Online appendix for the paper

ASPeRiX, a First Order Forward Chaining Approach for Answer Set Computing

published in Theory and Practice of Logic Programming

Claire Lefèvre

LERIA, University of Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
claire@info.univ-angers.fr

Christopher Béatrix

LERIA, University of Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
beatrix@info.univ-angers.fr

Igor Stéphan

LERIA, University of Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
stephan@info.univ-angers.fr

Laurent Garcia

LERIA, University of Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
garcia}@info.univ-angers.fr

submitted 24 March 2014; revised 24 November 2014; accepted 25 February 2015

Appendix A Hanoi example

The following ASP program is the *Hanoi* example with 4 discs.

```
%------ Initial settings
number_of_moves(10000).
largest_disc(4).

%------ Initial state
initial_state(towers(l(4,l(3,l(2,l(1,nil)))),nil,nil)).

% ------ Goal state
goal(towers(nil, nil, l(4,l(3,l(2,l(1,nil)))))).

% ------ all discs involved ------
disc(1..4).
```

```
% ------ legal stacks ------
legalStack(nil).
legalStack(l(T,nil)) :- disc(T).
legalStack(l(T,l(T1,S))) :- legalStack(l(T1,S)), disc(T), T > T1.

% ------ possible moves ------
possible_state(0,towers(S1,S2,S3))
      :- initial_state(towers(S1,S2,S3)),
      legalStack(S1), legalStack(S2), legalStack(S3).
possible_state(I,towers(S1,S2,S3))
      :- possible_move(I,T,towers(S1,S2,S3)).

% From stack one to stack two.
possible_move(J,towers(l(X,S1),S2,S3),towers(S1,l(X,S2),S3))
      :- possible_state(I,towers(l(X,S1),S2,S3)),
      number_of_moves(N), I<=N, legalStack(l(X,S2)), J=I+1, not ok(I).

% From stack one to stack three.
possible_move(J,towers(l(X,S1),S2,S3),towers(S1,S2,l(X,S3)))
      :- possible_state(I,towers(l(X,S1),S2,S3)),
      number_of_moves(N), I<=N, legalStack(l(X,S3)), J=I+1, not ok(I).

% From stack two to stack one.
possible_move(J,towers(S1,l(X,S2),S3),towers(l(X,S1),S2,S3))
      :- possible_state(I,towers(S1,l(X,S2),S3)),
      number_of_moves(N), I<=N, legalStack(l(X,S1)), J=I+1, not ok(I).

% From stack two to stack three.
possible_move(J,towers(S1,l(X,S2),S3),towers(S1,S2,l(X,S3)))
      :- possible_state(I,towers(S1,l(X,S2),S3)),
      number_of_moves(N), I<=N, legalStack(l(X,S3)), J=I+1, not ok(I).

% From stack three to stack one.
possible_move(J,towers(S1,S2,l(X,S3)),towers(l(X,S1),S2,S3))
      :- possible_state(I,towers(S1,S2,l(X,S3))),
      number_of_moves(N), I<=N, legalStack(l(X,S1)), J=I+1, not ok(I).

% From stack three to stack two.
possible_move(J,towers(S1,S2,l(X,S3)),towers(S1,l(X,S2),S3))
      :- possible_state(I,towers(S1,S2,l(X,S3))),
      number_of_moves(N), I<=N, legalStack(l(X,S2)), J=I+1, not ok(I).

%------ actual moves ------
% a solution exists if and only if there is a "possible_move"
% leading to the goal.
```

```
% in this case, starting from the goal, we proceed backward
% to the initial state to single out the full set of moves.

% Choose from the possible moves.
move(I,towers(S1,S2,S3))
     :- goal(towers(S1,S2,S3)), possible_state(I,towers(S1,S2,S3)).
ok(I) :- move(I,towers(S1,S2,S3)), goal(towers(S1,S2,S3)).
win :- ok(I).
:- not win.

move(J,towers(S1,S2,S3))
     :- move(I,towers(A1,A2,A3)),
     possible_move(I,towers(S1,S2,S3),towers(A1,A2,A3)), J=I-1,
     not nomove(J,towers(S1,S2,S3)).

nomove(J,towers(S1,S2,S3))
     :- move(I,towers(A1,A2,A3)),
     possible_move(I,towers(S1,S2,S3),towers(A1,A2,A3)), J=I-1,
     not move(J,towers(S1,S2,S3)).

%------ precisely one move at each step ------
moveStepI(I) :- move(I,T).

:- legalMoveNumber(I), ok(J), I<J, not moveStepI(I).

:- legalMoveNumber(I), move(I,T1), move(I,T2), T1!=T2.

legalMoveNumber(0).

legalMoveNumber(K)
     :- legalMoveNumber(I), number_of_moves(J), I < J, K=I+1.

#hide.
#show move/2.
```

Appendix B Proofs

B.1 Proof of Theorem 2

We first give some material needed in the proof. Auxiliary Lemma 1 is used in the proof of Lemma 2. Lemmas 2 and 3 establish completeness and correctness.

Lemma 1 shows that the generating rules of a program can be ordered so as to correspond to the order of application of rules in an `ASPeRiX` computation. Condition (1) says that a rule used at step i is supported at this step. Condition (2) says that if a rule is a member of Δ_{pro} at step i but is used at a later stage j,

then all rules used at steps between i and j are members of Δ_{pro} at step i. In other words, condition (2) says that propagation is entirely completed before making a choice.

Lemma 1. Let P be a normal logic program and X be an answer set of P. Then, there exists an enumeration $\langle r_i \rangle_{i \in [1..n]}$ of $GR_P(X)$, the set of generating rules of X, such that for all $i \in [1..n]$ the following two conditions are satisfied:

(1) $body^+(r_i) \subseteq head(\{r_k \mid k < i\})$
(2) for all $j > i$, if $body^+(r_j) \subseteq head(\{r_k \mid k < i\})$ and $body^-(r_j) \subseteq body^-(\{r_k \mid k < i\})$ then $body^-(r_i) \subseteq body^-(\{r_k \mid k < i\})$.

Proof. (of Lemma 1) Let P be a normal logic program and X be an answer set of P. By a theorem from (Konczak et al. 2006), there exists an enumeration $\langle r_i \rangle_{i \in [1..n]}$ of $GR_P(X)$ such that $\forall i \in [1..n]$, $body^+(r_i) \subseteq head(\{r_k \mid k < i\})$, i.e. such that condition (1) is satisfied. This enumeration can be recursively modified in the following way in order to verify condition (2). For each $i \in [1..n]$, if r_i satisfies (2) then r_i remains at rank i, else there exists r_j with $j > i$ that falsifies condition (2). In this last case, it suffices to swap the two rules in the enumeration to satisfy condition (2) at rank i. □

Notation. If P is a normal logic program and $\langle R_i, \langle IN_i, OUT_i \rangle \rangle_{i=0}^{\infty}$ is a sequence of ground rule sets R_i and partial interpretations $\langle IN_i, OUT_i \rangle$, then Δ^i_{pro} denotes $\Delta_{pro}(P, \langle IN_i, OUT_i \rangle, R_i)$ and Δ^i_{cho} denotes $\Delta_{cho}(P, \langle IN_i, OUT_i \rangle, R_i)$.

Lemma 2. Let P be a normal logic program and X be an answer set of P. Then there exists an `ASPeRiX` computation that converges to X.

Proof. (of Lemma 2) Let P be a normal logic program and X be an answer set of P. Then, there exists an enumeration $\langle r_i \rangle_{i \in [1..n]}$ of $GR_P(X)$ that satisfies conditions (1) and (2) from Lemma 1.

Let $\langle R_i, \langle IN_i, OUT_i \rangle \rangle_{i=0}^{\infty}$ be the sequence defined as follows.

- $R_0 = \emptyset$, $IN_0 = \emptyset$ and $OUT_0 = \{\bot\}$
- $\forall i, 1 \leq i \leq n$, $R_i = R_{i-1} \cup \{r_i\}$, $IN_i = IN_{i-1} \cup \{head(r_i)\}$ and $OUT_i = OUT_{i-1} \cup body^-(r_i)$
- $\forall i > n$, $R_i = R_{i-1}$, $IN_i = IN_{i-1}$ and $OUT_i = OUT_{i-1}$

For all $i \in [1..n]$, we have:

(*1) $X = head(GR_P(X))$ (by Theorem 1)
(*2) $IN_i = \bigcup_{j=1}^{i} \{head(r_j)\}$ and $IN_\infty = \bigcup_{i=0}^{\infty} IN_i = X$ (by (*1))
(*3) $OUT_i = \bigcup_{j=1}^{i} body^-(r_j)$ and therefore $OUT_i \cap X = \emptyset$ (by Definition 2 of $GR_P(X)$)
(*4) $\Delta_{pro}(P, \langle IN_i, OUT_i \rangle, R_i) \subseteq GR_P(X)$

Property (*4) can be proved as follows. By definition 5, $\Delta^i_{pro} = \{r \in ground(P) \setminus R_i \mid body^+(r) \subseteq IN_i \text{ and } body^-(r) \subseteq OUT_i\}$. And by (*2) and (*3), $IN_i \subseteq X$ and $OUT_i \cap X = \emptyset$. Thus $\Delta^i_{pro} \subseteq GR_P(X)$.

We are now able to prove that the sequence $\langle R_i, \langle IN_i, OUT_i\rangle\rangle_{i=0}^{\infty}$ is an ASPeRiX computation.

Let us first note that $\forall i,\ \langle IN_i, OUT_i\rangle$ is a partial interpretation since $IN_i \cap OUT_i = \emptyset$ (by (*2) and (*3)).

Now we prove that Revision principle holds for each $i \geq 1$. Let i such that $1 \leq i \leq n$, then r_i is such that $body^+(r_i) \subseteq head(\{r_k \mid k < i\}) = IN_{i-1}$. Two cases are possible. First, if $body^-(r_i) \subseteq body^-(\{r_k \mid k < i\}) = OUT_{i-1}$, then $r_i \in \Delta_{pro}^{i-1}$ and Revision principle holds at rank i. Second, if $body^-(r_i) \not\subseteq body^-(\{r_k \mid k < i\})$ then, by definition of enumeration $\langle r_i\rangle_{i\in[1..n]}$, there is no rule r_j with $j > i$ such that $body^+(r_j) \subseteq IN_{i-1}$ and $body^-(r_j) \subseteq OUT_{i-1}$. So $\Delta_{pro}^{i-1} \cap GR_P(X) = \emptyset$. And as $\Delta_{pro}^{i-1} \subseteq GR_P(X)$ (by (*4)), $\Delta_{pro}^{i-1} = \emptyset$. Moreover, r_i is a generating rule, thus $body^-(r_i) \cap X = \emptyset$ and $body^-(r_i) \cap IN_{i-1} = \emptyset$ (since $IN_{i-1} \subseteq X$). Thereby $r_i \in \Delta_{cho}^{i-1}$ and Revision principle holds. If $i > n$, Revision principle trivially holds (Stability).

At step n, we have $IN_n = \bigcup_{j=1}^{n}\{head(r_j)\} = X$ and $R_n = \bigcup_{j=1}^{n}\{r_j\} = GR_P(X)$. $\Delta_{cho}^{n+1} = \{r \in ground(P) \setminus R_n \mid body^+(r) \subseteq X \text{ and } body^-(r) \cap X = \emptyset\}$. Thus $\Delta_{cho}^{n+1} = \emptyset$. Convergence principle holds and $IN_\infty = IN_n = X$. □

Lemma 3. Let P be a normal logic program and $\langle R_i, \langle IN_i, OUT_i\rangle\rangle_{i=0}^{\infty}$ be an ASPeRiX computation for P. Then, IN_∞ is an answer set of P.

Proof. (of Lemma 3) Let $\langle R_i, \langle IN_i, OUT_i\rangle\rangle_{i=0}^{\infty}$ be an ASPeRiX computation for P.

We first prove that $\forall i > 0,\ \forall j \geq i-1,\ R_i \subseteq GR_P(IN_j)$. For each rule r_i, $body^+(r_i) \subseteq IN_{i-1}$ and IN set increases monotonically, thus $body^+(r_i) \subseteq IN_j, \forall j \geq i-1$. If $r_i \in \Delta_{pro}^{i-1}$, then $body^-(r_i) \subseteq OUT_{i-1}$ and $OUT_{i-1} \cap IN_{i-1} = \emptyset$. Since IN and OUT sets grow monotonically with an empty intersection, $body^-(r_i) \cap IN_j = \emptyset, \forall j \geq i-1$. If $r_i \in \Delta_{cho}^{i-1}$, then $body^-(r_i) \cap IN_{i-1} = \emptyset$. And, since $OUT_i = OUT_{i-1} \cup body^-(r_i)$, we have $\forall j \geq i, body^-(r_i) \subseteq OUT_j$, and thus, with the same reasonning as above ($r_i \in \Delta_{pro}^{i-1}$), $body^-(r_i) \cap IN_j = \emptyset, \forall j \geq i-1$.

$R_i = \bigcup_{k=1}^{i}\{r_k\}$ and, since $\forall j \geq k-1,\ r_k \in GR_P(IN_j)$, $r_k \in GR_P(IN_i)$. Thus $R_i \subseteq GR_P(IN_i)$.

By Convergence principle we have $\exists i,\ \Delta_{cho}^{i} = \{r \in ground(P) \setminus R_i \mid body^+(r) \subseteq IN_i \text{ and } body^-(r) \cap IN_i = \emptyset\} = \emptyset$, then $GR_P(IN_i) \subseteq R_i$. Since $\forall i, R_i \subseteq GR_P(IN_i)$, $GR_P(IN_i) = R_i$. And $IN_i = head(R_i)$ (by definition of an ASPeRiX computation), thus $IN_i = head(GR_P(IN_i))$ and IN_i is an answer set of P (by Theorem 1). □

Proof. (of Theorem 2) Lemmas 2 and 3 prove each one direction of the equivalence. □

B.2 Proof of Theorem 3

Lemmas 4 and 5 establish completeness and correctness.

Lemma 4. Let P be a normal logic program and X be an answer set for P. Then there exists a mbt ASPeRiX computation for P that converges to X.

Proof. (of Lemma 4) Let P be a normal logic program and X an answer set for P. By Theorem 2, there exists an ASPeRiX computation $\langle R_i, \langle IN_i, OUT_i\rangle\rangle_{i=0}^{\infty}$ with

$IN_\infty = X$. Let $C = \langle K_i, R_i, \langle IN_i, MBT_i, OUT_i\rangle\rangle_{i=0}^{\infty}$ with $K_i = MBT_i = \emptyset$, $\forall i \geq 0$. C is clearly a mbt `ASPeRiX` computation for P where "Rule exclusion" is never used and thus "Mbt-propagation" is not used either. □

Lemma 5. Let P be a normal logic program and $\langle K_i, R_i, \langle IN_i, MBT_i, OUT_i\rangle\rangle_{i=0}^{\infty}$ be a mbt `ASPeRiX` computation for P. Then IN_∞ is an answer set of P.

Proof. (of Lemma 5) Let $\langle K_i, R_i, \langle IN_i, MBT_i, OUT_i\rangle\rangle_{i=0}^{\infty}$ a mbt `ASPeRiX` computation for P. Then $C = \langle R_i, \langle IN_i, OUT_i\rangle\rangle_{i=0}^{\infty}$ is an `ASPeRiX` computation for P: it satisfies Revision principles of an `ASPeRiX` computation and it trivially satisfies Convergence too. By Theorem 2, C converges to an answer set IN_∞. □

Proof. (of Theorem 3) Lemmas 4 and 5 prove each one direction of the equivalence. □