_2) \lor leftof(id_2, id_1) \\
& \quad \lor below(id_1, id_2) \lor below(id_2, id_1)
\end{align*} \]

\[ \begin{align*}
\forall id_1 \ id_2 : & \quad leftof(id_1, id_2) \leftarrow \\
& \quad pos_x(id_1) + size(id_1) \leq pos_x(id_2)
\end{align*} \]

\[ pos_x(largest) = 0 \land pos_y(largest) = 0 \]

\[ \{ \forall id_1 : largest = id_1 \leftarrow \forall id_2 : size(id_1) \geq size(id_2) \} \]
GROUNDING

- **Preprocessing:**
  - **Unnest** functions:
    
    \[ P(f(x)) \implies !y: f(x)=y \implies P(y) \]
  - **Graph** functions:
    
    \[ f(x)=y \implies F(x,y), \#\{y: F(x,y)\}=1 \]

- **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
    \[
    \neg x[1,10]: x<3 \Rightarrow P(x) \quad \text{[Wittockx, 2010]}
    \]
  - ...

Supported symbols

- Allowing functions results “constraints”
  \[ \text{pos}_x(\text{largest})=0 \]
  \[ \text{pos}_x(\text{id}) + \text{size}(\text{id}) < \text{pos}_x(\text{id}') \]

- 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**
    \[ P(f(x)) \implies !y: f(x)=y \implies P(y) \]
  - Graph functions **not in S**
    \[ f(x)=y \implies F(x,y), \#\{y: F(x,y)\}=1 \]

- 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(.)

\[ L_1 \lor \ldots \lor L_n. \]
\[ Q(\overline{c}). \]
\[ f(\overline{c}) \sim c'. \]
\[ \text{agg} \left( \{L_1 : c_1\} \cup \cdots \cup \{L_n : c_n\} \right) \sim c'. \]

\[ P(\overline{c}) \leftarrow L_1 \land \ldots \land L_n. \]
\[ P(\overline{c}) \leftarrow L_1 \lor \ldots \lor L_n. \]
\[ P(\overline{c}) \leftarrow Q(\overline{c}'). \]
\[ P(\overline{c}) \leftarrow f(\overline{c}) \sim c'. \]
\[ P(\overline{c}) \leftarrow \text{agg} \left( \{L_1 : c_1\} \cup \cdots \cup \{L_n : c_n\} \right) \sim c'. \]
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 representing it
  
  E.g.: $c1 = c2$
  
  if $c1 < 5$, then would propagate $c2 < 5$
  instead, add clause $Tc1 < 5 \Rightarrow Tc2 < 5$

- 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 \( P \Leftrightarrow c \leq c' \)

\[
\forall x [D \cup D'] \left( T_{c \leq x} \land T_{c' \geq x} \Rightarrow P \right).
\forall x [D \cup D'] \left( T_{c > x} \land T_{c' < x} \Rightarrow \neg P \right).
\forall x [D] \left( T_{c' \leq x} \land P \Rightarrow T_{c \leq x} \right).
\forall x [D] \left( T_{c' \geq x} \land \neg P \Rightarrow T_{c > x} \right).
\forall x [D'] \left( T_{c \geq x} \land P \Rightarrow T_{c' \geq x} \right).
\forall x [D'] \left( T_{c \leq x} \land \neg P \Rightarrow T_{c' < x} \right).
\]
Nested terms

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

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

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

CDCL

Choice
Learn
Backtrack

Clauses
Completion
Unfounded sets
Wellfounded
Pseudo-boolean Aggregates

Propagate
Explain

Order encoding $a \in [1,10000]$ 
$P \Leftrightarrow a < b$
$P \Leftrightarrow Q(a,b,c)$
$P \Leftrightarrow a+b+c+d = n$
Symmetry propagation

Minimize
RESULTS

# INSTANCES SOLVED WITHIN TIMEOUT
## Results

### Grounding size (#atoms)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>weighted seq-disj, sched</td>
<td></td>
</tr>
<tr>
<td>incr sched</td>
<td></td>
</tr>
<tr>
<td>crossing min. *</td>
<td></td>
</tr>
<tr>
<td>still life *</td>
<td></td>
</tr>
<tr>
<td>packing</td>
<td></td>
</tr>
<tr>
<td>solitaire</td>
<td></td>
</tr>
<tr>
<td>pattern matching</td>
<td></td>
</tr>
<tr>
<td>no-mystery</td>
<td></td>
</tr>
<tr>
<td>sokoban</td>
<td></td>
</tr>
<tr>
<td>concrete deliv. *</td>
<td></td>
</tr>
<tr>
<td>graceful graphs</td>
<td></td>
</tr>
<tr>
<td>bottle fill</td>
<td></td>
</tr>
<tr>
<td>valve location</td>
<td></td>
</tr>
<tr>
<td>ricochet robots</td>
<td></td>
</tr>
<tr>
<td>stable marriage</td>
<td></td>
</tr>
</tbody>
</table>

![Bar chart showing results for various problems](chart.png)
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

  - Derive matching automaton (trivial)

  ![Diagram of a DFA with transitions labeled 'a' and 'b' and accepting and rejecting states]

  - 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]