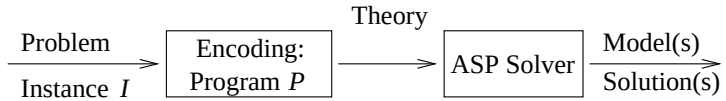
Towards Programming Support Methods for Answer-Set Programs

Hans Tompits

Vienna University of Technology
Institute of Information Systems
Knowledge-Based Systems Group

Context: Answer-Set Programming (ASP)

- Important paradigm for declarative problem solving based on logic programming.
- ASP emerged from research efforts during the 1990s to provide a clear, fully declarative semantics for the negation-as-failure operator (a.k.a. "default negation") for logic programs.
- Basic application schema of ASP:
  - Problems are solved by encoding them in terms of programs such that solutions are determined by the models of the associated programs.



  - Compact, easily maintainable representation.
  - Integration of knowledge representation, database, and search techniques.
  - Handling dynamic, knowledge intensive applications: data, defaults, exceptions, closures, ...

Context (ctd.)

Many applications of ASP

- planning and causal reasoning (Eiter, Erdem, Faber, 2008; Son & Pontelli, 2006)
- diagnosis (Eiter et al., 1999)
- *USA-Advisor* (Gelfond, 2002):
  - decision support system for controlling the Space Shuttle in the presence of multiple system failures
- bioinformatics (Erdem et al., 2009)
- linguistics (history of Indo-European languages; Erdem et al., 2003)
- e-tourism (Leone et al.)
- music composition (ANTON system, Univ. of Bath, Boenn et al., 2009)
- Semantic-Web reasoning (Rosati, 2006; Eiter et al., 2008;)
- qualitative decision theory (Brewka, 2006)
- and many more

Context (ctd.)

- Sophisticated solver technology available:
  - `DLV` (Vienna Univ. of Technology, Univ. of Calabria)
  - `Clasp` (University of Potsdam)
  - `Smodels`, `GnT` (Aalto University)
  - `ASSAT` (Hong Kong University of Science and Technology)
  - ⋮

Challenges & Project Overview

➤ One main challenge for a wider acceptance of ASP:

- Lack of *engineering tools* for developing answer-set programs!
- Such tools would be helpful towards a wider acceptance of ASP—especially outside academia.

➥ FWF project

"Methods and Methodologies for Developing Answer-Set Programs"

addresses these engineering requirements in a systematic manner.

$\Longrightarrow$ Internal project name:

The m&m's of ASP

Research Topics

Systematic program development

Testing methodologies

Debugging

Implementations and IDE

Testing Methodologies

- "Testing is the process of executing a program with the intent of finding errors." (Myers 1979, Art of Software Testing).
  - Goal: Generate test cases with high potential to reveal faults.
- Prominent methods from software testing:
  - *White-box testing*
    - Aims at deriving test cases that cover all possible "paths" through a software component.
  - *Black-box testing*
    - Derive test cases from the specification but abstract from the internal structure of a program component.

Testing Methodologies (ctd.)

We introduced a framework for white-box testing for ASP (joint work with Aalto University; ECAI 2010, LPNMR 2011).

- *Coverage* measures the degree to which extent test inputs cover the logic (source code) of a program.
- In conventional testing:
  - *Path coverage*: all execution paths in the control-flow-graph are exercised at least once.
  - *Branch coverage*: all edges in the control-flow-graph are traversed at least once.

↪ We defined basic coverage notions in ASP corresponding to path and branch coverage.

Testing Methodologies (ctd.)

- Example for analogue of branch coverage: *rule coverage*
  - Generally, in our setting, a *test case* is a pair $\langle I, O \rangle$, where
    - I, the *input*, is a set of facts, and
    - O, the *output*, is a collection of expected answer sets.
  - For any program P and input I of P, a rule $r \in P$ is
    - *positively covered by* I if the body of r is true under some answer set of $P \cup I$, and
    - *negatively covered by* I if the body of r is false under some answer set of $P \cup I$.
- We furthermore laid down basic techniques for test automation using ASP itself, for *deciding test verdicts*, *determining positive/negative coverage*, and *generating test cases with increasing coverage*.

Testing Methodologies (ctd.)

- In current work we also deal with black-box testing for ASP.
- We introduced an annotation language, `Lana`, for ASP (together with M. de Vos, Univ. of Bath, and D. Kisem, Sabanci Univ.)
  - allows to group rules into coherent blocks
  - furthermore, provision to specify *language signatures*, *types*, *pre- and postconditions*, and *unit tests* for such blocks is realised.
    - Preconditions are expected to hold for any input of a block, while postconditions have to hold for any output.
    - Together, they can be used to verify correctness of an ASP encoding with respect to such assertions.
- Unit testing is realised in the implementation `ASPUnit`.

Debugging

Develop methods to *debug answer-set programs*.

- Question: What constitutes an error in an answer-set program?
  - The syntactical structure of programs is simple.
  - Syntactically correct programs have well-defined semantics.

- A typical error in ASP is a *mismatch* between the *intended* and the *actual semantics* of a program.

  $\Longrightarrow$ We focus on such *conceptual errors* of programs!

Debugging (ctd.)

- We developed different debugging methods for ASP (Oetsch, Pührer, & Tompits, 2010, 2011).
- One addresses the following debugging question: Why is a given interpretation not an answer set?
  - This is answered based on a *meta-programming approach*.
    - I.e., a program over a meta-language manipulates another program over an object language.
    - Here: both the meta-language and the object language is ASP!
    - Obvious advantage: we can exploit existing ASP solver technology to compute debugging requests.
- A second debugging approach is based on *stepping*, providing an interactive "tracing-like" method allowing step-wise reconstruction of answer sets.

Debugging (ctd.)

Step-by-step execution is popular in procedural programming.

- programmer gets insight in actual behaviour of the program
- stepping is a central debugging feature of many integrated development environments (IDEs)

Debugging (ctd.)

- We devised a framework that allows for
  - building an answer set the user has in mind by
  - stepwise adding further active rules.
- Realised by emulating a bottom-up generation of an answer set
  - considered interpretation grows monotonically
  - user serves as an oracle, choosing rules considered to be active
  - bugs can be found when computation deviates from expectations.

Implementations and IDE

SeaLion—an integrated development environment for ASP.

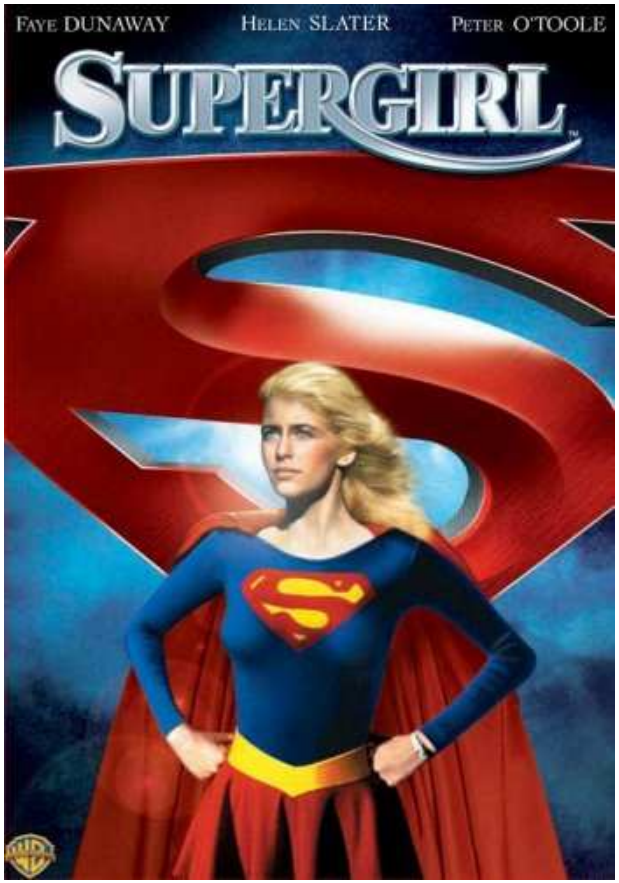
J. Jolly Leo II, the LION fighting for the advancement of Software Engineering for Answer-Set Programming

Implementations and IDE (ctd.)

- Implemented as plugin of the popular *Eclipse platform*
- Aimed for supporting a developer to write, evaluate, debug, and test answer-set programs
- Current main features
  - source-code editors for the languages of the most prominent ASP solvers `Gringo`/`Clasp` and `DLV`
  - syntax highlighting, syntax checking
  - visual program outline
  - initial refactoring possibilities
  - support for running solvers
  - visualisation and visual editing of answer sets $\longrightarrow$ `Kara` plugin
  - documentation generation $\longrightarrow$ `ASPDoc`, similar to Javadoc, based on `Lana`, provides HTML document of program.

Implementations and IDE (ctd.)

Visualisation tool `Kara`:



- Name inspired by "Kara Zor-El", the Kryptonian name of Supergirl.
  $\Longrightarrow$ Kryptonians have visual superpowers on Earth!

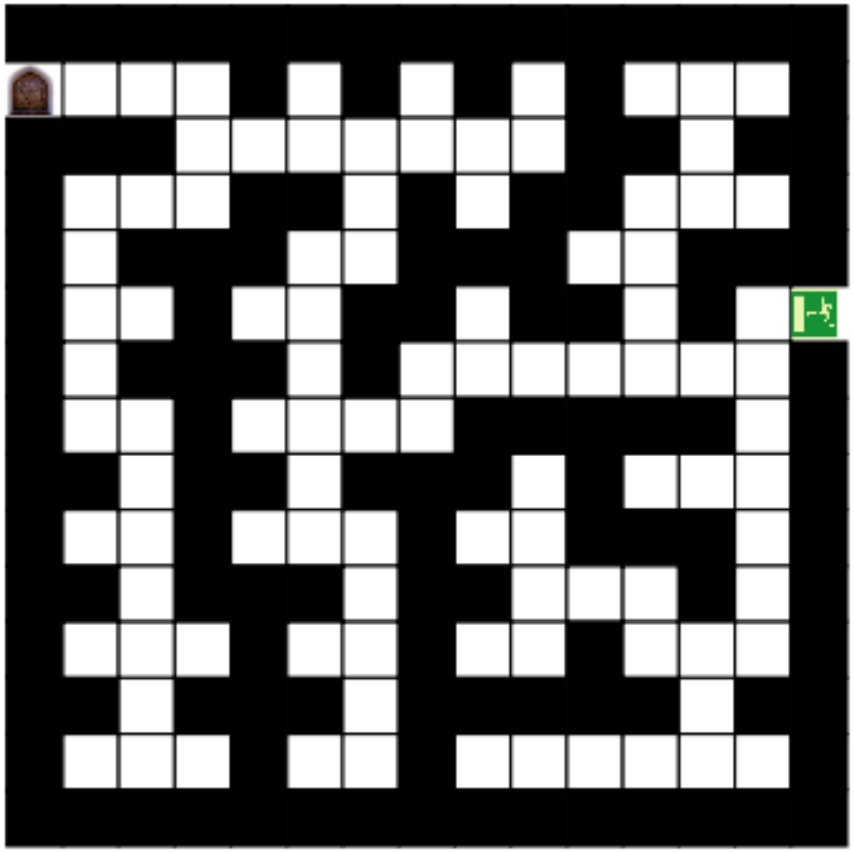
Implementations and IDE (ctd.)

- Reading the answer-set output of a solver is often a difficult task because of large answer sets.
- Instead of output like this:

```
entrance(1,2) exit(5,1) row(1) row(2) row(3) row(4) row(5)
row(6) row(7) ... adjacent(2,11,2,12) adjacent(2,12,2,13)
adjacent(2,13,2,14) adjacent(2,14,2,15) adjacent(3,1,3,2)
... wall(3,10) wall(3,11) wall(3,12) wall(3,13) wall(4,3)
wall(4,7) wall(4,10) wall(4,12) wall(5,3) wall(5,4)
wall(5,5) wall(5,9) wall(5,10) wall(5,12) ... empty(13,14)
empty(14,2) empty(14,3) empty(14,4) empty(14,5) empty(14,6)
empty(14,7) empty(14,8) empty(14,9) empty(14,10)
empty(14,12) empty(14,14)
```

Implementations and IDE (ctd.)

- `Kara` provides a visualisation like this:



- Arguably, it is much easier to *check the correctness* of the solution this way!

Implementations and IDE

- `Kara` offers two modes of visualisation:
  1. *problem-specific visualisation* defined by an answer-set program, following the approach of previous tools
     - `ASPVIZ`—Cliffe, De Vos, Brain and Padget (University of Bath, 2008)
     - `IDPDraw`—Wittocx (Katholieke Universiteit Leuven, 2009)
  2. *generic visualisation* as a hypergraph
- Also: *Visual editing* of answer sets supported
  - the textual representation of the answer set can be edited by modifying its visualisation

Postlude—Vienna and Logic

Tracing back to . . .

Albert von Sachsen (1320–1390), first rector of University of Vienna (1365) and acknowledged medieval scholar on logic.

Postlude—Vienna and Logic (ctd.)

- Gottfried Wilhelm Leibniz (1646-1716), grandfather of automated deduction.
- He spent two years in Vienna, trying to form a scientific academy, and chatting with Prince Eugen about (most probably) almost everything.

Postlude—Vienna and Logic (ctd.)

. . . and up to the Viennese circle around Moritz Schlick in the 1930s and Wittgenstein, Popper, Gödel, and others.

Haus Wittgenstein (1926-1928)