DETECTION AND EXPLOITATION OF FUNCTIONAL DEPENDENCIES FOR MODEL GENERATION

Broes De Cat and Maurice Bruynooghe
KRR group - KU Leuven
INTRODUCTION

- Model generation
  - Find models of a set of logical sentences (cfr. ASP, CSP)

- State-of-the-art approach
  ground & solve

- One direction of research
  - Add new language constructs to increase efficiency
  reducing grounding size
  increase search performance
CASP / ASP MODULO CP

Example:
- Find 2 numbers between m and m’ with product p
- Basic predicate logic:
  \[ \exists x : n_1(x) \land m \leq x \leq m \]
  \[ \exists x : n_2(x) \land m \leq x \leq m \]
  \[ \forall x \ x' : n_1(x) \land n_1(x') \Rightarrow x = x' \]
  \[ \forall x \ x' : n_2(x) \land n_2(x') \Rightarrow x = x' \]
  \[ \forall x \ x' : n_1(x) \land n_2(x') \Rightarrow x \times x' = p \]
**Introduction: CASP / ASP modulo CP**

- **Example:**
  - Find 2 numbers between \( m \) and \( m' \) with product \( p \)
  - Basic predicate logic:
    
    \[
    \exists x : n_1(x) \land m \leq x \leq m \\
    \exists x : n_2(x) \land m \leq x \leq m \\
    \forall x \ x' : n_1(x) \land n_1(x') \Rightarrow x = x' \\
    \forall x \ x' : n_2(x) \land n_2(x') \Rightarrow x = x' \\
    \forall x \ x' : n_1(x) \land n_2(x') \Rightarrow x \times x' = p
    \]
  - **Clingcon:**
    
    \[
    \text{var}(n_1,n_2). \\
    \text{domain}(m..m'). \\
    n_1 \times n_2 = p.
    \]
  - **FO(.)^{IDP}:**
    
    \[
    n_1: \text{int}, n_2: \text{int}. \\
    m=\ll n_1=\ll m'. \quad m=\ll n_2=\ll m'. \\
    n_1 \times n_2 = p.
    \]
Ground input (with finite-domain constraints)

Sugar, BEE, Gringo, Mingo

Transform to pure SAT – ASP – MIP

SAT – ASP – MIP solver

Clingcon, EZ(CSP)

ASP solver

CP Solver

Native SAT+ASP+CP

Inca, MinisatID
Automated detection and rewriting

- Increased performance
  - Burden on the **modeler**!
    => One of the challenges of ASP (cfr. Torsten)

- Why is it difficult to use those new constructs?
  - Unawareness
    - Of property (symmetry, alldifferent, function, ...)
    - Of performance gain
  - Property does not hold for all instances
  - Modeling preference
  - Automatic transformation from another language
    E.g. ASP-Core-2

- The modeler should not have to care!
**Knowledge**

- Vocabulary
- Theory
- Structure

**Inferences**

- Generate models
- Query
- Verify/prove
- Visualize
- Debug
- ...

**Procedural interface**
THE LANGUAGE FO(.)\text{IDP}

- First-order logic
  - Explicit universal and existential quantification

- Strongly typed (e.g. P(t_1, ..., t_n) )
  - Type is set of domain elements
  - Atoms are false outside their type

- Definitions (set of rules)
  - Only defined symbols are minimized, per definition
    \{\forall x[node] : r(x) \leftrightarrow start(x) \lor (\exists y[node] : edge(y, x) \land r(y))\}

- Aggregates (weight/card rules)
  \[\text{sum}\{x[node] \ y[node] : edge(x, y) : weight(x, y)\}\]
The language FO(.)^{IDP}

- Functions symbols: $f(t_1, ..., t_n):t_{out}$
  - By default intensional / non-Herbrand
  - Partial/total

- Defined function symbols

\[
\begin{align*}
    fib(1) &= 0. \\
    fib(2) &= 1. \\
    \forall x[node]: fib(x) &= fib(x - 1) + fib(x - 2) \quad \leftarrow x > 2
\end{align*}
\]

- Transformational semantics: transform into graph Fib/2

- (Constructed functions)
CASP in FO(.)\textsuperscript{IDP}

Through functions!
- Previous grounding algorithm:
  - Replace functions by their graph
- Expressions with functions can be interpreted as constraint atoms
  \[ n_1 \cdot n_2 = p \quad n_1 \leq m_1 \quad , \ldots \]
  - Product constraint
  - Comparison constraint
  - ...
- Don’t remove functions during grounding

[De Cat et al., accepted ICTAI2013]
2-D SQUARE PACKING

\[ \forall id : \exists !(x\ y) : \text{pos}(id, x, y) \]
\[ \forall id : \exists !s : \text{size}(id, s) \]

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

\[ \forall id_1\ id_2 : \begin{cases} 
\text{noOverlap}(id_1, id_2) \leftarrow \\
\text{leftof}(id_1, id_2) \lor \text{leftof}(id_2, id_1) \\
\text{below}(id_1, id_2) \lor \text{below}(id_2, id_1) 
\end{cases} \]

\[ \forall id_1\ id_2 : \begin{cases} 
\text{leftof}(id_1, id_2) \leftarrow \\
\exists x_1\ y_1\ x_2\ y_2\ s_1 : \text{pos}(id_1, x_1, y_1) \land \text{size}(id_1, s_1) \\
\land \text{pos}(id_2, x_2, y_2) \land x_1 + s_1 \leq x_2 
\end{cases} \]

\[ \forall id_1\ id_2 : \begin{cases} 
\text{below}(id_1, id_2) \leftarrow \\
\exists x_1\ y_1\ x_2\ y_2\ s_1 : \text{pos}(id_1, x_1, y_1) \land \text{size}(id_1, s_1) \\
\land \text{pos}(id_2, x_2, y_2) \land y_1 + s_1 \leq y_2 
\end{cases} \]
AUTOMATED DETECTION AND REWRITING

- Approach to improve model generation

![Diagram showing automated detection and rewriting process]

theory $T$ → detect → rewrite

structure $I$ → detect → rewrite → ground & solve

model $M \models \mathcal{T}, \mathcal{I} \subseteq M$
FUNCTIONAL DEPENDENCY

- Given an n-ary predicate $P(t_1,\ldots,t_n)$ and theory $T$
- i-th argument of $P$ *depends functionally in* $T$ on some other arguments $S$ of $P$ if *in every model*
  - A true $P$ atom *exists* for every $S$ instantiation
  - Instantiation for $i$ is *unique* for every $S$ instantiation

- E.g. $pos(Id,X,Y) \Rightarrow d<pos,\{1\},2>$?
  - for each id, some atom $pos(id,x,y)$ is true
  - for each id, there are no 2 such true atoms with different $x$

- Dependency can be *partial*
PART 1: REWRITING FOR d<\text{p,s,i}> in T

- Introduce new symbols
  - function symbol $f(\text{types}|_S):t_i$
    - (partial if dependency is)
  - “remainder” predicate symbol $P_r(\text{types}\setminus t_i)$

$$\text{pos}(Id,X,Y) + d<\text{pos},{1},2> \implies fx(Id):X \text{ and posr}(Id,Y)$$

- Replace atoms $\text{pos}(id,x,y)$ with $\text{pos}_x(id)=x \land \text{pos}_r(id,y)$

- Useful property:
  - $S + i$ covers all arguments $\implies P_r$ always true
**FUNCTIONAL PACKING**

- Introduce new function symbols \((\text{pos}_x, \text{pos}_y, \text{size})\)

\[
\forall id : \exists!(x \ y) : \text{pos}_x(id) = x \land \text{pos}_y(id) = y
\]

\[
\forall id : \exists!s : \text{size}(id) = s
\]

\[
\forall id_1 \ id_2 : \text{leftof}(id_1, id_2) \leftarrow
\exists x_1 \ y_1 \ x_2 \ y_2 \ s_1 : \text{pos}_x(id_1) = x
\land \text{pos}_y(id_1) = y_1 \land \text{size}(id_1) = s_1
\land \text{pos}_x(id_2) = x_2 \land \text{pos}_y(id_2) = y_2 \land x_1 + s_1 \leq x_2
\]
FUNCTIONAL PACKING

- Replace assigned variables

\[ \forall id : \exists! (x \ y) : pos_x(id) = x \land pos_y(id) = y \]
\[ \forall id : \exists! s : size(id) = s \]
\[ \forall id_1 \ id_2 : \text{leftof}(id_1, id_2) \leftarrow \]
\[ \exists x_1 \ y_1 \ x_2 \ y_2 \ s_1 : pos_x(id_1) = x \]
\[ \land pos_y(id_1) = y_1 \land size(id_1) = s_1 \]
\[ \land pos_x(id_2) = x_2 \land pos_y(id_2) = y_2 \]
\[ \land pos_x(id_1) + size(id_1) \leq pos_x(id_2) \]
FUNCTIONAL PACKING

- Drop variables only used in assignment

\[ \forall \text{id} : \exists!(x \ y) : \text{true} \land \text{true} \]
\[ \forall \text{id} : \exists!\text{s} : \text{true} \]
\[ \forall \text{id}_1 \ \text{id}_2 : \ \text{leftof}(\text{id}_1, \text{id}_2) \leftarrow \]
\[ \exists x_1 \ y_1 \ x_2 \ y_2 \ s_1 : \text{true} \]
\[ \land \text{true} \land \text{true} \]
\[ \land \text{true} \land \text{true} \]
\[ \land \text{pos}_x(\text{id}_1) + \text{size}(\text{id}_1) \leq \text{pos}_x(\text{id}_2) \]
FUNCTIONAL PACKING

- Drop quantified, unused variables

\[ \forall id_1 \; id_2 : \; leftof(id_1, id_2) \leftarrow \]
\[ pos_x(id_1) + size(id_1) \leq pos_x(id_2) \]

- Grounding size of the whole specification:
  \[ |id|^2 \text{ instead of } |width|^2 |height|^2 |id|^2 \]
**FUNCTIONAL PACKING**

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

\[ \forall id_1 \ id_2 : \quad \text{noOverlap}(id_1, id_2) \leftarrow \]

\[ \text{leftof}(id_1, id_2) \lor \text{leftof}(id_2, id_1) \]

\[ \lor \text{below}(id_1, id_2) \lor \text{below}(id_2, id_1) \]

\[ \forall id_1 \ id_2 : \quad \text{leftof}(id_1, id_2) \leftarrow \]

\[ \text{pos}_x(id_1) + \text{size}(id_1) \leq \text{pos}_x(id_2) \]

\[ \forall id_1 \ id_2 : \quad \text{below}(id_1, id_2) \leftarrow \]

\[ \text{pos}_y(id_1) + \text{size}(id_1) \leq \text{pos}_y(id_2) \]

- Grounding size of the whole specification:
  \[ |id|^2 \text{ instead of } |\text{width}|^2 |\text{height}|^2 |id|^2 \]
Rewriting: Model-Equivalence?

- Original symbols? Input-output?
  - Input: structure provides symbols $pos_{ct}$ and $pos_{cf}$
    
    \[
    \forall id \ x \ y : pos_{ct}(id, x, y) \Rightarrow pos_x(id) = x \land pos_y(id) = y
    \]
    
    \[
    \forall id \ x \ y : pos_{cf}(id, x, y) \Rightarrow pos_x(id) \neq x \land pos_y(id) \neq y
    \]
  
  - Output: define original symbol
    
    \[
    \{ \forall id : pos(id, pos_x(id), pos_y(id)) \leftarrow true \}
    \]

- Grounding increases again? No!
  - Lifted unit propagation up front
  - Output definition evaluation afterwards
PART 2: DETECTION

- Given a symbol $P(t_1, ..., t_n)$ in theory $T$
  - Does $d<P,S,i>$ hold in $T$?

- $d<P,S,i>$ holds iff
  - A true $P$ atom exists for every $S$ instantiation
    \[
    \forall x_{s_1} \ldots x_{s_n} : \exists x_{s'_1} \ldots x_{s'_m} : P(\bar{x})
    \]
  - Instantiation for $i$ is unique for every $S$ instantiation
    \[
    \forall \bar{x} \ \bar{x}' : P(\bar{x}) \land P(\bar{x}') \land \bar{x}\big|_S = \bar{x}'\big|_S \Rightarrow \bar{x}_i = \bar{x}'_i
    \]

- Prove entailed by $T$
**Detection**

- Theorem prover: checks whether $T_1 \models T_2$
  - Readily available for FO
    - http://www.cs.miami.edu/~tptp/CASC/
  - Function constraints are in FO
  - $T$ is in FO(\text{IDP})
    - $\Rightarrow$ Transform to FO theory $T_{\text{FO}}$

- Transform into TPTP format
- Call any TPTP theorem prover
FO(.)^IDP TO FO

- Transform T into *model-equivalent* $T_{FO}$
  - **Types:** sentences enforcing type
    \[
    \forall id \ x \ y : \text{pos}(id, x, y) \Rightarrow \text{Id}(id) \land X(x) \land Y(y)
    \]
  - **Partial functions:** graph + uniqueness constraint
  - **Aggregates:**
    - Small cardinality ~ to function constraints
    - Otherwise: define “accumulator” function
  - **Definitions:**
    - First-order level mapping
      (derived from [Janhunen et al.,2009])
FO(.)^{IDP} TO FO

- Level mapping and accumulator
  - Heavily recursive
  - Difficult to prove

- Entailment is transitive
  + only need positive answer

- Can weaken theory!
  - Use (Clark’s) completion for definitions
  - Replace aggregate atoms by true
2-D SQUARE PACKING

\( \forall id : \exists!(x, y) : \text{pos}(id, x, y) \)
\( \forall id : \exists!s : \text{size}(id, s) \)

\( \forall id_1, id_2 : id_1 \neq id_2 \Rightarrow \text{noOverlap}(id_1, id_2) \)

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

\[ \begin{aligned}
\forall id_1, id_2 : & \quad \text{leftof}(id_1, id_2) \leftarrow \\
& \quad \exists x_1, y_1, x_2, y_2, s_1 : \text{pos}(id_1, x_1, y_1) \land \text{size}(id_1, s_1) \\
& \quad \land \text{pos}(id_2, x_2, y_2) \land x_1 + s_1 \leq x_2
\end{aligned} \]
2-D Square Packing in FO

\( \forall id : \exists x \ y : pos(id, x, y) \)
\( \forall id \ x \ y \ x' \ y' : pos(id, x, y) \land pos(id, x', y') \Rightarrow x = x' \land y = y' \)

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

\( \forall id_1 \ id_2 : noOverlap(id_1, id_2) \iff \\
\text{leftof}(id_1, id_2) \lor \text{leftof}(id_2, id_1) \\
\lor \text{below}(id_1, id_2) \lor \text{below}(id_2, id_1) \)

\( \forall id_1 \ id_2 : \text{leftof}(id_1, id_2) \iff \\
\exists x_1 \ y_1 \ x_2 \ y_2 \ s_1 : pos(id_1, x_1, y_1) \land size(id_1, s_1) \\
\land pos(id_2, x_2, y_2) \land x_1 + s_1 \leq x_2 \)
WHICH DEPENDENCIES TO CHECK?

- Basic algorithm
  For all symbols $P$ in $T$, subsets of arguments $S$ of $P$, arguments $i$ of $P$
  check whether $d<P,S,i>$ holds in $T$
  if entailed, rewrite $T$
  continue

- Anytime

- Powerset of arity?
  - Aim: small arity symbols
  - If dependent, also dependent on supersets
    => From smallest to largest index sets
  - First largest dependency (very common)
# Offline Detection Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#\text{\texttt{var}}_{in}</th>
<th>#\text{\texttt{var}}_{out}</th>
<th>#f (#\text{\texttt{tot}}/#\text{\texttt{part}})</th>
<th>#calls</th>
<th>time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permutation-P.-Matching</td>
<td>21</td>
<td>8</td>
<td>3(3/0)</td>
<td>18</td>
<td>18.99</td>
</tr>
<tr>
<td>Valve-Location</td>
<td>27</td>
<td>24</td>
<td>1(1/0)</td>
<td>76</td>
<td>18.99</td>
</tr>
<tr>
<td>Connected-D.-M.-Still-Life</td>
<td>31</td>
<td>31</td>
<td>0(0/0)</td>
<td>24</td>
<td>43.98</td>
</tr>
<tr>
<td>Graceful-Graphs</td>
<td>32</td>
<td>21</td>
<td>2(1/1)</td>
<td>21</td>
<td>1.49</td>
</tr>
<tr>
<td>Bottle-Filling</td>
<td>26</td>
<td>14</td>
<td>4(4/0)</td>
<td>30</td>
<td>1.49</td>
</tr>
<tr>
<td>NoMystery</td>
<td>155</td>
<td>84</td>
<td>16(2/14)</td>
<td>181</td>
<td>363.05</td>
</tr>
<tr>
<td>Ricochet-Robots</td>
<td>109</td>
<td>77</td>
<td>13(0/13)</td>
<td>222</td>
<td>447.07</td>
</tr>
<tr>
<td>Reachability</td>
<td>5</td>
<td>5</td>
<td>0(0/0)</td>
<td>24</td>
<td>51.99</td>
</tr>
<tr>
<td>Visit-all</td>
<td>22</td>
<td>10</td>
<td>2(2/0)</td>
<td>19</td>
<td>36.99</td>
</tr>
<tr>
<td>KnightTourWithHoles</td>
<td>80</td>
<td>50</td>
<td>8(4/4)</td>
<td>63</td>
<td>118.02</td>
</tr>
<tr>
<td>Crossing-Minimization</td>
<td>38</td>
<td>28</td>
<td>1(0/1)</td>
<td>81</td>
<td>12.00</td>
</tr>
<tr>
<td>Solitaire</td>
<td>114</td>
<td>54</td>
<td>3(0/3)</td>
<td>88</td>
<td>197.05</td>
</tr>
<tr>
<td>Weighted-Sequence</td>
<td>49</td>
<td>29</td>
<td>6(6/0)</td>
<td>21</td>
<td>33.99</td>
</tr>
<tr>
<td>Stable-Marriage</td>
<td>20</td>
<td>4</td>
<td>3(0/3)</td>
<td>3</td>
<td>0.26</td>
</tr>
<tr>
<td>Incremental-Scheduling</td>
<td>89</td>
<td>60</td>
<td>4(4/0)</td>
<td>84</td>
<td>11.01</td>
</tr>
<tr>
<td>Maximal-Clique</td>
<td>7</td>
<td>7</td>
<td>0(0/0)</td>
<td>16</td>
<td>0.79</td>
</tr>
<tr>
<td>Graph-Colouring</td>
<td>9</td>
<td>4</td>
<td>1(0/0)</td>
<td>9</td>
<td>0.49</td>
</tr>
<tr>
<td>Database</td>
<td>15</td>
<td>2</td>
<td>5(5/0)</td>
<td>4</td>
<td>0.14</td>
</tr>
<tr>
<td>Packing</td>
<td>37</td>
<td>13</td>
<td>5(5/0)</td>
<td>35</td>
<td>14.01</td>
</tr>
</tbody>
</table>
INDUCTIVE RELATIONS

- Proving of inductive properties
  - Unsolved problem
  - E.g. weighted sequence problem:
    \[ \{ \text{weight}(\text{node} + 1, w) \leftarrow \text{weight}(n, \ldots) \ldots \} \]
    Neither SPASS nor Princess could detect the function

- Possible solution:
  - Prove function constraints for base case
  - Afterwards prove for inductive case given base case

- Promising manual experiment with Princess
## Improving Model Generation

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>pred. (#)</th>
<th>func. (#)</th>
<th>pred. (sec)</th>
<th>func. (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N01-Perm.-P.-Matching</td>
<td>2/5</td>
<td>5/5</td>
<td>10</td>
<td>624</td>
</tr>
<tr>
<td>N02-Valve-L.-Problem</td>
<td>1/5</td>
<td>5/5</td>
<td>800.81</td>
<td>54.47</td>
</tr>
<tr>
<td>N05-Graceful-Graphs</td>
<td>5/5</td>
<td>5/5</td>
<td>2.59</td>
<td>0.89</td>
</tr>
<tr>
<td>N20-Visit-All</td>
<td>5/5</td>
<td>5/5</td>
<td>3.18</td>
<td>2.01</td>
</tr>
<tr>
<td>OO-database</td>
<td>0/5</td>
<td>3/5</td>
<td>1000</td>
<td>415.47</td>
</tr>
<tr>
<td>Packing</td>
<td>0/5</td>
<td>5/5</td>
<td>1000</td>
<td>52.07</td>
</tr>
</tbody>
</table>
Course-scheduling problem

- Sessions have a
  - Timeslot
  - Teacher
  - Student group
  - Course

Naive modeler might express this as

\[
\text{planned}(\text{session}, \text{group}, \text{course}, \text{teacher}, \text{time})
\]

Teacher cannot teach multiple sessions

\[
\forall \text{sid} \; \text{sg} \; \text{c} \; \text{ts} \; \text{te} : \text{planned}(\text{sid}, \text{sg}, \text{c}, \text{ts}, \text{te}) \Rightarrow \\
\neg \exists \text{sid}_2 \; \text{sg}_2 \; \text{c}_2 : \text{sid}_2 \neq \text{sid} \land \text{planned}(\text{sid}_2, \text{sg}_2, \text{c}_2, \text{ts}, \text{te})
\]
ADDITIONAL APPLICATION
SYMBOL SPLITTING

- Course-scheduling problem
  - Sessions have a
    - Timeslot
    - Teacher
    - Student group
    - Course
  - As automatically converted into
    taughtBy(session):teacher, at(session):time, ...

- Teacher cannot teach multiple sessions
  \[ \forall sid \; sid_2 : at(sid) = at(sid_2) \rightarrow taughtBy(sid) \neq taughtBy(sid_2) \]
Corrupted encoding
queensC.lp

\{ queen(1..n,1..n,1..n) \}.

:- not \{ queen(I,J,K) \} == n.
:- queen(I,J,K), queen(I,JJ,K), J != JJ.
:- queen(I,J,K), queen(II,J,K), I != II.
:- queen(I,J,K), queen(II,JJ,K), (I,J)!=(II,JJ), I-J==II-JJ.
:- queen(I,J,K), queen(II,JJ,K), (I,J)!=(II,JJ), I+J==II+JJ.

queen(I,J) :- queen(I,J,K).
FUTURE WORK

- Investigate inductively defined functions
- Extend constraint support in solver
- More complex constraints
  - alldifferent, partition, ...
  1. Find potential candidates
  2. Represent as FO sentence + prove
  3. Find which part of the theory can be dropped

⇒ No clear solutions for these
⇒ 1. Approach for symmetry detection
    [Mears et Al., 2011]
⇒ 2. using meta-representation?
    [Asparteme Banbara et al.]
CONCLUSION

- Next-generation ASP/SMT solvers
  - More complex constraints

- Allowing function terms in the grounding
  - Reduces grounding size
  - Solver input contains (basic) finite-domain CSP constraints

- Approach to
  - detect functional dependencies
  - Rewrite the theory making it explicit

- Experimental results
  - Many functional dependencies
  - Improve model generation
  - If the prover can prove them
INTERESTING APPLICATION

- Given a list of n persons (1000+)
  - info on age, gender and city and country of origin

- Partition them into groups of size between 10 and 15 persons

- Never from the same city

- Approximately the same ratio male/female in every group

- Preferably from different countries