Practical Tools for Knowledge Representation and Nonmonotonic Reasoning

AAAI 2000 — Tutorial SA2

Ilkka Niemelä
Helsinki University of Technology
Department of Computer Science and Engineering
Laboratory for Theoretical Computer Science
P.O.Box 5400, FIN-02015 HUT, Finland `http://www.tcs.hut.fi/~ini/`
e-mail: `Ilkka.Niemela@hut.fi`

Mirosław Truszczyński
Department of Computer Science
University of Kentucky
Lexington, KY 40506–0046
`http://www,cs.engr.uky.edu/~mirek/`
e-mail: `mirek@cs.uky.edu`

1 Overview

This handout contains supporting materials for the AAAI 2000 tutorial SA2. It consists of:

1. information on how to obtain software discussed in the tutorial
2. smodels programs to find a Hamilton cycle in a directed graph
3. smodels programs to solve the missionaries and cannibals puzzle
4. a complete description of the French-Phrases-Italian-Soda puzzle
5. an smodels program encoding this puzzle
6. references to additional materials
7. complete set of slides

2 Where to get the software?

This tutorial discusses knowledge representation tools based on a new computational paradigm of Anser Set Programming (ASP). ASP has its roots in nonmonotonic reasoning and, especially, in logic programming with negation under the stable model semantics. There are several implementations of this paradigm. The one discussed in detail in the tutorial is *smodels*. It has been developed in Helsinki University of Technology by a group led by Dr. Ilkka Niemelä. Two other systems are *dlv* and *dcs*. Dlv has been developed at the Technical University of Vienna by Nicola Leone and his colaborators. Dcs has been developed at the University of Kentucky by Deborah East and Mirosław Truszczynski. The urls for all these systems are given below.

Smodels: `http://www.tcs.hut.fi/Software/smodels/`
Dlv: `http://www.dbai.tuwien.ac.at/dlv/`
Dcs: `http://www.cs.engr.uky.edu/dcs/`

3 Hamilton cycle program

The program must contain:

- the description of an input graph
- rules describing the constraints

Format for the input:

```
vtx(a).                       %for every vertex a
edge(a,b),                    %for every edge {a,b}
initialvtx(a0).               %for some vertex a0
```

Program:

```
% GENERATOR
   { hc(X,Y) } :- edge(X,Y).

% TESTER - each vertex has at most one outgoing edge
   :- 2 { hc(X,Y):edge(X,Y) }, vtx(X).

% TESTER - each vertex has at most one incoming edge
   :- 2 { hc(X,Y):edge(X,Y) }, vtx(Y).

% TESTER - all vertices are visited
   :- vtx(X), not r(X).
   r(Y) :- hc(X,Y), edge(X,Y), initialvtx(X).
   r(Y) :- hc(X,Y), edge(X,Y), r(X), not initialvtx(X).
```

4 Missionaries and cannibals

There are 3 cannibals and 3 missionaries on the left bank of a river. They have a boat. The boat has capacity 2. If at any time there are more cannibals than missionaries on either bank, the cannibals eat missionaries. The goal is to arrange the way of moving all 6 of them to the right bank.

State of the problem is given by a quadruple (CC,MM,B,T). It gives the number of cannibals (CC) and missionaries on the left bank of the river at time T. B=1 indicates that boat is on the left bank. B=0 indicates that the boat is on the right bank

Smodels Program for Missionaries-and-Cannibals puzzle (basic rules only):

```
%  Types are defined here

   time(0..12).         %  specifies time range (number of moves) in
                        %  which the solution is to be found
   number(0..3).        %  specifies allowed number of cannibals (or
                        %  missionaries on a bank
   boatcap(0..2).       %  specifies allowed numbers of people in the boat

%  Initial state

   state(3,3,1,0).

%  Define safe states

   safestate(CC,MM) :- number(CC;MM), MM==0.
```

```
   safestate(CC,MM) :- number(CC;MM), MM>=CC.

%  Select a valid move at time T, given a current state. Use
%  move - othermove construction. Make sure no moves are made
%  after goal is reached

   move(C,M,T) :- number(CC;MM), boatcap(C;M), time(T),
               state(CC,MM,1,T),
               1 <= C+M, C+M <= 2, C <= CC, M <= MM,
               safestate(CC-C,MM-M),
               safestate(3-CC+C,3-MM+M),
               not othermove(C,M,T),
               not goal(T).

   move(C,M,T) :- number(CC;MM), boatcap(C;M), time(T),
               state(CC,MM,0,T),
               1 <= C+M, C+M <= 2, C <= 3-CC, M <= 3-MM,
               state(CC,MM,0,T),
               safestate(CC+C,MM+M),
               safestate(3-CC-C,3-MM-M),
               not othermove(C,M,T),
               not goal(T).

   othermove(C,M,T) :- boatcap(C;C1;M;M1), time(T),
               move(C1,M1,T), C1!=C.

   othermove(C,M,T) :- boatcap(C;C1;M;M1), time(T),
               move(C1,M1,T), M1!=M.

%  Compute the next state
```

```
  state(CC-C,MM-M,0,T+1) :- number(CC;MM), boatcap(C;M), time(T),
                            state(CC,MM,1,T), move(C,M,T).

  state(CC+C,MM+M,1,T+1) :- number(CC;MM), boatcap(C;M), time(T),
                            state(CC,MM,0,T), move(C,M,T).

% Determine that the goal has been reached at time T

   goal(T) :- time(T), state(0,0,0,T).

% Determine that the goal was reached

   ok :- time(T), goal(T).

% Kill all plans which do not lead to the goal

   :- not ok.
```

Smodels Program for Missionaries-and-Cannibals puzzle (using cardinality rules):

```
% Types are defined here

   time(0..12).          % specifies time range (number of moves) in
                         % which the solution is to be found
   number(0..3).         % specifies allowed number of cannibals (or
                         % missionaries on a bank
   boatcap(0..2).        % specifies allowed numbers of people in the boat

% Initial state

   state(3,3,1,0).

% Define safe states

   safestate(CC,MM) :- number(CC;MM), MM==0.
   safestate(CC,MM) :- number(CC;MM), MM>=CC.

% Select a valid move at time T, given a current state

good1(CC,MM,C,M) :- boatcap(C;M), number(CC;MM), time(T),
                    1 <= C+M,
                    C+M <= 2,
                    C <= CC,
                    M <= MM,
                    safestate(CC-C,MM-M),
                    safestate(3-CC+C,3-MM+M).
```

```
1 { move(C,M,T): boatcap(C): boatcap(M):
                 good1(CC,MM,C,M)} 1 :- time(T), number(CC;MM),
                 state(CC,MM,1,T),
                 not goal(T).

good0(CC,MM,C,M) :- boatcap(C;M), number(CC;MM), time(T),
                    1 <= C+M,
                    C+M <= 2,
                    C <= 3-CC,
                    M <= 3-MM,
                    safestate(CC+C,MM+M),
                    safestate(3-CC-C,3-MM-M).

1 { move(C,M,T): boatcap(C):
                 boatcap(M):
                 good0(CC,MM,C,M)} 1 :- time(T), number(CC;MM),
                 state(CC,MM,0,T),
                 not goal(T).

%  Compute the next state

  state(CC-C,MM-M,0,T+1) :- number(CC;MM), boatcap(C;M), time(T),
                            state(CC,MM,1,T), move(C,M,T).

  state(CC+C,MM+M,1,T+1) :- number(CC;MM), boatcap(C;M), time(T),
                            state(CC,MM,0,T), move(C,M,T).

%  Determine that the goal has been reached at time T

   goal(T) :- time(T), state(0,0,0,T).
```

```
% Determine that the goal was reached

  ok :- time(T), goal(T).

% Kill all plans which do not lead to the goal

  :- not ok.
```

5 French Phrases, Italian Soda

This puzzle is due to Mark T. Zegarelli and was published in *Dell Logic Puzzles*, p. 43, June 1999.

Here is the text of the puzzle:

Claude looks forward to every Wednesday night, for this is the night he can speak in his native language to the other members of the informal French club. Last week, Claude and five other people (three women named Jeanne, Kate and Liana, and two men named Martin and Robert) shared a circular table at their regular meeting place, the Café du Monde. Claude found this past meeting to be particularly interesting, as each of the six people described an upcoming trip that he or she is planning to take to a different French-speaking part of the world. During the discussion, each person sipped a different flavor of Italian soda, a specialty at the café. Using the following clues, can you match each person with his or her eat (numbered one through six [circularly]) and determine the flavor of soda that each person drank, as well as the place that each plans to visit?

1. The person who is planning a trip to Quebec, who drank either blueberry or lemon soda, didn't sit in seat number one.

2. Robert, who didn't sit next to Kate, sat directly across from the person who drank peach soda.

3. Liana sat immediately between the person who drank tangelo soda and the person who is going to visit Tahiti, in some order.

4. Jeanne didn't sit next to the person who enjoyed the kiwi soda.

5. The three men are the person who is going to Haiti, the one in seat number three, and Claude's brother.

6. The person who sipped grapefruit soda sat immediately between Claude and the person who is arranging a trip to Martinique, in some order.

7. The person who has a plane ticket to Belgium, who sat in seat four or seat five, didn't order the tangelo soda.

8. The three people who sat in even-numbered seats are Kate, Claude and a person who didn't drink lemon soda, in some order.

9. The person who is going to visit the Ivory Coast sat immediately between two men.

6 Smodels program for the French-Phrases-Italian-Soda puzzle

```
% Problem-based constraints
    %  REQUIRED men: claude martin robert
         gender_person(men,claude)  .
         gender_person(men,martin)  .
         gender_person(men,robert)  .
    %  REQUIRED women: jeanne kate liana
         gender_person(women,jeanne)  .
         gender_person(women,kate)  .
         gender_person(women,liana)  .
    %  CONFLICT quebec 1
         :- position_visits(1,quebec).
    %  REQUIRED quebec blueberry OR quebec lemon
         soda_visits(blueberry,quebec) :- not soda_visits(lemon,quebec) .
         soda_visits(lemon,quebec) :- not soda_visits(blueberry,quebec) .
    % OFFSET !+-1 position: robert kate
         differ_position_1(Position1, ((Position1+1-1) mod 6)+1) :-
                position(Position1) .
         differ_position_1(((Position1+1-1) mod 6)+1, Position1) :-
                position(Position1) .
         :- person_position(robert,Position1),
                person_position(kate,Position2),
                position(Position1), position(Position2),
                differ_position_1(Position1,Position2) .
    % OFFSET +-3 position: robert peach
         differ_position_3(Position1, ((Position1+3-1) mod 6)+1) :-
                position(Position1) .
         differ_position_3(((Position1+3-1) mod 6)+1, Position1) :-
```

```
          position(Position1) .
    :- person_position(robert,Position1),
          position_soda(Position2,peach),
          position(Position1),
          position(Position2),
          not differ_position_3(Position1,Position2) .
% CONFLICT tangelo tahiti
    :- soda_visits(tangelo,tahiti).
% OFFSET +-1 position: liana tangelo
    :- person_position(liana,Position1),
          position_soda(Position2,tangelo),
          position(Position1),
          position(Position2),
          not differ_position_1(Position1,Position2) .
% OFFSET +-1 position: liana tahiti
    :- person_position(liana,Position1),
          position_visits(Position2,tahiti),
          position(Position1),
          position(Position2),
          not differ_position_1(Position1,Position2) .
% OFFSET !+-1 position: jeanne kiwi
    :- person_position(jeanne,Position1),
          position_soda(Position2,kiwi),
          position(Position1),
          position(Position2),
          differ_position_1(Position1,Position2) .
% VAR x: soda
    soda_x(X) :- soda(X), not other_soda_x(X) .
    other_soda_x(X) :- soda(X;Y), soda_x(Y), X != Y .
% REQUIRED men: haiti 3 x
```

```
    gender_visits(men,haiti)  .
    gender_position(men,3)  .
    gender_soda(men,X) :- soda(X), soda_x(X) .
% CONFLICT haiti 3 x
    :- soda_visits(X,haiti), soda(X), soda_x(X).
    :- position_visits(3,haiti).
    :- position_soda(3,X), soda(X), soda_x(X).
% CONFLICT x claude
    :- person_soda(claude,X), soda(X), soda_x(X).
% CONFLICT claude martinique
    :- person_visits(claude,martinique).
% OFFSET +-1 position: grapefruit claude
    :- position_soda(Position1,grapefruit),
           person_position(claude,Position2),
           position(Position1),
           position(Position2),
           not differ_position_1(Position1,Position2) .
% OFFSET +-1 position: grapefruit martinique
    :- position_soda(Position1,grapefruit),
           position_visits(Position2,martinique),
           position(Position1),
           position(Position2),
           not differ_position_1(Position1,Position2) .
% REQUIRED belgium 4 OR belgium 5
    position_visits(4,belgium) :- not position_visits(5,belgium) .
    position_visits(5,belgium) :- not position_visits(4,belgium) .
% CONFLICT belgium tangelo
    :- soda_visits(tangelo,belgium).
% VAR y: person
    person_y(X) :- person(X), not other_person_y(X) .
```

```
    other_person_y(X) :- person(X;Y), person_y(Y), X != Y .
% MATCH 2 4 6, kate claude y
    :- not person_position(kate,2),
           not person_position(claude,2),
           not person_position(Y,2), person(Y), person_y(Y).
    :- not person_position(kate,4),
           not person_position(claude,4),
           not person_position(Y,4), person(Y), person_y(Y).
    :- not person_position(kate,6),
           not person_position(claude,6),
           not person_position(Y,6), person(Y), person_y(Y).
    :- not person_position(kate,2),
           not person_position(kate,4),
           not person_position(kate,6).
    :- not person_position(claude,2),
           not person_position(claude,4),
           not person_position(claude,6).
    :- not person_position(Y,2), person(Y), person_y(Y),
           not person_position(Y,4), person(Y), person_y(Y),
           not person_position(Y,6), person(Y), person_y(Y).
% CONFLICT y lemon
    :- person_soda(Y,lemon), person(Y), person_y(Y).
% VAR a: person
    person_a(X) :- person(X), not other_person_a(X) .
    other_person_a(X) :- person(X;Y), person_a(Y), X != Y .
% VAR b: person
    person_b(X) :- person(X), not other_person_b(X) .
    other_person_b(X) :- person(X;Y), person_b(Y), X != Y .
% REQUIRED men: a b
    gender_person(men,A) :- person(A), person_a(A) .
```

```
    gender_person(men,B) :- person(B), person_b(B) .
% OFFSET 1 position: ivory a
    person_position(A,((Position1+1-1) mod 6)+1) :-
           position_visits(Position1,ivory),
           position(Position1),
           position(Position1), person(A), person_a(A) .
% OFFSET 1 position: b ivory
    position_visits(((Position1+1-1) mod 6)+1,ivory) :-
           person_position(B,Position1),
           position(Position1),
           person(B), person_b(B),
           position(Position1) .

% Members
    visits(quebec;tahiti;haiti;martinique;belgium;ivory) .
    position(1;2;3;4;5;6) .
    gender(men;women) .
    soda(blueberry;lemon;peach;tangelo;kiwi;grapefruit) .
    person(claude;jeanne;kate;liana;martin;robert) .

% existence: one hit per row, one hit per column in each table
    1 {position_visits(Position,Visits):position(Position)} 1 :-
           visits(Visits) .
    1 {position_visits(Position,Visits):visits(Visits)} 1 :-
           position(Position) .
    1 {position_soda(Position,Soda):position(Position)} 1 :-
           soda(Soda) .
    1 {position_soda(Position,Soda):soda(Soda)} 1 :-
           position(Position) .
    1 {gender_visits(Gender,Visits):gender(Gender)} 1 :-
```

```
          visits(Visits) .
    1 {gender_position(Gender,Position):gender(Gender)} 1 :-
          position(Position) .
    1 {gender_soda(Gender,Soda):gender(Gender)} 1 :-
          soda(Soda) .
    1 {gender_person(Gender,Person):gender(Gender)} 1 :-
          person(Person) .
    1 {soda_visits(Soda,Visits):soda(Soda)} 1 :-
          visits(Visits) .
    1 {soda_visits(Soda,Visits):visits(Visits)} 1 :-
          soda(Soda) .
    1 {person_visits(Person,Visits):person(Person)} 1 :-
          visits(Visits) .
    1 {person_visits(Person,Visits):visits(Visits)} 1 :-
          person(Person) .
    1 {person_position(Person,Position):person(Person)} 1 :-
          position(Position) .
    1 {person_position(Person,Position):position(Position)} 1 :-
          person(Person) .
    1 {person_soda(Person,Soda):person(Person)} 1 :-
          soda(Soda) .
    1 {person_soda(Person,Soda):soda(Soda)} 1 :-
          person(Person) .

% closures of predicates that connect attributes
    position_visits(Position,Visits) :-
          position(Position), visits(Visits), soda(Soda),
          position_soda(Position,Soda) ,
          soda_visits(Soda,Visits) .
    position_visits(Position,Visits) :-
```

```
        position(Position), visits(Visits), person(Person),
        person_position(Person,Position) ,
        person_visits(Person,Visits) .
position_soda(Position,Soda) :-
        position(Position), soda(Soda), visits(Visits),
        position_visits(Position,Visits) ,
        soda_visits(Soda,Visits) .
position_soda(Position,Soda) :-
        position(Position), soda(Soda), person(Person),
        person_position(Person,Position) ,
        person_soda(Person,Soda) .
gender_visits(Gender,Visits) :-
        gender(Gender), visits(Visits), position(Position),
        gender_position(Gender,Position) ,
        position_visits(Position,Visits) .
gender_visits(Gender,Visits) :-
        gender(Gender), visits(Visits), soda(Soda),
        gender_soda(Gender,Soda) ,
        soda_visits(Soda,Visits) .
gender_visits(Gender,Visits) :-
        gender(Gender), visits(Visits), person(Person),
        gender_person(Gender,Person) ,
        person_visits(Person,Visits) .
gender_position(Gender,Position) :-
        gender(Gender), position(Position), visits(Visits),
        gender_visits(Gender,Visits) ,
        position_visits(Position,Visits) .
gender_position(Gender,Position) :-
        gender(Gender), position(Position), soda(Soda),
        gender_soda(Gender,Soda) ,
```

```
        position_soda(Position,Soda) .
gender_position(Gender,Position) :-
        gender(Gender), position(Position), person(Person),
        gender_person(Gender,Person) ,
        person_position(Person,Position) .
gender_soda(Gender,Soda) :-
        gender(Gender), soda(Soda), visits(Visits),
        gender_visits(Gender,Visits) ,
        soda_visits(Soda,Visits) .
gender_soda(Gender,Soda) :-
        gender(Gender), soda(Soda), position(Position),
        gender_position(Gender,Position) ,
        position_soda(Position,Soda) .
gender_soda(Gender,Soda) :-
        gender(Gender), soda(Soda), person(Person),
        gender_person(Gender,Person) ,
        person_soda(Person,Soda) .
gender_person(Gender,Person) :-
        gender(Gender), person(Person), visits(Visits),
        gender_visits(Gender,Visits) ,
        person_visits(Person,Visits) .
gender_person(Gender,Person) :-
        gender(Gender), person(Person), position(Position),
        gender_position(Gender,Position) ,
        person_position(Person,Position) .
gender_person(Gender,Person) :-
        gender(Gender), person(Person), soda(Soda),
        gender_soda(Gender,Soda) ,
        person_soda(Person,Soda) .
soda_visits(Soda,Visits) :-
```

```
        soda(Soda), visits(Visits), position(Position),
        position_soda(Position,Soda) ,
        position_visits(Position,Visits) .
soda_visits(Soda,Visits) :-
        soda(Soda), visits(Visits), person(Person),
        person_soda(Person,Soda) ,
        person_visits(Person,Visits) .
person_visits(Person,Visits) :-
        person(Person), visits(Visits), position(Position),
        person_position(Person,Position) ,
        position_visits(Position,Visits) .
person_visits(Person,Visits) :-
        person(Person), visits(Visits), soda(Soda),
        person_soda(Person,Soda) ,
        soda_visits(Soda,Visits) .
person_position(Person,Position) :-
        person(Person), position(Position), visits(Visits),
        person_visits(Person,Visits) ,
        position_visits(Position,Visits) .
person_position(Person,Position) :-
        person(Person), position(Position), soda(Soda),
        person_soda(Person,Soda) ,
        position_soda(Position,Soda) .
person_soda(Person,Soda) :-
        person(Person), soda(Soda), visits(Visits),
        person_visits(Person,Visits) ,
        soda_visits(Soda,Visits) .
person_soda(Person,Soda) :-
        person(Person), soda(Soda), position(Position),
        person_position(Person,Position) ,
```

```
position_soda(Position,Soda) .
```

References

[AAdR99] Rafael Accorsi, Carlos Areces, and Maarten de Rijke. Towards feature interaction via stable models. In *Proceedings of the Second Workshop on Formal Methods*, Florianpolis, Brasil, October 1999.

[ABS00] T. Aura, M. Bishop, and D. Sniegowski. Analyzing single-server network inhibition. In *Proceedings of teh IEEE Computer Security Foundations Workshop*, Cambridge, UK, July 2000. IEEE Computer Society Press.

[DNK97] Y. Dimopoulos, B. Nebel, and J. Koehler. Encoding planning problems in non-monotonic logic programs. In *Proceedings of the Fourth European Conference on Planning*, pages 169–181, Toulouse, France, September 1997. Springer-Verlag.

[ET00] D. East and M. Truszczyński. DATALOG with Constraints. In *Proccedings of the Seventeenth National Conference on Artificial Intelligence(AAAI-2000)*, July 2000.

[Hel99] K. Heljanko. Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe Petri nets. In *Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 240–254, Amsterdam, the Netherlands, March 1999. Springer-Verlag.

[HMN00] M. Hietalahti, F. Massacci, and I. Niemelä. DES: a challenge problem for nonmonotonic reasoning systems. In *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning*, Breckenridge, Colorado, USA, April 2000. cs.AI/0003039.

[JNSY00] T. Janhunen, I. Niemelä, P. Simons, and J. You. Unfolding partiality and disjunctions in stable model semantics. In A.G. Cohn, F. Guinchiglia, and B. Selman, editors, *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning*, Breckenridge, Colorado, USA, April 2000. Morgan Kaufmann Publishers. Accepted for publication.

[Lif99] V. Lifschitz. Answer set planning. In *Proceedings of the 16th International Conference on Logic Programming*, pages 25–37, Las Cruces, New Mexico, December 1999. The MIT Press.

[Lif00] V. Lifschitz. Missionaries and cannibals in the causal calculator, A.G. Cohn, F. Giunchiglia and B. Selman", In *Principles of Knowledge Representation and Reasoning, Proceedings of the Seventh International Conference (KR2000)*, April 2000, pp. 85–96. Morgan Kaufmann Publishers.

[LRS98] Xinxin Liu, C.R. Ramakrishnan, and Scott A. Smolka. Fully local and efficient evaluation of alternating fixed points. In Bernhard Steffen, editor, *Proceedings of the 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 5–19, Lisbon, Portugal, March/April 1998. Springer-Verlag.

[MT99] Marek, W., and Truszczyński, M. Stable models and an alternative logic programming paradigm. In Apt, K.; Marek, W.; Truszczyński, M.; and Warren, D., eds., *The Logic Programming Paradigm: a 25-Year Perspective*. Springer Verlag, 1999. 375–398.

[Nie99] I. Niemelä. Logic programming with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3,4):241–273, 1999.

[NS96] I. Niemelä and P. Simons. Efficient implementation of the well-founded and stable model semantics. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 289–303, Bonn, Germany, September 1996. The MIT Press.

[NS97] I. Niemelä and P. Simons. Smodels – an implementation of the stable model and well-founded semantics for normal logic programs. In *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning*, pages 420–429, Dagstuhl, Germany, July 1997. Springer-Verlag.

[NSS99] I. Niemelä, P. Simons, and T. Soininen. Stable model semantics of weight constraint rules. In *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 317–331, El Paso, Texas, USA, December 1999. Springer-Verlag.

[Sim97] P. Simons. Towards constraint satisfaction through logic programs and the stable model semantics. Research report A47, Helsinki University of Technology, Helsinki, Finland, August 1997. Available at http://www.tcs.hut.fi/pub/reports/A47.ps.gz.

[Sim99] P. Simons. Extending the stable model semantics with more expressive rules. In *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 305–316, El Paso, Texas, USA, December 1999. Springer-Verlag.

[Sim00] P. Simons. Extending and implementing the stable model semantics. Doctoral Dissertation. Research report A58, Helsinki University of Technology, Helsinki, Finland, April 2000.

[SN99] T. Soininen and I. Niemelä. Developing a declarative rule language for applications in product configuration. In Gopal Gupta, editor, *Proceedings of the First International Workshop on Practical Aspects of Declarative Languages*, pages 305–319, San Antonio, Texas, January 1999. Springer-Verlag.

[SNG99] T. Soininen, I. Niemelä, and E. Gelle. A fixpoint definition of dynamic constraint satisfaction. In Joxan Jaffar, editor, *Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming*, pages 419–433, Alexandria, Virginia, USA, October 1999. Springer-Verlag.

[Syr99] T. Syrjänen. A rule-based formal model for software configuration. Research report A55, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Helsinki, Finland, December 1999.

[Var99] K. Varpaaniemi. Stable models for stubborn sets. In H.-D. Burkhard, L. Czaja, H.S. Nguyen, and P.H. Starke, editors, *Concurrency, Specification and Programming (Proceedings of the CS&P'99 Workshop)*, pages 263–274, Warsaw, Poland, 1999.

7 Slides