Appendix
Towards Practical Tabled Abduction Usable in Decision Making

Ari Saptawijaya, Luís Moniz Pereira

Centro de Inteligência Artificial (CENTRIA)
Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa

KES IDT 2013
Sesimbra, 28 June 2013
Query Transformation

- Add (input and output) abductive contexts
- Conjoin with $not \ false$, to meet ICs
  - ICs in program are translated as any other rules.
  - In case no ICs, add $not\_false(l, l)$ in the program.
- In case of negative query, made it “positive”.
- Example: ?- $not\ p$.
- This query is called as a top goal:

  \[ ?- not\_p([\ ], T), not\_false(T, O). \]
Program Transformation: Program with Variables

\[
p(1) \leftarrow a(1). \quad q(2) \leftarrow a(2).
\]

- Query: \(?- p(X). \)
  - Abductive solution: \([a(1)] \quad X = 1; \)
  - Abductive solution: \([a(2)] \quad X = 2; \)
  - no

- Query: \(?- \neg p(X). \)
  - Abductive solution: \([\neg a(1), \neg a(2)] \quad X = _{h171} \)
  - Dual rule for \(p\), i.e., \(\neg p(Y)\), is obtained from two different \(p\) rules.
    - Failing \(p\) should be achieved by calling \(\neg p_1\) and \(\neg p_2\) \textit{independently}, and succeeding both.
    - Different variants of calling argument \(Y\) should be used in the calls of \(\neg p_1\) and \(\neg p_2\).

\[
\neg p(Y, l, O) \leftarrow \text{variant}(Y, Y_1), \neg p_1(Y_1, l, T), \\
\text{variant}(Y, Y_2), \neg p_2(Y_2, T, O).
\]
Program Transformation: Loops

- Mostly employ XSB-Prolog’s tabling to deal with loops.
- $P_3$:
  - $p \leftarrow q$. $q \leftarrow p$.
    - Direct positive loop: $?-p$. is correctly answered: ‘no’.
      - Detected via loop between tabled predicates $p^{ab}$ and $q^{ab}$.
    - What about query: $?-not \ p$.
      - It loops, instead of ‘yes’.

\[
\text{not}_p(I,O) \leftarrow \text{not}_p(I,O). \quad \text{not}_p(I,O) \leftarrow \text{not}_q(I,O).
\text{not}_q(I,O) \leftarrow \text{not}_q(I,O). \quad \text{not}_q(I,O) \leftarrow \text{not}_p(I,O).
\]

- Detect such loops by maintaining an ancestor list (with just negative “not_” literals)
  \[
  \text{not}_p \sim
  \]
  ancestor: [ ]
- When a positive literal is called, reset ancestor list to [ ].

- Additionally, negative loops over negation are also handled by the transformation, e.g., programs like

$P_4$:

\[
p \leftarrow q. \quad q \leftarrow not \ p.
\]
Program Transformation: Loops

- Mostly employ XSB-Prolog’s tabling to deal with loops.

  $P_3 : \quad p \leftarrow q. \quad q \leftarrow p.$

  - Direct positive loop: $?-p.$ is correctly answered: ‘no’.
    - Detected via loop between tabled predicates $p^{ab}$ and $q^{ab}$.
  - What about query: $?-not\ p$.
    - It loops, instead of ‘yes’.

        not\_p(I, O) \leftarrow not\_p_1(I, O). \quad not\_p_1(I, O) \leftarrow not\_q(I, O).
        not\_q(I, O) \leftarrow not\_q_1(I, O). \quad not\_q_1(I, O) \leftarrow not\_p(I, O).

  - Detect such loops by maintaining an ancestor list (with just negative “not_” literals)
    
    ancestor: \[ \] \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad [not\_p]

    - When a positive literal is called, reset ancestor list to \[ \].

- Additionally, negative loops over negation are also handled by the transformation, e.g., programs like

  $P_4 : \quad p \leftarrow q. \quad q \leftarrow not\ p.$
Program Transformation: Loops

- Mostly employ XSB-Prolog’s tabling to deal with loops.
- $P_3 : \quad p \leftarrow q. \quad q \leftarrow p.$
  - Direct positive loop: $?-p$. is correctly answered: ‘no’.
    - Detected via loop between tabled predicates $p^{ab}$ and $q^{ab}$.
  - What about query: $?-not \ p$.
    - It loops, instead of ‘yes’.
      
      \[
      not_p(I, O) \leftarrow not_p_1(I, O). \quad not_p_1(I, O) \leftarrow not_q(I, O). \\
      not_q(I, O) \leftarrow not_q_1(I, O). \quad not_q_1(I, O) \leftarrow not_p(I, O). \\
      \]
    
    - Detect such loops by maintaining an ancestor list (with just negative “not” literals)
      
      \[
      not_p \leadsto not_q \leadsto not_p
      \]
      
      ancestor: [ ] [not_p] [not_p, not_q] loop!
    
    - When a positive literal is called, reset ancestor list to [ ].
  
  - Additionally, negative loops over negation are also handled by the transformation, e.g., programs like

  \[
  P_4 : \quad p \leftarrow q. \quad q \leftarrow not \ p.
  \]
Evaluation: Tabling Abductive Solutions

Program:

- $q_0(0, 1)$.  
- $q_1(1)$.  
- $q_n(n)$.  

Missing solutions, IC: $\leftarrow \text{not } q_m(1001)$

![Abduction Time Missing Solutions: n = 1000](image)

**Figure:** The abduction time for debugging missing solutions $q_m(1001)$, for $100 \leq m \leq 1000$, and with the program size $n = 1000$. 