Lecture II - part 2
the knowledge base system IDP

June 7, 2016

Inference of the KBS: progress report

- Model expansion and visualisation
- Imperative + Declarative Programming (IDP)

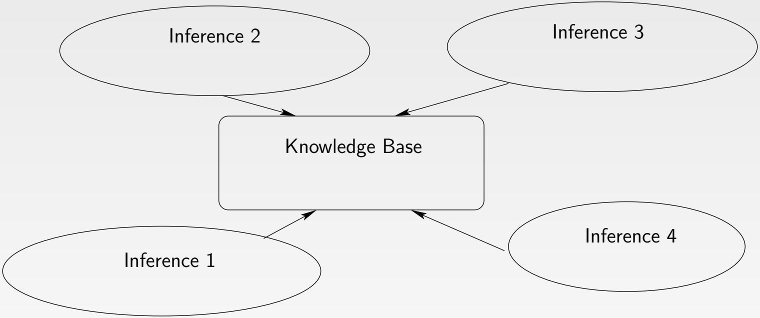
Inference of the KBS: progress report

- Model expansion and visualisation
- Imperative + Declarative Programming (IDP)

How to use Logic for problem solving

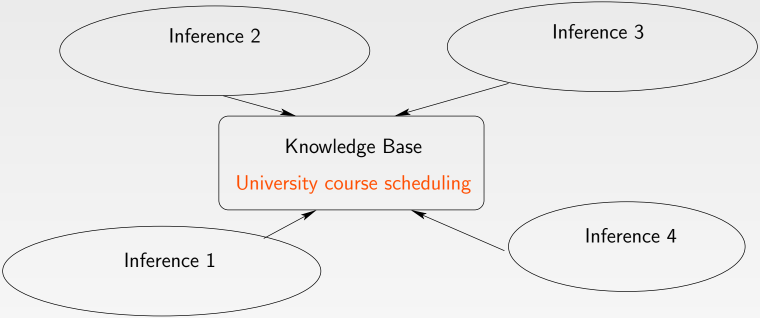
- A logic theory is a bag of (descriptive) information
  - A logic theory cannot be executed
  - A logic theory is not a program
  - A logic theory is not a representation of a problem
- So how can we use a logic theory to solve problems?

A Knowledge Base System (KBS)



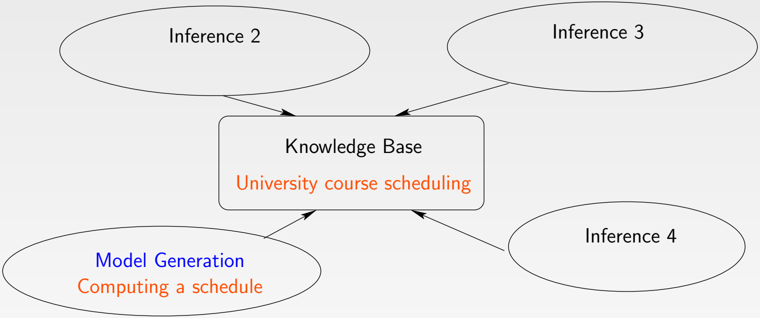
- Manages a declarative Knowledge Base (KB): a theory
- Equiped with different forms of inference:

A Knowledge Base System (KBS)



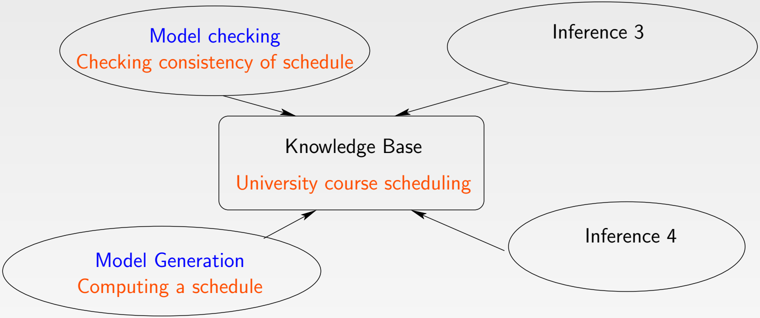
- Manages a declarative Knowledge Base (KB): a theory
- Equiped with different forms of inference:

A Knowledge Base System (KBS)



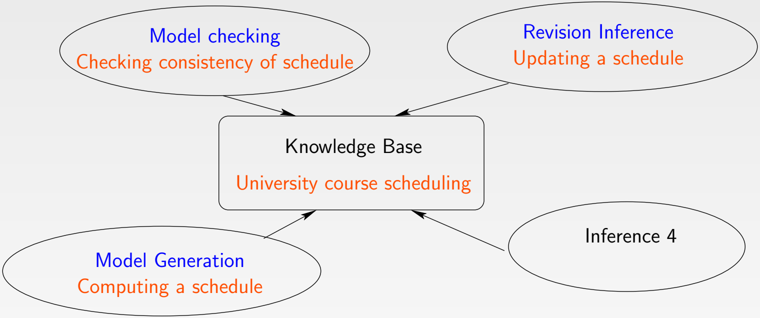
- Manages a declarative Knowledge Base (KB): a theory
- Equiped with different forms of inference:
  - Model generation: Computing a schedule

A Knowledge Base System (KBS)



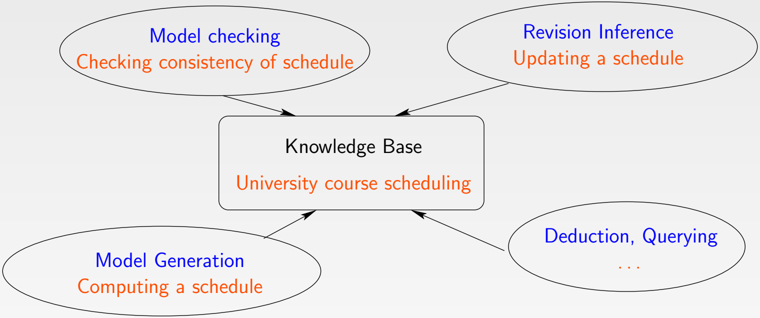
- Manages a declarative Knowledge Base (KB): a theory
- Equiped with different forms of inference:
  - Model generation: Computing a schedule
  - Model checking: Verifying consistency of a schedule

A Knowledge Base System (KBS)



- Manages a declarative Knowledge Base (KB): a theory
- 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

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 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
  - Δ-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

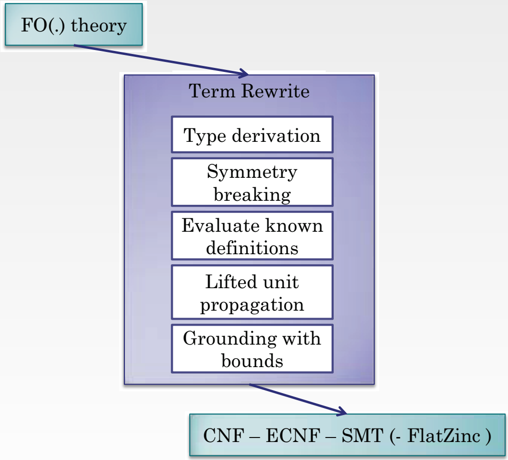
Model Expansion

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

Special case: Herbrand Model Generation

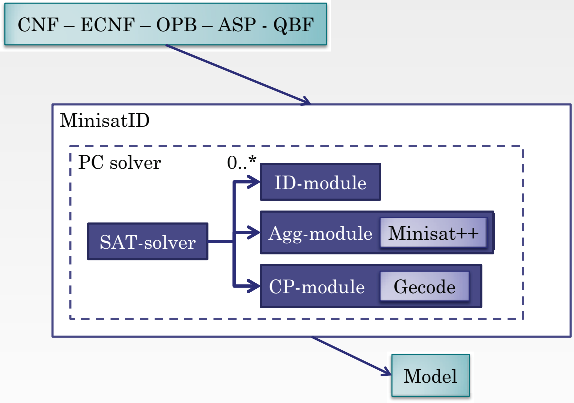
The grounder

Grounding = Eliminating quantification



MinisatID: SMT solver

Solving = computing a model



Technology

Technologies from different computational logic areas integrated:

- Constraint Programming technology
- Sat Modulo Theory (SMT)
- Logic Programming
- Answer Set Programming
- MIP: Mixed Integer Programming

Solver	AST (sec.)	PSI (%)
minisatid	950.91	51.62
g12cpx	1126.98	41.68
fzn2smt	1143.47	38.13
ortools	1316.25	30.65
g12lazyfd	1306.10	30.31
gecode	1354.65	29.51
izplus	1350.42	28.05
bprolog	1423.45	24.73
jacop	1435.123	24.67
g12fd	1424.80	23.57
mistral	1525.83	16.91
g12mip	1597.54	12.58

Table : Experimental evaluation of MiniZinc solvers on the CSPs in Benchmark Set B AmadiniGM13.

Benchmark	# solved IDP	# solved Gringo-Clasp
`Perm. P. Matching`	10	10
`Valves Location *`	**7**	4
`Still-Life *`	2	**3**
`Graceful Graphs`	3	**9**
`Bottle Filling`	10	10
`NoMystery`	**9**	6
`Sokoban`	**7**	5
`Ricochet Robots`	7	**10**
`Crossing Minim. *`	0	**9**
`Solitaire`	8	**9**
`Weighted Sequence`	10	10
`Stable Marriage`	10	10
`Incremental Sched.`	**6**	5
`Visit All` $_{core}$	6	**7**
`Knight's Tour` $_{core}$	**1**	0
`Maximal Clique` $*_{core}$	0	**1**
`Graph Colouring` $_{core}$	**7**	4

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/`

IDPd3: logic based visualisation

`dtai.cs.kuleuven.be/krr/idp-ide/?present=SudokuVisualisatie`

- 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 $\mathfrak{A}$. De solution is $sudoku^{\mathfrak{A}}$. 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 $\mathfrak{A}$. Δ-model expansion on T_D3 and input structure $\mathfrak{A}$ 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 Cell(r,c), and a text object 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 sudoku number at cell (r, c) is the label of the text object..
- that text objects are in front of rectangular objects

Δ-model expansion expands a structure interpreting sudoku into a structure interpreting all these graphical symbols. This is fed into d3.

IDPd3: an example

Input structure

$$sudoku = \{1,1 \to 1; \ldots ; 9,9 \to 4\}$$
$$\ldots$$

$\downarrow$ Δ-model generation

$$d3\_type = \{1, Cell(1,1) \to rect; 1, Text(1,1) \to text; \ldots\}$$
$$d3\_color = \{1, Cell(1,1) \to white; 1, Text(1,1) \to black; \ldots\}$$
$$d3\_x = \{1, Cell(1,1) \to 5; 1, Cell(1,2) \to 10; \ldots\}$$
$$\ldots$$

$\downarrow$ 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,Row,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 Δ-model expansion
  - computing a model of a definition Δ
  - A special case of model expansion
  - No search
  - Can be implemented very differently
  - $=$ View materialisation in deductive databases.

Acces 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
  - In the lecture I show a little video showing a idpd3-generated video with the optimal plan to remove all gold.
- Use Progression for interactive execution.
  - Input: T, structure $\mathfrak{A}$ representing state at time i
  - Output: structure $\mathfrak{A}'$ 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
  - In the lecture I show a little video showing a idpd3-generated video with the optimal plan to remove all gold.
- Use Progression for interactive execution.
  - Input: T, structure $\mathfrak{A}$ representing state at time i
  - Output: structure $\mathfrak{A}'$ representing possible state at time $i+1$.
- Illustration: pacman.

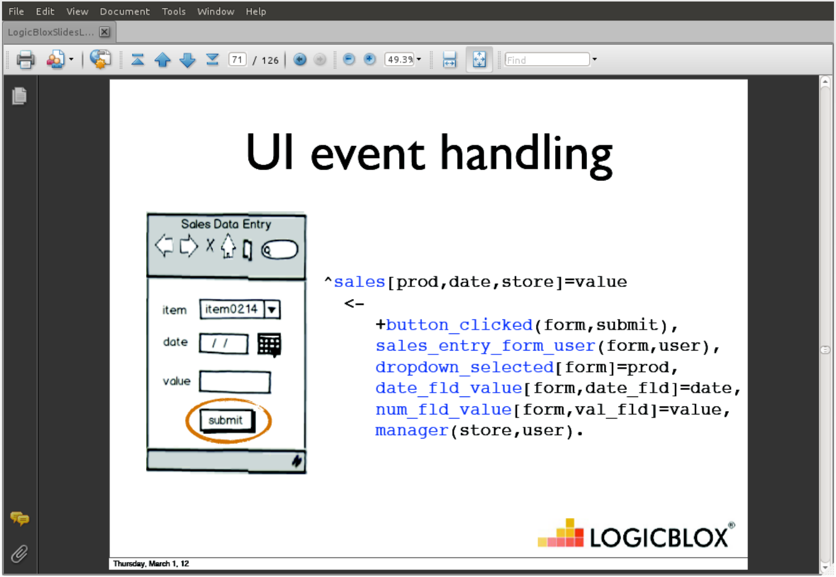
IDP is the only system that we know of that can use the same formal specification to solve both tasks.

The pacman demo:

- Go to the IDP-web page `https://dtai.cs.kuleuven.be/software/idp/`
- Let column: select "Demos"
- Next page contains different visualised demos of IDP. Select bottom middle "More visualised demos.
- Next page, select link below "Pacman".
- Next page, press Run
- A labyrinth with above, a green and black rectangles appear, indicating possible move actions van the pacman. Click the black rectangles and pacman moves in the right direction.
- Due to memory limitations on the server, only a few moves can be played.
- To avoid memory limitations, download the system.

A company running standard software systems on this principle:

- LogicBlox - Datalog :
  http://www.logicblox.com/


File Edit View Document Tools Window Help
LogicBloxSlidesL...
71 / 126
49.3%
Find
UI event handling
Sales Data Entry
item
item0214
date
/ /
value
submit
^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).
LOGICBLOX
Thursday, March 1, 12

some clients

at&t
Argos
Toys"R"Us
PLANIXS
dELiA*s
М.видео mvideo.ru
Walgreens
POLO RALPH LAUREN
Harrods
BEST BUY
McKESSON
Sainsbury's
carhartt.
THE LIMITED
THE Carphone Warehouse
RenaissanceRe

Generic interactive configuration

http://krr.bitbucket.org/autoconfig/

Another application, selectable from the demos webpage (left middle box) is interactive decision enactment = business logic terminology for generic interactive configuration.

- Automatic generation of a window with values for all ground literals of a configuration problem
  - Three-valued structure underneath
- User can select values
  - System propagates user choices
  - Optimal propagation versus approximate propagation
- System expands current partial structure on demand
- System searches model minimizing a cost term on demand
- Generic, incremental:
  - Adding symbols
  - Other vocabularies

The application is a benchmark problem of business rules

- USERV
- Deciding car insurance policy for client.

To be submitted to RuleML.

Advantages compared to business rule systems

- Clear declarative sematincs: formal and informal
- Reasoning backward from desired outcome.
- Increased functionalities
  - computing solutions under uncertainty
  - propagation
  - optimisation
  - explanation (is not implemented)
  - ...

Integrating logic in imperative languages

Joost Vennekens: Lowering the learning curve for declarative programming: a Python API for the IDP system. International Workshop on User-Oriented Logic Programming (IULP 2015) 31st August 2015, Cork (Ireland) CoRR abs/1511.00916

```
idp.Type("Number", range(10))
idp.Function("Given(Square):  Number", d)
idp.Function("Sol(Square):  Number")
idp.Define("Diff(Square, Square)", "lambda x,y:  x !=
y and (SameRow(x,y) or SameCol(x,y) or
SameSmallSq(x,y))")
idp.Constraint("all(Sol(x) == Given(x) for x in
Square if Given(x) != 0)")
idp.Constraint("all(Sol(x) != 0 for x in Square)")
idp.Constraint("all(Sol(x) != Sol(y) for (x,y) in
Diff)")
show(idp.Sol)
```

See webpages:

- `https://dtai.cs.kuleuven.be/software/idp/`
- Online IDE (edit, safe, download, run, many examples)
- Demos
  - map coloring
  - interactive course selection
  - more visualised demos
    - pacman - interactive execution
    - Science Week example : visualisation of errors
- tutorial, manual
- ICLP-contest 2015 solutions

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.