The FO(·) Knowledge Base System project

Marc Denecker
Knowledge Representation & Reasoning (KRR)
Katholieke Universiteit Leuven

February 26, 2016
Introduction: the FO(·) KBS project

FO(·)

Inference of the KBS: progress report
  Model expansion and visualisation
  Imperative + Declarative Programming (IDP)

Conclusions
Introduction: the FO(·) KBS project

FO(·)

Inference of the KBS: progress report
Model expansion and visualisation
Imperative + Declarative Programming (IDP)

Conclusions
The fundamental KRR research question

- Humans experts possess (declarative) knowledge.
- They use it
  - to accomplish tasks
  - to solve problems
  - or to build programs that do this for us
    (computer science)
- How does this work?
The fundamental KRR research question

- Humans experts possess (declarative) knowledge.
- They use it
  - to accomplish tasks
  - to solve problems
  - or to build programs that do this for us (computer science)
- How does this work?

- Inherently a KRR research question.
  - (KRR: Knowledge Representation and Reasoning)
- If we ever want to be able to build software systems in a principled way, we will NEED to understand this.
- This places KRR at the foundations of computer science.
State of the art

- About every area in Computational logic is involved in aspects of this question.
- Scientific understanding is partial and scattered over the many fields of computational logic and declarative problem solving.
State of the art

- About every area in Computational logic is involved in aspects of this question.
- Scientific understanding is partial and scattered over the many fields of computational logic and declarative problem solving.
- One issue that fragments computational logic more than anything else:

  the reasoning/inference task
State of the art

For every type of reasoning task, a new logic (or more than one):

- Classical first order logic (FO): deduction
- Deductive Databases (SQL, Datalog): query answering & other database operations
- Answer set Programming (ASP): answer set computation
- Abductive Logic Programming: abduction
- Constraint Programming (CP): constraint solving
- Description logics: subsumption ⊆ deduction
- Planning languages PDDL: planning
- Temporal logics: model checking
State of the art

A declarative proposition:

Each lecturer teaches at least one course in the first bachelor
State of the art

A declarative proposition:

Each lecturer teaches at least one course in the first bachelor

What is its purpose? What task is it to be used for?

- It could be a query to a database.
State of the art

A declarative proposition:

Each lecturer teaches at least one course in the first bachelor

What is its purpose? What task is it to be used for?

- It could be a query to a database.
- It could be a constraint in a course assignment problem.
State of the art

A declarative proposition:

Each lecturer teaches at least one course in the first bachelor

What is its purpose? What task is it to be used for?

▶ It could be a query to a database.
▶ It could be a constraint in a course assignment problem.
▶ It could be a desired property, to be proven from a formal specification of the course assignment domain.
▶ ...
State of the art

A declarative proposition:

Each lecturer teaches at least one course in the first bachelor

What is its purpose? What task is it to be used for?

- It could be a query to a database.
- It could be a constraint in a course assignment problem.
- It could be a desired property, to be proven from a formal specification of the course assignment domain.

... 

In the current state of the art, depending on the task to be solved, we need a different system and a different logic to represent this proposition.
Is declarative knowledge not independent of the task (and hence, of a specific form of inference)?
An illustration

Question: How to solve a graph coloring problem in a declarative way?

- Step 1: What is the knowledge? What is the central proposition of a correct graph coloring?
An illustration

Question: How to solve a graph coloring problem in a declarative way?

- Step 1: What is the knowledge? What is the central proposition of a correct graph coloring?
  - No two adjacent vertices have the same color.
  - In FO (classical first order logic):
    \[ \forall x \forall y (G(x, y) \Rightarrow Col(x) \neq Col(y)) \]
An illustration

Question: How to solve a graph coloring problem in a declarative way?

▶ Step 1: What is the knowledge? What is the central proposition of a correct graph coloring?
  ▶ No two adjacent vertices have the same color.
  ▶ In FO (classical first order logic):
    \[ \forall x \forall y (G(x, y) \Rightarrow \text{Col}(x) \neq \text{Col}(y)) \]

▶ Step 2: Input/Output:
  ▶ Input: a pair of a graph \( G(\text{Vertex}, \text{Vertex}) \) and a set \( \text{Color} \) of colours; in FO, this is called a structure
  ▶ Output: a function \( \text{Col}(\text{Vertex}) : \text{Color} \) satisfying the proposition
An illustration

Question: How to solve a graph coloring problem in a declarative way?

▶ Step 1: What is the knowledge? What is the central proposition of a correct graph coloring?
  ▶ No two adjacent vertices have the same color.
  ▶ In FO (classical first order logic):
    \[ \forall x \forall y (G(x, y) \Rightarrow Col(x) \neq Col(y)) \]

▶ Step 2: Input/Output:
  ▶ Input: a pair of a graph \( G(\text{Vertex}, \text{Vertex}) \) and a set \( \text{Color} \) of colours; in FO, this is called a structure
  ▶ Output: a function \( Col(\text{Vertex}) : \text{Color} \) satisfying the proposition

▶ Step 3: What kind of inference do we need to solve the problem?
Question: How to solve a graph coloring problem in a declarative way?

▶ Step 1: What is the knowledge? What is the central proposition of a correct graph coloring?
  ▶ No two adjacent vertices have the same color.
  ▶ In FO (classical first order logic):

\[
\forall x \forall y (G(x, y) \implies Col(x) \neq Col(y))
\]

▶ Step 2: Input/Output:
  ▶ Input: a pair of a graph \(G(\text{Vertex}, \text{Vertex})\) and a set \(\text{Color}\) of colours; in FO, this is called a structure
  ▶ Output: a function \(Col(\text{Vertex}): \text{Color}\) satisfying the proposition

▶ Step 3: What kind of inference do we need to solve the problem?
  ▶ model generation
“What kind of inference do we need to solve this problem?”

▶ A question that until recently, for FO, was not even asked let alone addressed.
“What kind of inference do we need to solve this problem?”

- A question that until recently, for FO, was not even asked let alone addressed.
- FO was seen exclusively as the logic of deductive reasoning.
  - In some fields, this is still the dominating view.
“What kind of inference do we need to solve this problem?”

- A question that until recently, for FO, was not even asked let alone addressed.
- FO was seen exclusively as the logic of deductive reasoning.
  - In some fields, this is still the dominating view.
- Deduction is utterly useless for solving the graph coloring problem.

Constraint Programming Languages
- Ilog, Zinc, Constraint Logic Programming, ...
- Answer Set Programming (ASP)
“What kind of inference do we need to solve this problem?”

- A question that until recently, for FO, was not even asked let alone addressed.
- FO was seen exclusively as the logic of **deductive reasoning**.
  - In some fields, this is still the dominating view.
- Deduction is utterly useless for solving the graph coloring problem.
- Instead, people developed new logics to handle problems like this:
  - Constraint Programming Languages
    - Ilog, Zinc, Constraint Logic Programming, . . .
  - Answer Set Programming (ASP)
Why do we need all these syntaxes for expressing the same information?

Isn’t it possible to solve multiple types of tasks using the same language?
Spread out over all disciplines of computational logic, there is an enormous expertise about KR and inference.

If only we could bundle what is known about KR and inference in a coherent scientific framework!?
The FO(·)-KBS project: an integration project

On the logical level: FO(·)

Knowledge exists, it can be studied through the methods of formal empirical science

Study “knowledge” by principled development of expressive KR languages

- Clear informal semantics
- Expressive languages, rich enough so that the information, relevant to solve a problem CAN be represented.
- Classical first-order logic (FO) as foundation, extended where necessary.
The FO(·)-KBS project: an integration project

On the logical level: FO(·)

Knowledge exists, it can be studied through the methods of formal empirical science

Study “knowledge” by principled development of expressive KR languages

▶ Clear informal semantics
▶ Expressive languages, rich enough so that the information, relevant to solve a problem CAN be represented.
▶ Classical first-order logic (FO) as foundation, extended where necessary.

(FO(·)= family of extensions of FO)
The FO(⋅)-KBS project: an integration project

- On the inference level:
  - Building solvers for various forms of inference for FO(⋅)
  - Integrating various solving techniques from various declarative programming paradigms in one Knowledge Base System.
What do I mean with “inference”?

An Inference is a computational task:

- **Input**: a tuple of objects including a logic theory
- **Output**: a set of computed objects such that the output is invariant under replacing the input theory by a logically equivalent theory.

- **E.g., Deduction inference**
  - Input theory $T$, sentence $\varphi$
  - Output: true if $T \models \varphi$

- **E.g., Query inference**:
  - Input structure $\mathcal{A}$, set expression $\{\bar{x} : \varphi\}$
  - Output $\{\bar{x} : \varphi\}^\mathcal{A}$
The FO(·)-KBS project: an integration project

- On the application level:
  - Towards a typology of tasks and computational problems in terms of (the same) logic and inference.
  - Eagerly searching for novel ways of using declarative specifications to solve problems.
The FO(·)-KBS project is the long-term research goal of the KU Leuven KRR research group.
Introduction: the FO(·) KBS project

FO(·)

Inference of the KBS: progress report
Model expansion and visualisation
Imperative + Declarative Programming (IDP)

Conclusions
Why FO as a foundation?

FO: the language that **failed** in the seventies?

- Too expressive for building “practical” systems?
  - Undecidability
  - Expressivity/Efficiency trade-off

- FO is not suitable for describing **common sense knowledge**?
  - Nonmonotonic reasoning

- FO as a language is **too difficult** for practical use?
  - E.g., quantifiers
Why FO as a foundation?

- FO, the outcome of 18’s and 19’s century’s research in “laws of thought”
  - E.g., Leibniz, De Morgan, Boole, Frege, Peirce

- Model semantics as a way to formalize meaning.

\[ \forall x (\text{Human}(x) \Rightarrow \text{Man}(x) \lor \text{Woman}(x)) \]

means All humans are men or women
Why FO as a foundation?

- FO, the outcome of 18’s and 19’s century’s research in “laws of thought”
  - E.g., Leibniz, De Morgan, Boole, Frege, Peirce

- FO is about a small set of connectives:
  - $\land, \lor, \neg, \forall, \exists, \leftrightarrow, \Rightarrow$
  - Essential for KR, the right semantics in FO
Why FO as a foundation?

- FO, the outcome of 18’s and 19’s century’s research in “laws of thought”
  - E.g., Leibniz, De Morgan, Boole, Frege, Peirce

- FO is about a small set of connectives:
  \[ \land, \lor, \neg, \forall, \exists, \leftrightarrow, \Rightarrow \]
  - Essential for KR, the right semantics in FO

- Crystal clear informal semantics
  \[ \forall x (\text{Human}(x) \Rightarrow \text{Man}(x) \lor \text{Woman}(x)) \]
  means
  All humans are men or women
Why FO as a foundation?

- FO, the outcome of 18’s and 19’s century’s research in “laws of thought”
  - E.g., Leibniz, De Morgan, Boole, Frege, Peirce

- FO is about a small set of connectives:
  \[ \land, \lor, \neg, \forall, \exists, \leftrightarrow, \Rightarrow \]
  - Essential for KR, the right semantics in FO

- Crystal clear informal semantics
  \[ \forall x (\text{Human}(x) \Rightarrow \text{Man}(x) \lor \text{Woman}(x)) \]
  means
  All humans are men or women

- Model semantics as a way to formalize meaning.
Claims

(1) Every expressive declarative modelling language has a substantial overlap with FO, in one form or the other.
Claims

(1) Every expressive declarative modelling language has a substantial overlap with FO, in one form or the other.

- For some languages, the syntax, conceptuology, terminology may severely obscure the relationship.
  - SQL
  - ALLOY
  - Zinc (a constraint programming language)
  - Answer Set Programming
Claims

(1) Every expressive declarative modelling language has a substantial overlap with FO, in one form or the other.

- For some languages, the syntax, conceptuology, terminology may severely obscure the relationship.
  - SQL
  - ALLOY
  - Zinc (a constraint programming language)
  - Answer Set Programming

(2) But FO is not enough for practical KR.
But FO is undecidable?
But FO is undecidable?

To be precise:

- **Deductive inference** in FO is Undecidable
- Other forms of inference are tractable (P) or almost (NP)
- Other forms of inference have far more applications.
- Many/most practical software problems are not deductive ones.
But FO is undecidable?

To be precise:

- **Deductive inference** in FO is Undecidable
- Other forms of inference are tractable (P) or almost (NP)
- Other forms of inference have far more applications.
- Many/most practical software problems are not deductive ones.

- Much richer “logics” are in use in industry (SQL, ILOG, Zinc)
But FO is undecidable?

To be precise:

- **Deductive inference** in FO is Undecidable
- Other forms of inference are tractable (P) or almost (NP)
- Other forms of inference have far more applications.
- Many/most practical software problems are not deductive ones.

Much richer “logics” are in use in industry (SQL, ILOG, Zinc)

In the FO(·)-KBS project, we focus on expressivity with the aim to develop expressive natural KR languages suitable to express domain knowledge in REAL problems.
But FO is undecidable?

To be precise:
- Deductive inference in FO is Undecidable
- Other forms of inference are tractable (P) or almost (NP)
- Other forms of inference have far more applications.
- Many/most practical software problems are not deductive ones.

Much richer “logics” are in use in industry (SQL, ILOG, Zinc)

In the FO(·)-KBS project, we focus on expressivity with the aim to develop expressive natural KR languages suitable to express domain knowledge in REAL problems.

We ignore the expressivity/tractability trade-off!
FO(·): turning FO into a practical KR language

- FO does not suffice for knowledge representation, modelling, specification
FO(·): turning FO into a practical KR language

- FO does not suffice for knowledge representation, modelling, specification
  ⇒ FO(·)
FO(·): turning FO into a practical KR language

- FO does not suffice for knowledge representation, modelling, specification

⇒ FO(Types)

- Types
**FO(·): turning FO into a practical KR language**

- FO does not suffice for knowledge representation, modelling, specification

  $\Rightarrow$ FO(Types,ID )
  - Types
  - (Inductive) Definitions
FO(·): turning FO into a practical KR language

- FO does not suffice for knowledge representation, modelling, specification

⇒ FO(Types,ID,Agg)

- Types
- (Inductive) Definitions
- Aggregates
FO(·): turning FO into a practical KR language

- FO does not suffice for knowledge representation, modelling, specification

\[ \Rightarrow \quad FO(Types,ID,Agg,Arit) \]

- Types
- (Inductive) Definitions
- Aggregates
- Arithmetic
FO(·): turning FO into a practical KR language

- FO does not suffice for knowledge representation, modelling, specification

\[ \Rightarrow \quad \text{FO(Types,ID,Agg,Arit,FD,Mod,HO,\ldots)} \]

- Types
- (Inductive) Definitions
- Aggregates
- Arithmetic
- Coinductive Definitions
FO(·): turning FO into a practical KR language

- FO does not suffice for knowledge representation, modelling, specification

\[ \Rightarrow \quad \text{FO(Types,ID,Agg,Arit,FD,Mod)} \]

- Types
- (Inductive) Definitions
- Aggregates
- Arithmetic
- Coinductive Definitions
- Modal operators
FO(·): turning FO into a practical KR language

- FO does not suffice for knowledge representation, modelling, specification

⇒ FO(Types,ID,Agg,Arit,FD,Mod,HO,...)
  - Types
  - (Inductive) Definitions
  - Aggregates
  - Arithmetic
  - Coinductive Definitions
  - Modal operators
  - Higher Order logic
  - ...

The FO(·) language framework
About adding Inductive Definitions to FO.
Some prototypical Inductive Definitions.

The two most common forms of ID’s.

The transitive closure $T_G$ of a graph $G$ is defined inductively:
- $(x, y) \in T_G$ if $(x, y) \in G$;
- $(x, y) \in T_G$ if for some vertex $z$, $(x, z), (z, y) \in T_G$.

We define $\mathcal{A} \models \varphi$ by induction on structure of $\varphi$:
- $\mathcal{A} \models q$ if $q \in \mathcal{A}$;
- $\mathcal{A} \models \alpha \land \beta$ if $\mathcal{A} \models \alpha$ and $\mathcal{A} \models \beta$;
- $\mathcal{A} \models \neg \alpha$ if $\mathcal{A} \not\models \alpha$
  (i.e., if not $\mathcal{A} \models \alpha$);

Monotone inductive  Induction over well-founded order
Properties of informal ID’s

- Linguistically, a set of informal rules (with negation)

- Semantically, two principles:
  - Non-constructively, the least set closed under rule application
  - Constructively, the set obtained by iterated rule application.

- These two principles coincide – Tarski!
Properties of informal ID’s

- Linguistically, a set of informal rules (with negation)

- Semantically, two principles:
  - Non-constructively, the least set closed under rule application
  - Constructively, the set obtained by iterated rule application.

- These two principles coincide – Tarski!

- Only for monotone definitions!
Informal (inductive) definitions (ID’s)

- Definitions in mathematics:
  - a special sort of knowledge:
    - of mathematical precision
    - broadly used
    - intuitively well understood
    - but not scientifically well-understood
  - An ideal topic for formal empirical exact scientific study.
Adding ID’s to FO

- Inductive definitions frequently occur in KR and formal specifications
- ID’s cannot be expressed in FO in general.
  - Compactness theorem
  - It is necessary to extend FO with them.
An FO(ID) theory:

- FO sentences
- Definitions: sets of rules

\[
\begin{align*}
\forall x \forall y (R(x, y) \leftarrow G(x, y)) \\
\forall x \forall y (R(x, y) \leftarrow \exists z (G(x, z) \land R(z, y))) \\
\end{align*}
\]

\(\forall x \forall y R(x, y)\) expresses that \(R\) is the reachability graph of graph \(G\) and \(G\) is a connected graph

Claim (KR 2014)

Rules under well-founded semantics provide a uniform formalism for expressing the most common forms of definitions.
FO(ID) (Denecker 2000, Denecker & Ternovska 2008)

An FO(ID) theory:
- FO sentences
- Definitions: sets of rules

Example:
\[
\{ \forall x \forall y (R(x, y) \leftarrow G(x, y)) \\
\forall x \forall y (R(x, y) \leftarrow \exists z (G(x, z) \land R(z, y))) \}
\]
\[\forall x \forall y R(x, y)\]

expresses that . . .
An FO(ID) theory:
- FO sentences
- Definitions: sets of rules

Example:

\[
\begin{aligned}
\forall x \forall y (R(x, y) & \iff G(x, y)) \\
\forall x \forall y (R(x, y) & \iff \exists z (G(x, z) \land R(z, y)))
\end{aligned}
\]

\[\forall x \forall y R(x, y)\]

expresses that ... \( R \) is the reachability graph of graph \( G \) and \( G \) is a connected graph
FO(ID) (Denecker 2000, Denecker & Ternovska 2008)

An FO(ID) theory:
- FO sentences
- Definitions: sets of rules

Example:
\[
\begin{align*}
\forall x \forall y (R(x, y) & \leftarrow G(x, y)) \\
\forall x \forall y (R(x, y) & \leftarrow \exists z (G(x, z) \land R(z, y)))
\end{align*}
\]
\[
\forall x \forall y R(x, y)
\]
expresses that ... \( R \) is the reachability graph of graph \( G \) and \( G \) is a connected graph

Claim (KR 2014)

Rules under well-founded semantics provide a uniform formalism for expressing the most common forms of definitions.
Another useful extension of FO are aggregates.
An example inductive definition with aggregate

A company A controls company B if the total sum of the shares in company B owned by A or by companies controlled by A is more than 50%.

An inductive definition over aggregates.

The vocabulary

- $\text{Cont}(x, y)$: company $x$ controls company $y$.
- $\text{OwnsSh}(x, y, s)$: company $x$ owns $s$ shares in company $y$. 

An example inductive definition with aggregate

A company \( A \) controls company \( B \) if the total sum of the shares in company \( B \) owned by \( A \) or by companies controlled by \( A \) is more than 50%.

An inductive definition over aggregates.

The vocabulary

- \( \text{Cont}(x, y) \): company \( x \) controls company \( y \).
- \( \text{OwnsSh}(x, y, s) \): company \( x \) owns \( s \) shares in company \( y \).

\[
\forall a \forall b (\text{Cont}(a, b) \iff \text{Sum} \{ (s, c) : (c = a \lor \text{Cont}(a, c)) \land \text{OwnsSh}(c, b, s) : s \} > 50)
\]
FO(ID) as a rule formalism

- Many rule-based formalisms
  - Logic Programming
  - Datalog
  - Answer Set Programming
  - Description logics with rules
  - Abductive Logic Programming
  - Business rule systems

- Unclear semantical status of rules.
- FO(ID) overlaps with many of them and provides two precise, well-understood declarative sorts of rules
  - Material implications, definitional rules
Future work

Adding

- Causal statements (ICLP 2014, ECAI 2014, NMR 2014)
- Modal operators
- Higher Order
Introduction: the FO(·) KBS project

FO(·)

Inference of the KBS: progress report
Model expansion and visualisation
Imperative + Declarative Programming (IDP)

Conclusions
How to use Logic for problem solving

- A logic theory is a bag of (descriptive) information
  - A logic theory cannot be executed
  - A logic theory is not a program
  - A logic theory is not a representation of a problem
- So how can we use a logic theory to solve problems?
A Knowledge Base System (KBS)

- Manages a declarative Knowledge Base (KB): a theory
- Equipped with different forms of inference:
  - Inference 1
  - Inference 2
  - Inference 3
  - Inference 4
A Knowledge Base System (KBS)

- Manages a declarative Knowledge Base (KB): a theory
- Equiped with different forms of inference:
  - Model generation: Computing a schedule
  - Model checking: Verifying consistency of a schedule
  - Update and Revision: Updating a given schedule
  - Deduction for verification of the KB
  - Querying of defined predicates, ...
A Knowledge Base System (KBS)

- Manages a declarative Knowledge Base (KB): a theory
- Equipped with different forms of inference:
  - Model generation: Computing a schedule

Inference 2

Knowledge Base
University course scheduling

Model Generation
Computing a schedule

Inference 3

Inference 4

Checking consistency of schedule
Computing a schedule
Updating a schedule
A Knowledge Base System (KBS)

- Manages a declarative Knowledge Base (KB): a theory
- Equiped with different forms of inference:
  - Model generation: Computing a schedule
  - Model checking: Verifying consistency of a schedule
A Knowledge Base System (KBS)

- Manages a declarative Knowledge Base (KB): a theory
- Equipped with different forms of inference:
  - Model generation: Computing a schedule
  - Model checking: Verifying consistency of a schedule
  - Update and Revision: Updating a given schedule

Knowledge Base
University course scheduling

Model checking
Checking consistency of schedule

Revision Inference
Updating a schedule

Inference 4

Model Generation
Computing a schedule
A Knowledge Base System (KBS)

- Manages a declarative **Knowledge Base (KB)**: a theory
- Equipped with different forms of inference:
  - **Model generation**: Computing a schedule
  - **Model checking**: Verifying consistency of a schedule
  - **Update and Revision**: Updating a given schedule
  - **Deduction** for verification of the KB
  - **Querying** of defined predicates, . . .
A KBS demo

The course selection demo: interactive configuration
http://krr.bitbucket.org/courses/

5 Used forms of inference:

- Model Checking (P)
- Propagation (P)
- Model Generation (NP)
- Model Generation+Optimization (NP^{NP})
- Explanation (P)
A KBS demo

Demonstrating a principle that procedural programming languages can’t do:

Reusing the same specification/theory/knowledge base to solve different types of problems.
Implementation of KBS

IDP3:

- A KBS system
- Programming environment
  - Programming with theories, structures, inference methods
  - In an extension of procedural language Lua:

Built by KRR-members Broes De Cat, Bart Bogaerts, Joachim Jansen, Pieter Van Hertum, Jo Devriendt, Ingmar Dasseville (and ex-members Johan Wittocx, Maarten Mariën, Stef De Pooter)
Implementation of KBS

- Forms of inference currently under development:
  - *(Finite) Model expansion* (the core component of IDP3)
  - Optimisation
  - Propagation
  - Querying structures
  - $\Delta$-model generation and revision:
    - $\sim$view materialisation and update in databases
    - Computing & updating defined predicates
  - Progression of temporal FO(.) theories.
  - Model revision
  - Debugging, Explanation.
Model generation/expansion

Model Expansion

- **Input:**
  - An FO(.) theory $T$
  - An (finite) structure $\mathcal{A}_i$ for a subvocabulary of $T$, expressing domain and data.

- **Output:** a model $\mathcal{A}$ of $T$ expanding $\mathcal{A}_i$

Special case: Herbrand Model Generation
The grounder

Grounding = Eliminating quantification

- FO(.) theory
- CNF – ECNF – SMT (- FlatZinc )
- Term Rewrite
  - Type derivation
  - Symmetry breaking
  - Evaluate known definitions
  - Lifted unit propagation
  - Grounding with bounds
MinisatID: SMT solver

Solving = computing a model

MinisatID

PC solver 0..*

SAT-solver

ID-module

Agg-module Minisat++

CP-module Gecode

Model

CNF – ECNF – OPB – ASP - QBF
Technologies from different computational logic areas integrated:

- Constraint Programming technology
- Sat Modulo Theory (SMT)
- Logic Programming
- Answer Set Programming
- MIP: Mixed Integer Programming
<table>
<thead>
<tr>
<th>Solver</th>
<th>AST (sec.)</th>
<th>PSI (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>minisatid</td>
<td>950.91</td>
<td>51.62</td>
</tr>
<tr>
<td>g12cpx</td>
<td>1126.98</td>
<td>41.68</td>
</tr>
<tr>
<td>fzn2smt</td>
<td>1143.47</td>
<td>38.13</td>
</tr>
<tr>
<td>ortools</td>
<td>1316.25</td>
<td>30.65</td>
</tr>
<tr>
<td>g12lazyfd</td>
<td>1306.10</td>
<td>30.31</td>
</tr>
<tr>
<td>gecode</td>
<td>1354.65</td>
<td>29.51</td>
</tr>
<tr>
<td>izplus</td>
<td>1350.42</td>
<td>28.05</td>
</tr>
<tr>
<td>bprolog</td>
<td>1423.45</td>
<td>24.73</td>
</tr>
<tr>
<td>jacop</td>
<td>1435.123</td>
<td>24.67</td>
</tr>
<tr>
<td>g12fd</td>
<td>1424.80</td>
<td>23.57</td>
</tr>
<tr>
<td>mistral</td>
<td>1525.83</td>
<td>16.91</td>
</tr>
<tr>
<td>g12mip</td>
<td>1597.54</td>
<td>12.58</td>
</tr>
</tbody>
</table>

Table: Experimental evaluation of MiniZinc solvers on the CSPs in Benchmark Set B AmadiniGM13.
<table>
<thead>
<tr>
<th>Benchmark</th>
<th># solved IDP</th>
<th># solved Gringo-Clasp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perm. P. Matching</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Valves Location *</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>Still-Life *</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Graceful Graphs</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>Bottle Filling</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>NoMystery</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>Sokoban</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Ricochet Robots</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>Crossing Minim. *</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>Solitaire</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>Weighted Sequence</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Stable Marriage</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Incremental Sched.</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>Visit All * Core</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Knight’s Tour * Core</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Maximal Clique * Core</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Graph Colouring Core</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

**Table**: Experimental results for benchmarks of the 2013 ASP competition.
Other demos

- Model expansion with logic-based visualisation
- Programming with logical inference
- Temporal Reasoning: planning
- Temporal reasoning: execution and optimisation

Demos and examples can be accessed from https://dtai.cs.kuleuven.be/software/idp/
This application serves to solve sudoku puzzles and to visualise the outcome. The theory $T$ expresses the 3 laws of sudokus: one occurrence of each number in each row, column and block. Model expansion inference solves a puzzle specified in the input structure by returning a model $\mathcal{A}$. The solution is $\text{sudoku}^{\mathcal{A}}$. The user theory $T_{\_D3}$ (see next slide) is mainly a definition of IDPd3 graphical predicate and function symbols defined in terms of symbols interpreted in $\mathcal{A}$. $\Delta$-model expansion on $T_{\_D3}$ and input structure $\mathcal{A}$ computes a value for these predicates which is then transferred to the graphical program d3.

This is a simple illustration of logic to transform one sort of datastructure in another.
theory T_D3 : V_out {
  
  d3_type(1, Cell(r,k)) = rect <-
  d3_rect_width(1, Cell(r,k)) = 4 <-.
  d3_rect_height(1, Cell(r,k)) = 4 <-.
  d3_color(1, Cell(r,k)) = "white" <-.
  d3_x(1, Cell(r,k)) = 5*k <-.
  d3_y(1, Cell(r,k)) = 5*r <-.

  d3_type(1, Text(r, k)) = text <-.
  d3_x(1, Text(r, k)) = 5*k <-.
  d3_y(1, Text(r, k)) = 5*r + 1 <-.
  d3_text_size(1, Text(r, k)) = 3 <-.
  d3_text_label(1, Text(r, k)) = t <-
    sudoku(r, k) = c & toString(c) = t.
  d3_color(1, Text(r, k)) = "black" <-.

  d3_order(1, Cell(r, k)) = 0 <-.
  d3_order(1, Text(r,k)) = 1 <-.
  
}

This definition defines slide 1 (argument 1) with:

▶ for each cell \((r, c)\) a rectangular object denoted \(\text{Cell}(r,c)\), and a text object \(\text{Text}(r,k)\).

▶ It defines type, width, height, color, x and y position of the rectangular object.

▶ It defines type, text size and label, color, x and y position of the text object.

▶ The \textbf{sudoku} number at cell \((r, c)\) is the label of the text object..

▶ that text objects are in front of rectangular objects

\(\Delta\)-model expansion expands a structure interpreting \textbf{sudoku} into a structure interpreting all these graphical symbols. This is fed into \(d3\).
IDPd3: an example

Input structure

\[
sudoku = \{1, 1 \rightarrow 1; \ldots; 9, 9 \rightarrow 4\}
\]

\[
\downarrow \Delta\text{-model generation}
\]

\[
d_3\text{\_type} = \{1, Cell(1, 1) \rightarrow \text{rect}; 1, Text(1, 1) \rightarrow \text{text}; \ldots\}
\]

\[
d_3\text{\_color} = \{1, Cell(1, 1) \rightarrow \text{white}; 1, Text(1, 1) \rightarrow \text{black}; \ldots\}
\]

\[
d_3\text{\_x} = \{1, Cell(1, 1) \rightarrow 5; 1, Cell(1, 2) \rightarrow 10; \ldots\}
\]

\[
\downarrow \text{translation to d3 input + d3}
\]
A prototype of a knowledge-based programming environment

- IDP3: A programming environment
- High level objects: vocabularies, theories, structures
- Functionalities for manipulation and inference
- Implemented in the language Lua
- A new way of mixing Declarative and Procedural knowledge
A demo: generating Sudoku-puzzles

Sudoku-puzzle requirements’

➤ (consistency) It should allow one unique solution
➤ (minimality) If we delete any value of the puzzle, it has at least two solutions.
Background knowledge base in IDP

Vocabulary

vocabulary sudokuVoc {
  extern vocabulary grid::simpleGridVoc
  type Num isa nat
  type Block isa nat
  Sudoku(Row,Col) : Num
  InBlock(Block,R,Col)
}

Theory

theory sudokuTheory : sudokuVoc {
  ! r n : ?1 c : Sudoku(r,c) = n.
  ! c n : ?1 r : Sudoku(r,c) = n.
  ! b n : ?1 r c : InBlock(b,r,c) & Sudoku(r,c) = n.
  ! b r c : InBlock(b,r,c)
    ==> b = ((r-1)/3)*3 + ((c-1)/3) + 1.
}
A demo: generating Sudoku-puzzles

Puzzle := empty
Generate at most 2 solutions for Puzzle
While 2 solutions were found do{
    Select a random position where the two solutions differ
    Extend Puzzle with the value of the first solution at this position
    Generate at most 2 solutions for Puzzle
}
For each position of Puzzle that contains a value do {
    Delete the value at this position
    Generate at most 2 solutions for Puzzle
    If there are two solutions, undo the deletion of the value.
}
Visualize the puzzle and its unique solution
Procedures

procedure createSudoku() {
    math.randomseed(os.time())
    local puzzle = grid::makeEmptyGrid(9)

    stdoptions.nrmodels = 2
    local currsols = modelExpand(sudokuTheory, puzzle)

    while #currsols > 1 do
        repeat
            col = math.random(1,9)
            row = math.random(1,9)
            num = currsols[1][sudokuVoc::Sudoku](row, col)
        until num ~= currsols[2][sudokuVoc::Sudoku](row, col)

        makeTrue(puzzle[sudokuVoc::Sudoku].graph, {row, col, num})
        currsols = modelExpand(sudokuTheory, puzzle)
    end

    printSudoku(puzzle)
}
Discussion

Two sorts of inferences:

▶ generating solutions to puzzles: model expansion
▶ Visualizing through $\Delta$-model expansion
  ▶ computing a model of a definition $\Delta$
  ▶ A special case of model expansion
  ▶ No search
  ▶ Can be implemented very differently
  ▶ = View materialisation in deductive databases.

Access and manipulation of structures.

▶ Structures are objects in the environment
▶ Puzzle and its solutions are structures
▶ Checking and updating values at positions of puzzle
Reasoning on Temporal theories

- Temporal theory $T$: Linear Time Calculus
- Use Model expansion for planning with optimisation
- Use Progression for interactive execution.
  - Input: $T$, structure $\mathcal{A}$ representing state at time $i$
  - Output: structure $\mathcal{A}'$ representing possible state at time $i + 1$.
- Illustration: pacman.
Reasoning on Temporal theories

- Temporal theory $T$: Linear Time Calculus
- Use Model expansion for planning with optimisation
- Use Progression for interactive execution.
  - Input: $T$, structure $\mathcal{A}$ representing state at time $i$
  - Output: structure $\mathcal{A}'$ representing possible state at time $i + 1$.
- Illustration: pacman.

IDP is the only system that we know of that can use the same formal specification to solve both tasks.
A company running standard software systems on this principle:

- LogicBlox - Datalog:
  http://www.logicblox.com/
UI event handling

```
^sales[prod,date,store]=value
  <
  +button_clicked(form,submit),
  sales_entry_form_user(form,user),
  dropdown_selected(form)=prod,
  date_fld_value(form,date_fld)=date,
  num_fld_value(form,val_fld)=value,
  manager(store,user).
```
some clients

- AT&T
- Argos
- Toys 'R' Us
- PLANIXS
- dELiA*s
- Boeing
- Walgreens
- Polo
- Harrods
- Best Buy
- McKesson
- Sainsbury’s
- T.J. Maxx
- The Limited
- The Carphone Warehouse
- RenaissanceRe
Future

Knowledge-based software engineering

- Important gains to be made:
  - development time
  - compactness
  - correctness
  - reuse
  - maintainability

- Great scientific and practical challenges

We are in the process of searching niches with industry where our technology could already make a difference
A KRR-team with Ingmar Dasseville, Jo Devriendt and Matthias van der Hallen won the International Logic Programming and Constraint Programming competition with IDP.
Introduction: the FO(·) KBS project

FO(·)

Inference of the KBS: progress report
  Model expansion and visualisation
  Imperative + Declarative Programming (IDP)

Conclusions
Conclusion

- Computational logic is getting too complex
  - Too many logics
  - FO(\cdot): reuse and integration
  - We need a M3C :-)
Conclusion

- Computational logic is getting too complex
  - Too many logics
  - FO(·): reuse and integration
  - We need a M3C :-)

- Economies to be made:
  - Reusing specifications/modellings
  - Reusing Languages
  - Reusing inference technologies
  - Integration raises challenging new research questions
  - Integration produces multiplication effects
Conclusion

- Computational logic is getting too complex
  - Too many logics
  - FO(·): reuse and integration
  - We need a M3C :-) 

- Economies to be made:
  - Reusing specifications/modellings
  - Reusing Languages
  - Reusing inference technologies
  - Integration raises challenging new research questions
  - Integration produces multiplication effects

- New insights in a fundamental KRR research question
  - How do we use declarative K for problem solving?
  - What is the role of logic in computer science?
  - How to build better SE-systems using logic inference
Main Publications

▶ The logic FO(ID)

Denecker, Marc; Ternovska, Eugenia. A logic of nonmonotone inductive definitions, ACM Transactions on Computational Logic, volume 9, issue 2, 2008.

▶ The KBS-paradigm:

Denecker, Marc; Vennekens, Joost. Building a knowledge base system for an integration of logic programming and classical logic, ICLP 2008

▶ Theory of course selection demo

Vlaeminck, Hanne; Vennekens, Joost; Denecker, Marc. A logical framework for configuration software, PPDP 2009
The IDP system:
Wittocx, Johan; Mariën, Maarten; Denecker, Marc. The IDP system: A model expansion system for an extension of classical logic, Denecker, Marc (ed.), Logic and Search, LaSh 2008
Wittocx, Johan; Mariën, Maarten; Denecker, Marc. Grounding FO and FO(ID) with bounds, JAIR, volume 38, 2010
Wittocx, Johan; Mariën, Maarten; Denecker, Marc. GidL: A grounder for FO+, Thielscher, Michael; Pagnucco, Maurice (eds.), NMR 2008
Mariën, Maarten; Wittocx, Johan; Denecker, Marc; Bruynooghe, Maurice. SAT(ID): Satisfiability of propositional logic extended with inductive definitions, SAT 2008 - Theory and Applications of Satisfiability Testing, 2008
Mariën, Maarten; Wittocx, Johan; Denecker, Marc. Integrating inductive definitions in SAT, Logic for Programming, Artificial Intelligence, and Reasoning, Yerevan, Armenia, 2007.
The IDP programming environment:
De Pooter, Stef; Wittocx, Johan; Denecker, Marc. A prototype of a knowledge-based programming environment, INAP 2011
IDP, IDP3 and the demos’s are available via our webpage
http://krr.bitbucket.org
https://dtai.cs.kuleuven.be/software/idp/
Publications are on line via my webpage