ANSWER SET PROGRAMMING

Chitta Baral

Department of Computer Science and Engg.
Arizona State University
Tempe, AZ 85287
chitta@asu.edu

http://www.public.asu.edu/~cbaral

August 2, 2003

INTRODUCTION

Declarative Programming

- Answer set programming is a kind of declarative programming.
- Declarative programming: Specify *what* you want, not *How*. Leave the *How* part to the system to figure out.
- Prolog (programming in logic) was one of the first declarative programming language.

Declarative Programming: an example

- Linear programming
  $max\ f(X_1, \ldots, X_n)$
  such that
  $g_1(X_1, \ldots, X_n) \leq k_1$
  $\vdots$
  $g_m(X_1, \ldots, X_n) \leq k_m$

Prolog vs Answer set programming

- Prolog
  - Problem solving is done by asking a query with a variable.
  - The instantiations of that variables encode the solution of the problem.
  - Prolog has many non-declarative features.
- Answer set programming (AnsProlog)
  - Problem solving is done by computing the “models” (answer sets) of the program.
  - Each answer set encodes a solution of the problem.
  - AnsProlog (also called A-Prolog) is full declarative.

INTRODUCTION TO ANSPROLOG BY EXAMPLES

Horn program 1

$$
\begin{array}{c}
p \leftarrow a. \\
q \leftarrow b. \\
a \leftarrow .
\end{array}
$$

- $\{a, p\}$ is an answer set of the above program.
- $\{a, p, b, q\}$ is not an answer set of the above program.
- $\{a\}$ is not an answer set of the above program.
- $\{a, p\}$ is **the only** answer set of the above program.

Horn program 2

$$p \leftarrow p.$$
$$q \leftarrow .$$

- $\{q\}$ is an answer set of the above program.
- $\{p, q\}$ is not an answer set of the above program.
- $\{p\}$ is not an answer set of the above program.
- $\{\}$ is not an answer set of the above program.
- $\{q\}$ is **the only** answer set of the above program.

Horn program 3: Ancestors and parents

$$
\begin{gathered}
anc(X,Y) \leftarrow par(X,Y). \\
anc(X,Y) \leftarrow anc(Z,Y), par(X,Z). \\
par(a,b) \leftarrow . \\
par(b,c) \leftarrow . \\
par(d,e) \leftarrow .
\end{gathered}
$$

- Computing the answer set:
  - Iteration 0: $\{\}$
  - Iteration 1: Add $\{par(a,b), par(b,c), par(d,e)\}$
  - Iteration 2: Add $\{anc(a,b), anc(b,c), anc(d,e)\}$
  - Iteration 3: Add $\{anc(a,c)\}$
  - Iteration 4: Add $\{\}$. STOP.
  - Answer set = $\{par(a,b), par(b,c), par(d,e), anc(a,b), anc(b,c), anc(d,e), anc(a,c)\}$.

- For efficiency purposes: Domains of variables are required in most AnsProlog systems (such as Smodels or Dlv).

$$
\begin{aligned}
&anc(X,Y) \leftarrow domain(X), domain(Y), par(X,Y).\\
&anc(X,Y) \leftarrow domain(X), domain(Y), domain(Z),\\
&\qquad\qquad\qquad anc(Z,Y), par(X,Z).\\
&par(a,b) \leftarrow.\\
&par(b,c) \leftarrow.\\
&par(d,e) \leftarrow.\\
&domain(a) \leftarrow.\\
&domain(b) \leftarrow.\\
&domain(c) \leftarrow.\\
&domain(d) \leftarrow.\\
&domain(e) \leftarrow.
\end{aligned}
$$

- Ordering of rules(top-to-bottom) in the file and ordering of predicates in the body of rule (left-to-right) is important in Prolog. NOT so in AnsProlog.

AnsProlog program 1

$$a \leftarrow \textbf{not}\ b.$$
$$q \leftarrow.$$

- $\{q, b\}$ is not an answer set of the above program.
- $\{q, a, b\}$ is not an answer set of the above program.
- $\{q, a\}$ is **the only** answer set of the above program.

AnsProlog program 2

$$
\begin{gathered}
a \leftarrow \textbf{not } b. \\
b \leftarrow \textbf{not } a.
\end{gathered}
$$

- $\{a\}$ and $\{b\}$ are answer sets of the above program.

$$
\begin{gathered}
a \leftarrow \textbf{not } b. \\
b \leftarrow \textbf{not } a. \\
p \leftarrow a.
\end{gathered}
$$

- $\{p, a\}$ and $\{b\}$ are answer sets of the above program.

Semantics of AnsProlog

- Intuition: A is an answer set of a program Π if using the assumption of A the 'simplification' of the program Π, a Horn program denoted by Π^A, has the only answer set A.
- Similar to solving $x = f(x)$.
  - We need to know what is false to be able to simplify the program to a Horn program where we can reason.
  - But to know what is false we need the answer set.
  - Prolog gives up on this.
  - This will turn out to be very important.
- The Simplification (Gelfond-Lifschitz transformation) of Π with respect to a set of atoms A – denoted by Π^A.

  - Remove a rule from Π if its body contains **not** b, where $b \in A$.
  - From the remaining rules remove from the body of a rule in Π a component '**not** c' if c is not in A.

- Consider $\Pi = \{a \leftarrow \textbf{not } b.; \;\; b \leftarrow \textbf{not } a.\}$.
- Let $A_1 = \{a\}$, $A_2 = \{b\}$, $A_3 = \{\}$, and $A_4 = \{a, b\}$.
- $\Pi^{A_1} = \{a \leftarrow .\}$ has the only answer set $\{a\} = A_1$.
  Hence, A_1 is an answer set of Π.
- $\Pi^{A_2} = \{b \leftarrow .\}$ has the only answer set $\{b\} = A_2$.
  Hence, A_2 is an answer set of Π.
- $\Pi^{A_3} = \{a \leftarrow .; b \leftarrow .\}$ has the only answer set $\{a, b\} = A_4$.
  Hence, A_3 is not an answer set of Π.
- $\Pi^{A_4} = \{\}$ has the only answer set $\{\} = A_3$.
  Hence, A_3 is not an answer set of Π.

AnsProlog programs 3 and 4

$$a \leftarrow \textbf{not}\ b.$$
$$b \leftarrow \textbf{not}\ c.$$
$$d \leftarrow .$$

- Answer set of the above program is $\{d, b\}$.

$$p \leftarrow \textbf{not}\ q, r.$$
$$q \leftarrow \textbf{not}\ p.$$

- Answer set of the above program is $\{q\}$.

AnsProlog programs 5, 6 and 7

$$
\begin{gathered}
p \leftarrow \mathbf{not}\ p, d. \\
r \leftarrow . \\
d \leftarrow .
\end{gathered}
$$

- The above program does not have an answer set.

$$
\begin{gathered}
p \leftarrow \mathbf{not}\ p, d. \\
r \leftarrow \mathbf{not}\ d. \\
d \leftarrow \mathbf{not}\ r.
\end{gathered}
$$

- The only answer set of the above program is $\{r\}$.

$$
\begin{gathered}
p \leftarrow \mathbf{not}\ p. \\
p \leftarrow \mathbf{not}\ d. \\
r \leftarrow \mathbf{not}\ d. \\
d \leftarrow \mathbf{not}\ r.
\end{gathered}
$$

- The only answer set of the above program is $\{r, p\}$.

SIMPLE ANSPROLOG MODULES

Integrity constraints

- To incorporate the integrity constraint
  $\leftarrow l_1, \ldots, l_m, \textbf{not}\ l_{m+1}, \ldots, \textbf{not}\ l_n.$
  to a program Π either of the following approach is fine.
- Using a *new* atom p which is not in Π add the following rule:
  $p \leftarrow \textbf{not}\ p, l_1, \ldots, l_m, \textbf{not}\ l_{m+1}, \ldots, \textbf{not}\ l_n.$
- Using *new* atoms p and q which is not in Π add the following rules:
  $p \leftarrow l_1, \ldots, l_m, \textbf{not}\ l_{m+1}, \ldots, \textbf{not}\ l_n.$
  $q \leftarrow \textbf{not}\ p.$
  $q \leftarrow \textbf{not}\ q.$

Finite enumeration

- Specification: We have atoms $p_1, \ldots, p_n$. We would like to construct a program where for each interpretation of the propositions we have an answer set.
- Program:

$p_1 \leftarrow \textbf{not}\ n\_p_1.$
$n\_p_1 \leftarrow \textbf{not}\ p_1.$
$\vdots$
$p_n \leftarrow \textbf{not}\ n\_p_n.$
$n\_p_n \leftarrow \textbf{not}\ p_n.$

General enumeration but at least one

- Specification: We have a predicate $possible(X)$ that denotes what is possible. Our goal is to enumerate what is possible, with the added stipulation that in each answer set at least one term should be chosen.
- Program:

  $chosen(X) \leftarrow possible(X), \textbf{not}\ not\_chosen(X).$

  $not\_chosen(X) \leftarrow possible(X), \textbf{not}\ chosen(X).$

  $some \leftarrow chosen(X).$

  $inconsistent \leftarrow \textbf{not}\ inconsistent, \textbf{not}\ some.$

Choice: General enumeration with exactly one

- Specification: We have a predicate $possible(X)$ that denotes what is possible. Our goal is to enumerate what is possible, with the stipulation that in each answer set exactly one term should be chosen and we have one answer set for each term.
- Program

  $possible(a) \leftarrow .$
  $possible(b) \leftarrow .$
  $possible(c) \leftarrow .$

  $diff\_chosen\_than(X) \leftarrow chosen(Y), X \neq Y.$
  $chosen(X) \leftarrow possible(X), \textbf{not}\ diff\_chosen\_than(X).$

- The answer sets: $\{p(a), p(b), p(c), c(a), d(b), d(c)\}$, $\{p(a), p(b), p(c), d(a), c(b), d(c)\}$, $\{p(a), p(b), p(c), d(a), d(b), c(c)\}$.

Constrained enumeration

- Specification: Place objects in a rectangular board so that no two objects are in the same row or in the same column.
- Program 1:

$$
\begin{aligned}
not\_chosen(X,Y) \leftarrow\ & row(X), column(Y), row(X'),\\
& chosen(X',Y), X' \neq X.\\
not\_chosen(X,Y) \leftarrow\ & row(X), column(Y), column(Y'),\\
& chosen(X,Y'), Y' \neq Y.\\
chosen(X,Y) \leftarrow\ & row(X), column(Y), \mathbf{not}\ not\_chosen(X,Y).
\end{aligned}
$$

- Each answer set will have a particular placement.

- Program 2:

$$chosen(X,Y) \leftarrow row(X), column(Y), \textbf{not}\ not\_chosen(X,Y).$$
$$not\_chosen(X,Y) \leftarrow row(X), column(Y), \textbf{not}\ chosen(X,Y).$$
$$filled\_row(X) \leftarrow chosen(X,Y).$$
$$filled\_column(Y) \leftarrow chosen(X,Y).$$
$$missing\_row \leftarrow row(X), \textbf{not}\ filled\_row(X).$$
$$missing\_column \leftarrow column(X), \textbf{not}\ filled\_column(X).$$
$$inconsistent \leftarrow \textbf{not}\ inconsistent, missing\_row.$$
$$inconsistent \leftarrow \textbf{not}\ inconsistent, missing\_column.$$
$$\begin{aligned} inconsistent \leftarrow\ & \textbf{not}\ inconsistent, chosen(X,Y), \\ & chosen(X,Z), Y \neq Z. \end{aligned}$$
$$\begin{aligned} inconsistent \leftarrow\ & \textbf{not}\ inconsistent, chosen(X,Y), \\ & chosen(Z,Y), X \neq Z. \end{aligned}$$

Propositional Satisfiability

- $S = \{p_1 \vee p_2 \vee p_3, p_1 \vee \neg p_3, \neg p_2 \vee \neg p_4\}$
- The AnsProlog program $\Pi(S)$ consists of the following:

$p_1 \leftarrow \textbf{not } n\_p_1.$
$n\_p_1 \leftarrow \textbf{not } p_1.$
$p_2 \leftarrow \textbf{not } n\_p_2.$
$n\_p_2 \leftarrow \textbf{not } p_2.$
$p_3 \leftarrow \textbf{not } n\_p_3.$
$n\_p_3 \leftarrow \textbf{not } p_3.$
$p_4 \leftarrow \textbf{not } n\_p_4.$
$n\_p_4 \leftarrow \textbf{not } p_4.$

$c_1 \leftarrow p_1.$
$c_1 \leftarrow p_2.$
$c_1 \leftarrow p_3.$
$\leftarrow \textbf{not } c_1.$

$c_2 \leftarrow p_1.$
$c_2 \leftarrow n\_p_3.$
$\leftarrow \textbf{not } c_2.$

$c_3 \leftarrow n\_p_2.$ $c_3 \leftarrow n\_p_4.$ $\leftarrow \textbf{not } c_3.$

- Models of S and answer sets of $\Pi(S)$ have a 1-1 correspondence.

Universal quantification: an example

- Consider an admission process in a college where admission is refused to an applicant if he has not taken any honors classes.
- An incorrect encoding

  $$\begin{aligned} refuse\_admission(X) \leftarrow\ & applicant(X), \\ & \textbf{not}\ taken\_honors\_class(X,Y). \end{aligned}$$

- A correct encoding

  $$has\_taken\_a\_honors\_class(X) \leftarrow taken\_honors\_class(X,Y).$$

  $$\begin{aligned} refuse\_admission(X) \leftarrow\ & applicant(X), \\ & \textbf{not}\ has\_taken\_a\_honors\_class(X). \end{aligned}$$

SOME LARGER PROGRAMS – N-QUEENS

The N-queens problem: Solution 1

1. Problem: Put n queens in a board of size $n \times n$ so that they do not attack each other, vertically, horizontally, or at 45 degrees.
2. Approach of solution 1: Place queens one by one with possibility space based on the squares.

   $at(I, X, Y)$ means queen I is in location (X, Y).
3. Declaration:

   $queen(1) \leftarrow . \quad \ldots \quad queen(n) \leftarrow .$
   $row(1) \leftarrow . \quad \ldots \quad row(n) \leftarrow .$
   $col(1) \leftarrow . \quad \ldots \quad col(n) \leftarrow .$
4. Enumeration

4.1. For each locations (X, Y) and each queen I, either I is in location (X, Y) or not.

$at(I, X, Y) \leftarrow queen(I), row(X), col(Y), \mathbf{not}\ not\_at(I, X, Y).$

$not\_at(I, X, Y) \leftarrow queen(I), row(X), col(Y), \mathbf{not}\ at(I, X, Y).$

4.2. For each queen I it is placed in at most one location.

$\leftarrow queen(I), row(X), col(Y), row(U), col(Z),$
$\quad at(I, X, Y), at(I, U, Z), Y \neq Z.$

$\leftarrow queen(I), row(X), col(Y), row(Z), col(V),$
$\quad at(I, X, Y), at(I, Z, V), X \neq Z.$

4.3. For each queen I it is placed in at least one location.

$placed(I) \leftarrow queen(I), row(X), col(Y), at(I, X, Y).$

$\leftarrow queen(I), \mathbf{not}\ placed(I).$

4.4. No two queens are placed in the same location.

$\leftarrow queen(I), row(X), col(Y), queen(J),$
$\quad at(I, X, Y), at(J, X, Y), I \neq J.$

5. Elimination:

5.1. No two distinct queens in the same row.

$$\leftarrow queen(I), row(X), col(Y), col(V), queen(J),$$
$$at(I, X, Y), at(J, X, V), I \neq J.$$

5.2. No two distinct queens in the same column.

$$\leftarrow queen(I), row(X), col(Y), row(U), queen(J),$$
$$at(I, X, Y), at(J, U, Y), I \neq J.$$

5.3. No two distinct queens attack each other diagonally.

$$\leftarrow row(X), col(Y), row(U), col(V),$$
$$queen(I), queen(J), at(I, X, Y), at(J, U, V),$$
$$I \neq J, abs(X - U) = abs(Y - V).$$

6. Note: Rule 4.4 is subsumed by both 5.1 and 5.2. In other words, in presence of 5.1 or 5.2 we do not need 4.4. It is needed in 4 if our only goal is enumeration.

The N-queens problem: Solution 2

1. Approach of solution 2: Place queens one by one in unique locations.
2. Declaration: As in Solution 1.
3. Enumeration:

$$\begin{aligned} other\_at(I,X,Y) \leftarrow\ & queen(I), row(X), col(Y), row(U), col(Z),\\ & at(I,U,Z), Y \neq Z.\\ other\_at(I,X,Y) \leftarrow\ & queen(I), row(X), col(Y), row(Z), col(V),\\ & at(I,Z,V), X \neq Z. \end{aligned}$$

$$at(I,X,Y) \leftarrow queen(I), row(X), col(Y), \textbf{not } other\_at(I,X,Y).$$

$$\begin{aligned} \leftarrow\ & queen(I), row(X), col(Y), queen(J),\\ & at(I,X,Y), at(J,X,Y), I \neq J. \end{aligned}$$

4. Elimination: As in Solution 1.

The N-queens problem: Solution 3

1. Approach of solution 3: Push two of the elimination constraints into the enumeration phase.
2. Declaration: As in Solution 1.
3. Enumeration:

$$
\begin{aligned}
other\_at(I,X,Y) \leftarrow\ & queen(I), row(X), col(Y), row(U), col(Z),\\
& at(I,U,Z), Y \neq Z.\\
other\_at(I,X,Y) \leftarrow\ & queen(I), row(X), col(Y), row(Z), col(V),\\
& at(I,Z,V), X \neq Z.
\end{aligned}
$$

$$at(I,X,Y) \leftarrow queen(I), row(X), col(Y), \textbf{not}\ other\_at(I,X,Y).$$

$$
\begin{aligned}
other\_at(I,X,Y) \leftarrow\ & queen(I), row(X), col(Y),\\
& col(V), queen(J), at(J,X,V), I \neq J.\\
other\_at(I,X,Y) \leftarrow\ & queen(I), row(X), col(Y), row(U),\\
& queen(J), at(J,U,Y), I \neq J.
\end{aligned}
$$

4. Elimination:

$$\begin{aligned}\leftarrow\ & row(X), col(Y), row(U), col(V), queen(I),\\ & \quad queen(J), at(I,X,Y), at(J,U,V), I \neq J,\\ & \quad abs(X-U) = abs(Y-V).\end{aligned}$$

The N-queens problem: Solution 4

1. Approach: Use predicate $in(X, Y)$ to mean that a queen is placed in position (X, Y).
2. Declarations:

   $row(1) \leftarrow . \quad \ldots \quad row(n) \leftarrow .$
   $col(1) \leftarrow . \quad \ldots \quad col(n) \leftarrow .$
3. Enumeration:
   - 3.1. A queen should not be placed in (X, Y) if there is a queen placed in the same row.

     $$not\_in(X, Y) \leftarrow row(X), col(Y), col(YY), \\ Y \neq YY, in(X, YY).$$

3.2. A queen should not be placed in (X, Y) if there is a queen placed in the same column.

$$not\_in(X,Y) \leftarrow row(X), col(Y), row(XX), \\ X \neq XX, in(XX,Y).$$

3.3. A queen must be placed in (X, Y) if it is not otherwise prevented.

$$in(X,Y) \leftarrow row(X), col(Y), \mathbf{not}\ not\_in(X,Y).$$

4. Elimination: We now need only one elimination constraint, the same one as before which prevents placements where queens can attack each other diagonally.

$$\leftarrow row(X), col(Y), row(XX), col(YY), \\ X \neq XX, Y \neq YY, in(X,Y), in(XX,YY), \\ abs(X - XX) = abs(Y - YY).$$

N-queens: a non-solution

- Pushing the last elimination rules to the enumeration phase:

$$
\begin{aligned}
not\_in(X,Y) \leftarrow\ & row(X), col(Y), row(XX), col(YY),\\
& X \neq XX, Y \neq YY, in(X,Y), in(XX,YY),\\
& abs(X-XX) = abs(Y-YY).
\end{aligned}
$$

- Resulting overall formulation:

$$row(1) \leftarrow . \qquad \ldots \qquad row(n) \leftarrow .$$
$$col(1) \leftarrow . \qquad \ldots \qquad col(n) \leftarrow .$$

$$
\begin{aligned}
not\_in(X,Y) \leftarrow\ & row(X), col(Y), col(YY), Y \neq YY, in(X,YY).\\
not\_in(X,Y) \leftarrow\ & row(X), col(Y), row(XX), X \neq XX, in(XX,Y).\\
not\_in(X,Y) \leftarrow\ & row(X), col(Y), row(XX), col(YY),\\
& X \neq XX, Y \neq YY, in(X,Y), in(XX,YY),\\
& abs(X-XX) = abs(Y-YY).\\
in(X,Y) \leftarrow\ & dim(X), dim(Y), \textbf{not}\ not\_in(X,Y).
\end{aligned}
$$

- The last formulation is *not correct* in the sense that it has answer sets where not all the n queens are placed.
- If $n = 4$, then one of the answer sets encode the following placement: $\{in(1,1), in(3,2), in(2,4)\}$, where only three queens are placed, instead of four.
- Why?
  - The above encoding places queens into a *valid* configuration, where a *valid* configuration is a configuration of queens that do not attack each other and to which additional queens can not be added without that queen attacking one or more of the already placed queens.
  - The configuration $\{in(1,1), in(3,2), in(2,4)\}$ is *valid* in that sense, as we can not add a new queen to it in such a way that it does not attack the already placed ones.

WHO LET THE ZEBRA OUT?

Problem statement

- There are five houses each of a different color (red, green, ivory, blue, and yellow) and inhabited by a person of a different nationality (Japanese, Englishman, Norwegian, Ukrainian, and Spaniard), with a different pet (horse, snails, zebra, fox, and dog), drink (water, coffee, tea, milk, and orange juice) and brand of cigarettes (Lucky strike, Winston, Chesterfields, Kools, and Parliaments).
- Constraints
  1. The Englishman lives in the red house.
  2. The Spaniard owns the dog.
  3. The Norwegian lives in the first house on the left.
  4. Kools are smoked in the yellow house.
  5. The man who smokes Chesterfields lives in the house next to the man with the fox.

6. The Norwegian lives next to the blue house.
7. The Winston smoker owns snails.
8. The Lucky Strike smoker drinks orange juice.
9. The Ukrainian drinks tea.
10. The Japanese smokes Parliaments.
11. Kools are smoked in the house next to the house where the horse is kept.
12. Coffee is drunk in the green house.
13. The green house is immediately to the right (your right) of the ivory house.
14. Milk is drunk in the middle house.

- The query: Who owns the zebra?

Solution 1 to the Zebra problem

- $together(U, V, W, X, Y, Z)$ meaning that the house number U is together with the cigarette brand V, country W, color X, pet Y, and drink Z. – NOT A GOOD WAY.
- Better representation 1: Use house number as the anchor, and have binary predicates *has_color*, *has_drink*, *has_animal*, *has_country*, and *has_smoke* where the first parameter is the house number and the second parameters are color, drink, pet, country, and smoke respectively.

- Domain specification:

$house(1) \leftarrow . \qquad \ldots \qquad house(5) \leftarrow .$

$right(X, X+1) \leftarrow house(X), house(X+1).$

$left(X+1, X) \leftarrow house(X), house(X+1).$

$next(X,Y) \leftarrow right(X,Y).$
$next(X,Y) \leftarrow left(X,Y).$

$color(red) \leftarrow . \qquad color(green) \leftarrow . \qquad color(blue) \leftarrow .$
$color(ivory) \leftarrow . \qquad color(yellow) \leftarrow .$

$country(norway) \leftarrow . \qquad country(japan) \leftarrow . \qquad country(uk) \leftarrow .$
$country(spain) \leftarrow . \qquad country(ukraine) \leftarrow .$

$smoke(parliaments) \leftarrow . \qquad smoke(lucky\_strike) \leftarrow .$
$smoke(kools) \leftarrow . \qquad smoke(chesterfield) \leftarrow .$
$smoke(winston) \leftarrow .$

$drink(coffee) \leftarrow . \qquad drink(tea) \leftarrow . \qquad drink(oj) \leftarrow .$
$drink(milk) \leftarrow . \qquad drink(water) \leftarrow .$

$animal(dog) \leftarrow .$ $animal(snails) \leftarrow .$ $animal(horse) \leftarrow .$
$animal(fox) \leftarrow .$ $animal(zebra) \leftarrow .$

- Enumeration of the predicates *has_color*, *has_drink*, *has_animal*, *has_country*, and *has_smoke*. For the predicate *has_color*, we make sure that every house has a unique color assigned to it, and every color corresponds to a unique house.

  $$\begin{aligned} other\_color(H,C) \leftarrow\ & house(H), color(C), color(CC), \\ & has\_color(H,CC), C \neq CC. \\ other\_color(H,C) \leftarrow\ & house(H), color(C), house(HH), \\ & has\_color(HH,C), H \neq HH. \\ has\_color(H,C) \leftarrow\ & house(H), color(C), \mathbf{not}\ other\_color(H,C). \end{aligned}$$

  The enumeration rules for the predicates *has_drink*, *has_animal*, *has_country*, and *has_smoke* are similar to that of the enumeration rules for *has_color* and is given below:

  $$\begin{aligned} other\_drink(H,C) \leftarrow\ & house(H), drink(C), drink(CC), \\ & has\_drink(H,CC), C \neq CC. \end{aligned}$$

$$other\_drink(H,C) \leftarrow house(H), drink(C), house(HH),$$
$$has\_drink(HH,C), H \neq HH.$$
$$has\_drink(H,C) \leftarrow house(H), drink(C), \textbf{not } other\_drink(H,C).$$
$$other\_animal(H,C) \leftarrow house(H), animal(C), animal(CC),$$
$$has\_animal(H,CC), C \neq CC.$$
$$other\_animal(H,C) \leftarrow house(H), animal(C), house(HH),$$
$$has\_animal(HH,C), H \neq HH.$$
$$has\_animal(H,C) \leftarrow house(H), animal(C),$$
$$\textbf{not } other\_animal(H,C).$$
$$other\_country(H,C) \leftarrow house(H), country(C), country(CC),$$
$$has\_country(H,CC), C \neq CC.$$
$$other\_country(H,C) \leftarrow house(H), country(C), house(HH),$$
$$has\_country(HH,C), H \neq HH.$$
$$has\_country(H,C) \leftarrow house(H), country(C),$$
$$\textbf{not } other\_country(H,C).$$
$$other\_smoke(H,C) \leftarrow house(H), smoke(C), smoke(CC),$$
$$has\_smoke(H,CC), C \neq CC.$$

$$\begin{aligned} other\_smoke(H,C) \leftarrow\ & house(H), smoke(C), house(HH), \\ & has\_smoke(HH,C), H \neq HH. \end{aligned}$$

$$has\_smoke(H,C) \leftarrow house(H), smoke(C), \mathbf{not}\ other\_smoke(H,C).$$

- The elimination constraints:

$$s_1 \leftarrow house(H), has\_country(H,uk), has\_color(H,red).$$
$$s_2 \leftarrow house(H), has\_animal(H,dog), has\_country(H,spain).$$
$$s_3 \leftarrow has\_country(1,norway).$$
$$s_4 \leftarrow house(H), has\_smoke(H,kools), has\_color(H,yellow).$$
$$\begin{aligned} s_5 \leftarrow\ & house(H), house(HH), has\_smoke(H,chesterfield), \\ & has\_animal(HH,fox), next(H,HH). \end{aligned}$$
$$\begin{aligned} s_6 \leftarrow\ & house(H), house(HH), has\_color(H,blue), \\ & has\_country(HH,norway), next(H,HH). \end{aligned}$$
$$s_7 \leftarrow house(H), has\_smoke(H,winston), has\_animal(H,snails).$$
$$s_8 \leftarrow house(H), has\_smoke(H,lucky\_strike), has\_drink(H,oj).$$
$$s_9 \leftarrow house(H), has\_country(H,ukraine), has\_drink(H,tea).$$
$$\begin{aligned} s_{10} \leftarrow\ & house(H), has\_smoke(H,parliaments), \\ & has\_country(H,japan). \end{aligned}$$

$$s_{11} \leftarrow house(H), house(HH), has\_smoke(H, kools),$$
$$has\_animal(HH, horse), next(H, HH).$$
$$s_{12} \leftarrow house(H), has\_color(H, green), has\_drink(H, coffee).$$
$$s_{13} \leftarrow house(H), house(HH), has\_color(HH, ivory),$$
$$has\_color(H, green), right(HH, H).$$
$$s_{14} \leftarrow has\_drink(3, milk).$$

The following rule encodes that $satisfied$ is true iff all the 14 constraints are individually satisfied.

$$satisfied \leftarrow s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}, s_{12}, s_{13}, s_{14}.$$

The following rule eliminates possibilities were $satsified$ is not true. I.e., when one of the 14 conditions does not hold.

$$\leftarrow \textbf{not } satisfied.$$

Solution 2 to the Zebra problem

- Representation idea: Use a single association predicate, and use two new predicates *object* and *same_type* to distinguish the association of a house number with objects of different types.
- Main advantage: It makes it easier to express the observations as we do not have to remember the particular association predicate for the object being described.

- Besides the domain specification in Solution 1 we have the following additional rules.

$object(X) \leftarrow color(X).$
$object(X) \leftarrow drink(X).$
$object(X) \leftarrow animal(X).$
$object(X) \leftarrow country(X).$
$object(X) \leftarrow smoke(X).$

$same\_type(X,Y) \leftarrow color(X), color(Y).$
$same\_type(X,Y) \leftarrow drink(X), drink(Y).$
$same\_type(X,Y) \leftarrow animal(X), animal(Y).$
$same\_type(X,Y) \leftarrow country(X), country(Y).$
$same\_type(X,Y) \leftarrow smoke(X), smoke(Y).$

- Enumeration: $has(X,Y)$ intuitively means that the house number X is associated with object Y.

$$\begin{aligned} other\_has(X,Y) \leftarrow\ & house(X), object(Y), house(Z), \\ & Z \neq X, has(Z,Y). \end{aligned}$$

$$other\_has(X,Y) \leftarrow house(X), object(Y), object(Z),$$
$$Z \neq Y, same\_type(Y,Z), has(X,Z).$$
$$has(X,Y) \leftarrow house(X), object(Y), \mathbf{not}\ other\_has(X,Y).$$

- Elimination Constraints:

$$s_1 \leftarrow house(H), has(H, uk), has(H, red).$$
$$s_2 \leftarrow house(H), has(H, dog), has(H, spain).$$
$$s_3 \leftarrow has(1, norway).$$
$$s_4 \leftarrow house(H), has(H, kools), has(H, yellow).$$
$$s_5 \leftarrow house(H), house(HH), has(H, chesterfield),$$
$$has(HH, fox), next(H, HH).$$
$$s_6 \leftarrow house(H), house(HH), has(H, blue),$$
$$has(HH, norway), next(H, HH).$$
$$s_7 \leftarrow house(H), has(H, winston), has(H, snails).$$
$$s_8 \leftarrow house(H), has(H, lucky\_strike), has(H, oj).$$
$$s_9 \leftarrow house(H), has(H, ukraine), has(H, tea).$$
$$s_{10} \leftarrow house(H), has(H, parliaments), has(H, japan).$$

$s_{11} \leftarrow house(H), house(HH), has(H, kools),$
$\quad has(HH, horse), next(H, HH).$

$s_{12} \leftarrow house(H), has(H, green), has(H, coffee).$

$s_{13} \leftarrow house(H), house(HH), has(H, green),$
$\quad has(HH, ivory), right(HH, H).$

$s_{14} \leftarrow has(3, milk).$

In addition as before we have

$satisfied \leftarrow s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}, s_{12}, s_{13}, s_{14}.$

$\leftarrow$ **not** $satisfied.$

PLANNING AND PLAN VERIFICATION

Simple Blocks world: problem description

1. The goal description:
   $\{on(a,c), on(c,b)\}$
2. The domain description
   - 2.1. Sort: $block(X)$. In the rest of the descriptions X and Y range over the sort $block$.
   - 2.2. fluents: $on(X), ontable(X), clear(X), handempty, holding(X)$.
   - 2.3. Executability Conditions:
     executable $pick\_up(X)$ **if** $clear(X), ontable(X), handempty$.
     executable $put\_down(X)$ **if** $holding(X)$.
     executable $stack(X,Y)$ **if** $holding(X), clear(Y)$.
     executable $unstack(X,Y)$ **if** $on(X,Y), clear(X), handempty$.

2.4. Effect propositions:

$pick\_up(X)$ **causes** $\neg ontable(X), \neg clear(X),$
$\neg handempty, holding(X).$

$put\_down(X)$ **causes** $\neg holding(X), clear(X), handempty,$
$ontable(X).$

$stack(X, Y)$ **causes** $\neg holding(X), \neg clear(Y),$
$clear(X), handempty, on(X, Y).$

$unstack(X, Y)$ **causes** $holding(X), clear(Y), \neg clear(X),$
$\neg handempty, \neg on(X, Y).$

3. The initial state description:

initially $clear(c)$
initially $clear(a)$
initially $ontable(c)$
initially $ontable(b)$
initially $on(a, b)$
initially $handempty$

Simple Blocks World: AnsProlog encoding

1. The Domain dependent part $\pi^{dep}_{strips1}(D_{bw}, O_{bw}, G_{bw})$:

   1.1. The domain $\pi^{dep.dom}_{strips1}(D_{bw})$:

   $block(a).$
   $block(b).$
   $block(c).$
   $block(d).$
   $fluent(on(X, Y)) \leftarrow block(X), block(Y).$
   $fluent(ontable(X)) \leftarrow block(X).$
   $fluent(clear(X)) \leftarrow block(X).$
   $fluent(holding(X)) \leftarrow block(X).$
   $fluent(handempty).$

$action(pick\_up(X)) \leftarrow block(X).$
$action(put\_down(X)) \leftarrow block(X).$
$action(stack(X, Y)) \leftarrow block(X), block(Y).$
$action(unstack(X, Y)) \leftarrow block(X), block(Y).$

1.2. The executability conditions $\pi^{dep.exec}_{strips1}(D_{bw})$:

$exec(pick\_up(X), clear(X))).$
$exec(pick\_up(X), ontable(X)).$
$exec(pick\_up(X), handempty).$
$exec(put\_down(X), holding(X)).$
$exec(stack(X, Y), holding(X)).$
$exec(stack(X, Y), clear(Y)).$
$exec(unstack(X, Y), clear(X)).$
$exec(unstack(X, Y), on(X, Y)).$
$exec(unstack(X, Y), handempty).$

1.3. The dynamic causal laws: $\pi^{dep.dyn}_{strips1}(D_{bw})$:

$causes(pick\_up(X), neg(ontable(X))).$
$causes(pick\_up(X), neg(clear(X))).$

$causes(pick\_up(X), holding(X)).$
$causes(pick\_up(X), neg(handempty)).$
$causes(put\_down(X), ontable(X)).$
$causes(put\_down(X), clear(X)).$
$causes(put\_down(X), neg(holding(X))).$
$causes(put\_down(X), handempty).$
$causes(stack(X, Y), neg(holding(X))).$
$causes(stack(X, Y), neg(clear(Y))).$
$causes(stack(X, Y), clear(X)).$
$causes(stack(X, Y), handempty).$
$causes(stack(X, Y), on(X, Y)).$
$causes(unstack(X, Y), holding(X)).$
$causes(unstack(X, Y), clear(Y)).$
$causes(unstack(X, Y), neg(clear(X))).$
$causes(unstack(X, Y), neg(handempty)).$
$causes(unstack(X, Y), neg(on(X, Y))).$

1.4. The initial state $\pi^{dep.init}_{strips1}(O_{bw.1})$:

$initially(handempty).$
$initially(clear(a)).$
$initially(clear(c)).$
$initially(ontable(c)).$
$initially(ontable(b)).$
$initially(on(a,b)).$

1.5. The goal conditions $\pi^{dep.goal}_{strips1}(G_{bw.1})$:

$finally(on(a,c)).$
$finally(on(c,b)).$

2. The domain independent part $\pi^{indep}_{strips1}$:

2.1. Defining time:

$time(1). \qquad \ldots \qquad time(length).$

2.2. Defining $goal(T)$:

$not\_goal(T) \leftarrow time(T), finally(X), \textbf{not } holds(X,T).$
$goal(T) \leftarrow time(T), \textbf{not } not\_goal(T).$

2.3. Eliminating possible answer sets which do not have a plan of the given length:

$\leftarrow \textbf{not } goal(length).$

2.4. Defining contrary:

$contrary(F, neg(F)).$

$contrary(neg(F), F).$

2.5. Defining executability:

$not\_executable(A, T) \leftarrow exec(A, F), \textbf{not } holds(F, T).$

$executable(A, T) \leftarrow T < length, \textbf{not } not\_executable(A, T).$

2.6. Fluent values at time-point 1:

$holds(F, 1) \leftarrow initially(F).$

2.7. Effect axiom:

$$holds(F, T + 1) \leftarrow T < length, executable(A, T), occurs(A, T), causes(A, F).$$

2.8. Inertia:

$$holds(F, T+1) \leftarrow contrary(F, G), T < length, holds(F, T), \\ \textbf{not } holds(G, T+1).$$

2.9. Enumerating action occurrences:

$$occurs(A, T) \leftarrow action(A), time(T), \textbf{not } other\_occurs(A, T).$$

$$other\_occurs(A, T) \leftarrow action(A), action(AA), time(T), \\ occurs(AA, T), A \neq AA.$$

$$\leftarrow action(A), time(T), occurs(A, T), \textbf{not } executable(A, T).$$

Using temporal constraints to guide the search

1. Defining bad_tower_below

$$
\begin{aligned}
holds(bad\_tower\_below(X), T) &\leftarrow holds(ontable(X), T),\\
&\quad finally(on(X, Y)).\\
holds(bad\_tower\_below(X), T) &\leftarrow holds(on(X, Y), T),\\
&\quad finally(ontable(X)).\\
holds(bad\_tower\_below(X), T) &\leftarrow holds(on(X, Y), T),\\
&\quad finally(holding(Y)).\\
holds(bad\_tower\_below(X), T) &\leftarrow holds(on(X, Y), T),\\
&\quad finally(clear(Y)).\\
holds(bad\_tower\_below(X), T) &\leftarrow holds(on(X, Y), T),\\
&\quad finally(on(X, Z)), Z \neq Y.\\
holds(bad\_tower\_below(X), T) &\leftarrow holds(on(X, Y), T),\\
&\quad finally(on(Z, Y)), Z \neq X.
\end{aligned}
$$

$$holds(bad\_tower\_below(X), T) \leftarrow holds(on(X,Y),T),\ holds(bad\_tower\_below(Y),T).$$

2. Defining *goodtower*

$$holds(goodtower(X),T) \leftarrow holds(clear(X),T),\ \textbf{not}\ finally(holding(X)),\ \textbf{not}\ holds(bad\_tower\_below(X),T).$$

3. Defining *badtower*

$$holds(badtower(X),T) \leftarrow holds(clear(X),T),\ \textbf{not}\ holds(goodtower(X),T).$$

4. The temporal constraints.

$$\leftarrow holds(goodtower(X),T), holds(on(Y,X),T+1),\ \textbf{not}\ holds(goodtower(Y),T+1).$$

$$\leftarrow holds(badtower(X),T), holds(on(Y,X),T+1).$$

$$\begin{aligned}\leftarrow\ & holds(ontable(X,T)), finally(on(X,Y)),\\ & \quad \textbf{not}\ holds(goodtower(Y),T),\\ & \quad holds(holding(X),T+1).\end{aligned}$$

The temporal conditions in (4) above can be replaced by the following direct constraints on action occurrences. This further cuts down on the planning time.

$$\begin{aligned}\leftarrow\ & holds(goodtower(X),T), occurs(stack(Y,X),T),\\ & \quad \textbf{not}\ holds(goodtower(Y),T+1).\\ \leftarrow\ & holds(badtower(X),T), occurs(stack(Y,X),T).\\ \leftarrow\ & holds(ontable(X,T)), finally(on(X,Y)),\\ & \quad \textbf{not}\ holds(goodtower(Y),T), occurs(pick\_up(X),T).\end{aligned}$$

Blocks world using a richer formalism: the description

1. Sort: $block(X)$
2. Basic fluents: $on(X), ontable(X), holding(X)$.
3. Defined fluents: $clear(X), handempty$.
4. Definition of defined fluents.
   - $clear(X)$ iff $\not\exists Y : on(Y, X)$
   - $handempty$ iff $\not\exists Y : holding(Y)$
5. Executability Conditions:

   executable $pick\_up(X)$ **if** $clear(X), ontable(X), handempty$
   executable $put\_down(X)$ **if** $holding(X)$
   executable $stack(X, Y)$ **if** $holding(X), ontable(Y)$
   executable $stack(X, Y)$ **if** $holding(X), on(Y, Z)$
   executable $unstack(X, Y)$ **if** $clear(X), on(X, Y), handempty$

6. Effect propositions:

   $pick\_up(X)$ **causes** $holding(X)$
   $put\_down(X)$ **causes** $ontable(X)$
   $stack(X,Y)$ **causes** $on(X,Y)$
   $unstack(X,Y)$ **causes** $holding(X)$

7. Causal ramification constraints:

   $on(X,Y)$ **causes** $\neg holding(X)$
   $holding(X)$ **causes** $\neg on(X,Y)$
   $ontable(X)$ **causes** $\neg holding(X)$
   $holding(X)$ **causes** $\neg ontable(X)$

8. Causal qualification constraint:

   $on(X,Y), on(Z,Y), X \neq Z$ **causes** $false$

AnsProlog encoding of the BWP in the richer formalism

1. The domain dependent part $\pi_{adl1}^{dep}(D_{bw1}, O_{bw1}, G_{bw1})$:

    1.1. The domain: $\pi_{adl1}^{dep.dom}(D_{bw1})$:
    $fluent(on(X,Y)).$
    $fluent(ontable(X)).$
    $fluent(holding(X)).$
    $defined\_fluent(clear(X)).$
    $defined\_fluent(handempty).$

1.2. The executability conditions $\pi_{adl1}^{dep.exec}(D_{bw1})$:

$$
\begin{array}{ll}
executable(pick\_up(X), T) \leftarrow T < length, & \\
\qquad holds(clear(X), T), holds(ontable(X), T), & \\
\qquad holds(handempty, T). & \\
executable(put\_down(X), T) \leftarrow T < length, & \\
\qquad holds(holding(X), T). & \\
executable(stack(X, Y), T) \leftarrow T < length, & \\
\qquad holds(holding(X), T), holds(ontable(Y), T). & \\
executable(stack(X, Y), T) \leftarrow T < length, & \\
\qquad holds(holding(X), T), holds(on(Y, Z), T). & \\
executable(unstack(X, Y), T) \leftarrow T < length, & \\
\qquad holds(clear(X), T), holds(on(X, Y), T), & \\
\qquad holds(handempty, T). &
\end{array}
$$

1.3. The dynamic causal laws $\pi_{adl1}^{dep.dyn}(D_{bw1})$: The use of defined fluents and causal ramification constraints drastically cuts down on the number of dynamic causal laws. Instead of 18 of them, we now have only 4 of them.

$causes(pick\_up(X), holding(X)).$
$causes(put\_down(X), ontable(X)).$
$causes(stack(X,Y), on(X,Y)).$
$causes(unstack(X,Y), holding(X)).$

1.4. Defined fluents $\pi^{dep.def}_{adl1}(D_{bw1})$:

$holds(neg(clear(X)), T) \leftarrow holds(holding(X), T).$
$holds(neg(clear(X)), T) \leftarrow holds(on(Y,X), T).$
$holds(clear(X), T) \leftarrow holds(ontable(X), T),$
$\qquad \textbf{not}\ holds(neg(clear(X)), T).$
$holds(clear(X), T) \leftarrow holds(on(X,Y), T),$
$\qquad \textbf{not}\ holds(neg(clear(X)), T).$
$holds(neg(handempty), T) \leftarrow holds(holding(X), T).$
$holds(handempty, T) \leftarrow \textbf{not}\ holds(neg(handempty), T).$

1.5. Qualification constraints $\pi^{dep.qual}_{adl1}(D_{bw1})$

$\leftarrow holds(on(X,Y), T), holds(on(Z,Y), T), neq(X,Z).$

1.6. Static ramification constraints $\pi^{dep.ram}_{adl1}(D_{bw1})$

$static\_causes(on(X,Y), neg(holding(X))).$

$static\_causes(holding(X), neg(on(X, Y))).$
$static\_causes(ontable(X), neg(holding(X))).$
$static\_causes(holding(X), neg(ontable(X))).$

1.7. The initial state $\pi_{adl1}^{dep.init}(O_{bw1})$: As before.

1.8. The goal state $\pi_{adl1}^{dep.goal}(G_{bw1})$: As before.

2. The domain independent part π_{adl1}^{indep}.

2.1. Defining *time*: as before.

2.2. Defining $goal(T)$: as before.

2.3. Defining plan existence: as before.

2.4. Defining contrary: as before.

2.5. Fluent values at time point 1: as before.

2.6. Defining literal:
$literal(G) \leftarrow fluent(G).$
$literal(neg(G)) \leftarrow fluent(G).$

2.7. Effect axioms: Here the effect axioms only define the effects of actions on literals made up of basic fluents.

$$holds(F, T+1) \leftarrow literal(F), T < length, \\ executable(A, T), occurs(A, T), causes(A, F).$$

2.8. Inertia: Here the inertia axioms are also defined only with respect to literals made up of basic fluents.

$$holds(F, T+1) \leftarrow literal(F), contrary(F, G), \\ T < length, holds(F, T), \textbf{not}\ holds(G, T+1).$$

2.9. Effect axiom for static causal laws:

$$holds(F, T) \leftarrow T < length, holds(G, T), static\_causes(G, F).$$

2.10. Occurrences of actions: as before.

CONCLUSION

AnsProlog*

- AnsProlog*: Adding disjunction, classical negation, empty heads to AnsProlog.
- AnsProlog and AnsProlog* are non-monotonic.
- AnsProlog and AnsProlog* are fully declarative.
- AnsProlog* is as expressive as Default logic but much simpler. (First-order logic can not express transitive closure, or aggregates.)
- Largest collection of building block results about AnsProlog* that help build large programs from small ones and analyze big programs by partitioning them to modules.
- Several efficient implementations. (Smodels, Dlv, others).
- Has been used in many applications.