Implementing 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

SDIA 2013
Angra do Heroísmo, 9 September 2013
Abduction

From observed evidence to some best explanation

Example

Previous beliefs:
- The shoes are wet if the grass is wet.
- The grass is wet if the sprinkler was running.
- The grass is wet if it rained.
- The clothes outside are wet if it rained.
- The clothes are dry.

Integrity Constraint: Clothes are not both dry and wet.

Abducibles: the sprinkler was running, it rained

Satisfying IC + Observation “The shoes are wet”

Single Explanation: the sprinkler was running
Abductive Logic Programming

- Abduction in Logic Programs
- Example (cont’d)
  - 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. \quad ?- r. \quad ?- p.
  - Explaining \( q \): \( [a] \).
  - Explaining \( r \): recompute \( q \)?
  - Explaining \( p \): recompute \( r \) and \( q \)?
- Adopt **tabling** in LP, for abductive solution reuse
  - Solutions reused in distinct contexts!
- 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:** \(?- q. \quad ?- r. \quad ?- p. \)
  - Explaining \( q \): [a].
  - Explaining \( r \): recompute \( q \)?
  - Explaining \( p \): recompute \( r \) and \( q \)?
- **Adopt tabling** in LP, for abductive solution reuse
  - Solutions reused in distinct contexts!
- **Example**
  - \(?- q \): table \([a]\) as solution to \( q \).
  - \(?- r \): 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

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

- Abducibles: \{a, b\}
- Query: \(?- q. \quad ?- r. \quad ?- p.\)
  - Explaining \(q\): \([a]\).
  - Explaining \(r\): recompute \(q\)?
  - Explaining \(p\): recompute \(r\) and \(q\)?
- Adopt tabling in LP, for abductive solution reuse
  - Solutions reused in distinct contexts!
- 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 $p^{ab}/1$

```prolog
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 $I$.

```prolog
q(I, O) \leftarrow q^{ab}(E), prod(O, I, E).
r(I, O) \leftarrow r^{ab}(E), prod(O, I, E).
p(I, O) \leftarrow p^{ab}(E), prod(O, I, E).
```

- $prod/3$: produces consistent abduction result in $O$. 
Program Transformation: Dealing with “not”

- $P_2 : \quad p \leftarrow a, \text{not } q. \quad q \leftarrow a, b. \quad q \leftarrow c.$
- Abductive solutions of not $q$
  - Needs to compute all abductive solutions for $q$, before negating them,
- Dual rules for negation, via dual transformation$^1$
  - Produce negation rules from the positive ones.
  - Find solutions incrementally.
  - Replace default literal $\text{not } q$ by $\text{not}_q$:
    $$p^{ab}(E) \leftarrow \text{not}_q([a], E).$$
  - Provide dual rules, e.g., for $\text{not}_q$
    $$\text{not}_q(I,O) \leftarrow \text{not}_{q_1}(I,T), \text{not}_{q_2}(T,O).$$
    $$\text{not}_{q_1}(I,O) \leftarrow \text{not}_a(I,O).$$
    $$\text{not}_{q_1}(I,O) \leftarrow \text{not}_b(I,O).$$
    $$\text{not}_{q_2}(I,O) \leftarrow \text{not}_c(I,O).$$

Implementation of Tabled Abduction (TABDUAL)

Technical progress in implementing TABDUAL system:

1. Approaches to realizing dual transformation *by-need*.
2. Modular mixes of abductive and non-abductive program parts.
   ▶ Simpler treatment of facts in programs.
3. Access to ongoing abductive solutions for dynamic manipulations.
By-need Dual Transformation

- Naive complete dual transformation
  - Produces \textit{all} dual rules, in advance.
  - Few might be invoked during abduction.
- Solution: compute dual rules \textit{by need}.
  - Based on the need of on-going invoked goals, during abduction.
- Dual rules for \( q \) in \( P_2 \):
  
  \[
  \text{not} \ q(I, O) \leftarrow \text{not} \ q_1(I, T), \text{not} \ q_2(T, O).
  \]
  
  \[
  \text{not} \ q_i(I, O) \leftarrow \text{dual}(i, q, I, O).
  \]
- Reduce extra computation in abduction, by memoizing generic dual rules.
  - Two approaches: tabling and trie.
Approach 1: Tabling Generic Dual Rules

Idea:

(I) Create and table generic dual rules,
(II) Instantiate these generic dual rules.

▶ *Eager* construction of dual rules, due to local table scheduling.
▶ Example: \textit{dual_rule}(1, q, Dual), both alternatives of dual rules are constructed, before \textit{call\_dual}/4 is invoked.
▶ Can we construct a generic dual rule lazily?
  ▶ Employ batched scheduling, but it requires new installation.
  ▶ Keep local scheduling: what to do?
Approach 2: Storing Dual Rules in a Trie

- XSB allows facts to be stored and manipulated in tries.
- Generic dual rule as fact: $d(N, P, Dual, Pos)$.
  - Generic dual rule $Dual$ from $N$-th rule of $P$ with tracking position $Pos$.
  - $Pos$: position of literals used in constructing the latest dual rule.
- Idea (trie $T$ has been created):
  1. Try to reuse generic dual rules stored in $T$, one at a time through backtracking.
  2. Construct the next generic dual rule lazily (and store it in $T$) by means of $Pos$.
- Lazy by-need dual rules construction.
- Memoizing dual rules is carried out explicitly, plus additional tracking information is maintained.
Transforming Facts

- Transformation of predicates comprised just of facts.
- Naive transformation of fact $p(1)$:
  \[ p_{ab}(1, []). \quad p(X, I, O) \leftarrow p_{ab}(X, E), \text{produce}(O, I, E). \]
  \[ \text{not } p(X, I, I) \leftarrow X \neq 1. \]
- But facts do not induce any abduction.
- Simpler transformation:
  \[ p(X, I, I) \leftarrow p(X). \quad \text{not } p(X, I, I) \leftarrow \text{not } p(X). \]
- All facts of $p/1$ defined non-abductive part of input program:
  \[ \text{beginProlog. } \quad p(1). \quad p(2). \quad p(3). \quad \text{endProlog.} \]
Accessing Abductive Solutions

- The need for dynamic manipulation of ongoing abductive solution:
  - Filtering solutions using preferences.
  - Eliminate nogood combinations (those violate constraints).
- TABDUAL encapsulates ongoing abductive solution in abductive context.
- The only way to access and manipulate ongoing abductive solutions: modify the transform directly, rather than original problem representation.
- Example:
  \[
  q \leftarrow r, \text{ab}dQ(s), t. \quad s(X) \leftarrow \text{prolog}(\text{preferred}(X)).
  \]
  - Transform: \( q_{ab}(E) \leftarrow r([ ], T_1), s(T_1, T_1, T_2), t(T_2, E). \)
  - \textit{preferred}/1 defines some preference rules, in non-abductive part.
Conclusions and Future Work

- Addressed the issue of tabling abductive solutions.
  - Abductive solution reuse from one context to another.
  - Abduction under negative goals with dual transformation.
- Progress in implementation towards its more practical use.
  - Two approaches of by-need dual transformation.
  - Simpler transformation of facts.
  - Accessing and manipulating ongoing abductive solution.
- Evaluation of TABDUAL: GAI 2 (EPIA) on Wed, Sept. 11.
- Future work:
  - Answer subsumption and interned ground terms on tabling abductive solutions.
  - Integrating TABDUAL with program updates (EVOLP/R) and other logic programming features.
  - Application to abductive moral decision making and others, plus evaluation.
Thank you!