Tabled Abduction in Logic Programs

Ari Saptawijaya, Luís Moniz Pereira

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

ICLP 2013
Istanbul, 27 August 2013
Abductive Logic Programming

- Abduction: from observed evidence to its best explanation
- Abduction in Logic Programs
  - Rules:
    - shoes_wet ← grass_wet.
    - grass_wet ← sprinkler_running.
    - grass_wet ← rained.
    - clothes_wet ← rained.
    - clothes_dry.
    - IC: false ← clothes_wet, clothes_dry.
  - Abducibles: sprinkler_running, rained.
  - Query: ?- shoes_wet, not false.
  - Abductive solutions: sprinkler_running

- Applications: diagnosis, decision making, reasoning of rational agents, …
Tabled Abduction: Motivation & Main Idea

\[ P_1 : \quad q \leftarrow a. \quad r \leftarrow b, q. \quad p \leftarrow r, q. \]

- Abducibles: \{a, b\}
- Query: ?- q. ?- r. ?- p.
  - Explaining q: [a].
  - Explaining r: recompute q?
  - Explaining p: recompute r and q?
- Adopt tabling in LP, for abductive solution reuse
  - Solutions reuse in distinct context!
- Example
  - ?-q: table [a] as solution to q.

<table>
<thead>
<tr>
<th>Goal</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>[a]</td>
</tr>
</tbody>
</table>
Tabled Abduction: Motivation & Main Idea

\[ P_1 : \quad q \leftarrow a. \quad r \leftarrow b, q. \quad p \leftarrow r, q. \]

- Abducibles: \( \{a, b\} \)
- Query: \( \text{-} q. \quad \text{-} r. \quad \text{-} p. \)
  - Explaining \( q \): \( [a] \).
  - Explaining \( r \): recompute \( q \)?
  - Explaining \( p \): recompute \( r \) and \( q \)?
- Adopt tabling in LP, for abductive solution reuse
  - Solutions reuse in distinct context!
- Example
  - \( \text{-} q \): table \( [a] \) as solution to \( q \).
  - \( \text{-} r \): reuse solution \( q \) with context \( [b] \), but

<table>
<thead>
<tr>
<th>Goal</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q )</td>
<td>( [a] )</td>
</tr>
<tr>
<td>( r )</td>
<td>( [a, b] )</td>
</tr>
</tbody>
</table>
Tabled Abduction: Motivation & Main Idea

$$P_1 : q \leftarrow a, r \leftarrow b, q, p \leftarrow r, q.$$  

- Abducibles: \{a, b\}
- Query: \(?- q, ?- r, ?- p.\)
  - Explaining \(q\): \([a]\).
  - Explaining \(r\): recompute \(q\)?
  - Explaining \(p\): recompute \(r\) and \(q\)?
- Adopt tabling in LP, for abductive solution reuse
  - Solutions reuse in distinct context!
- Example
  - \(?-q\): table \([a]\) as solution to \(q\).
  - \(?-r\): reuse solution \(q\) with context \([b]\), but
  - \(?-p\): reuse solution \(q\) with \(r\)'s solution (\([a, b]\)) as its context.

<table>
<thead>
<tr>
<th>Goal</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>([a])</td>
</tr>
<tr>
<td>r</td>
<td>([a, b])</td>
</tr>
<tr>
<td>p</td>
<td>([a, b])</td>
</tr>
</tbody>
</table>
Program Transformation: Tabling Solutions

- Table abductive solution entry
  - XSB-Prolog tabling
  - \( P_1 : \ q \leftarrow a. \quad r \leftarrow b, q. \quad p \leftarrow r, q. \)
  - Table \( q^{ab}/1, \ r^{ab}/1, \) and \( p^{ab}/1 \)
    \[
    q^{ab}([a]). \\
    r^{ab}(E) \leftarrow q([b], E). \\
    p^{ab}(E) \leftarrow r([], T), q(T, E). 
    \]
- Re-uptake context-independent solution \( E \) from “ab” tables into different input contexts \( I \)
  \[
  q(I, O) \leftarrow q^{ab}(E), \text{prod}(O, I, E). \\
  r(I, O) \leftarrow r^{ab}(E), \text{prod}(O, I, E). \\
  p(I, O) \leftarrow p^{ab}(E), \text{prod}(O, I, E). 
  \]
- \( \text{prod}/3 \): produces consistent abduction result in \( O \)
Program Transformation: Dealing with “not”

- \( P_2 : \quad p \leftarrow a, \neg q. \quad q \leftarrow a, b. \quad q \leftarrow c. \)
- Abductive solutions of \( \neg q \)
  - Needs to compute all abductive solutions for \( q \), before negating them,
- Dual rules for negation, via dual transformation
  - Produce negation rules from the positive ones.
  - Find solutions incrementally.
  - Replace default literal \( \neg q \) by \( \neg q \):
    \[
    p^{ab}(E) \leftarrow \neg q([a], E).
    \]
  - Provide dual rules, e.g., for \( \neg q \)
    \[
    \neg q(I, O) \leftarrow \neg q_1(I, T), \neg q_2(T, O).
    \]
    \[
    \neg q_1(I, O) \leftarrow \not a(I, O).
    \]
    \[
    \neg q_1(I, O) \leftarrow \not b(I, O).
    \]
    \[
    \neg q_2(I, O) \leftarrow \not c(I, O).
    \]
Other Aspects

- Dual transformation for programs with variables.
- Dealing with loops (i.e., positive loops and loops over negation) in the presence of abduction.
  - Employs XSB tabling as much as possible, e.g., for positive loops.
  - Pragmatic approaches to deal with positive loops in dualized negation (e.g., $\text{not} \_ q \leftarrow \text{not} \_ q$) and negative loops over negation.
- Implementation:
  - Transformation of predicates comprised just of facts.
  - Lazy by-need dual transformation by storing dual rules in a trie (available in XSB), in lieu of batched table scheduling.
- Evaluation: benefit of tabling abductive solutions, variants of dual transformation, tabling nogoods of subproblems, programs with loops.
Conclusions and Future Work

- Addressed the issue of tabling abductive solutions.
  - Abductive solution reuse from one context to others.
  - Abduction under negative goals with dual transformation.
  - Various pragmatics to deal with loops and dual transformation.

- Future work:
  - Answer subsumption on abductive solutions.
  - Integrating TABDUAL with program updates (EVOLP/R) and other logic programming features.
  - Application to abductive moral decision making and others.
Thank you!