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
Abduction (1)

- From observed evidence to its best explanation
- Example
  - 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.
  - Observation
    - The shoes are wet.

- Minimal explanations:
  - “The grass is wet”, or
  - “The sprinkler was running”, or
  - “It rained”.

Abduction (1)

- From observed evidence to its best explanation
- Example
  - 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.
  - Observation
    - The shoes are wet.
  - Abducibles:
    - “The sprinkler was running”,
    - “It rained”.
  - Minimal explanations:
    - “The grass is wet”, or
    - “The sprinkler was running”, or
    - “It rained”.
Abduction (2)

▶ Consistent explanations, not necessarily minimal.
▶ 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.
  ▶ Plus, new beliefs:
    ▶ The clothes outside are wet if it rained.
    ▶ The clothes are dry.
    ▶ Integrity Constraint: Clothes are not both dry and wet.
  ▶ Same 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 with TABDUAL
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 \(r\): recomputing \(q\)?
  - Explaining \(p\): recomputing \(r\) and \(q\)?
- Adopt \underline{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: \(?- q. \quad ?- r. \quad ?- p.\)
  - Explaining \( q \): \([a]\).
  - Explaining \( r \): recompute \( q \)\
  - Explaining \( p \): recompute \( r \) and \( q \)\
- Adopt tabling in LP, for abductive solution reuse
  - Solutions reuse in distinct context!
- 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 \underline{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
  - \( P_1 : \ q \leftarrow a. \ r \leftarrow b, q. \ p \leftarrow r, q. \)
  - 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 \( I \)
  
  \[ q(I, O) \leftarrow q^{ab}(E), prod(O, I, E). \]
  
  \[ r(I, O) \leftarrow r^{ab}(E), prod(O, I, E). \]
  
  \[ p(I, O) \leftarrow p^{ab}(E), prod(O, I, E). \]

- \( prod/3 \): produces consistent abduction result in \( O \)
Program Transformation: Dealing with “not”

- $P_2 : \quad p \leftarrow a, \neg q. \quad q \leftarrow a, b. \quad q \leftarrow c.$

- 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 $\neg q$
    \[ \neg q(I, O) \leftarrow \neg q_1(I, T), \neg q_2(T, O). \]
    \[ \neg q_1(I, O) \leftarrow \neg a(I, O). \]
    \[ \neg q_1(I, O) \leftarrow \neg b(I, O). \]
    \[ \neg q_2(I, O) \leftarrow \neg c(I, O). \]
TABDUAL Extensions and Applications in Decision Making
Picking up Abduction-based Actions

- Decision making under hypothetical reasoning
- Given an observation:
  - Several scenarios exist, each characterized by abducibles
  - Decisions are based on explanatory abducibles
- Example:

```
smoke ← fire. smoke ← tear_gas.
beginProlog.
decide(call_firefighters, Abds) ← member(fire, Abds).
decide(police_protection, Abds) ← member(tear_gas, Abds).
endProlog.
```

- Top-goal queries: \texttt{do(Action, Abducibles, Observation)}
  - \texttt{?- do(Action, Abducibles, smoke).}
    \texttt{Action = call_firefighters, Abducibles = [fire];}
    \texttt{Action = police_protection, Abducibles = [tear_gas]}

Declarative Debuging: Incorrect Solutions

A buggy program $P$:

\[
\begin{align*}
  a(1). & \quad a(X) \leftarrow b(X), c(Y, Y). \\
  b(2). & \quad b(3). \quad c(1, X). \quad c(2, 2).
\end{align*}
\]

- Transformation (inc/2 abducible):
  
  \[
  \begin{align*}
  a(1) & \leftarrow \text{not } inc(1, [1]). \\
  a(X) & \leftarrow b(X), c(Y, Y), \text{not } inc(2, [X]). \\
  b(2) & \leftarrow \text{not } inc(3, [2]). \\
  b(3) & \leftarrow \text{not } inc(4, [3]). \\
  c(1, X) & \leftarrow \text{not } inc(5, [1, X]). \\
  c(2, 2) & \leftarrow \text{not } inc(6, [2, 2]).
  \end{align*}
  \]

- **IC**: false $\leftarrow a(3)$.

- **Solutions**:
  
  $\text{[inc}(2, [3])], \text{[inc}(4, [3])], \text{[inc}(5, [1, 1])], \text{inc}(6, [2, 2])].$
Declarative Debugging: Missing Solutions

The same buggy program $P$:

\[
\begin{align*}
    a(1). & \quad a(X) \leftarrow b(X), c(Y, Y). \\
    b(2). & \quad b(3). \quad c(1, X). \quad c(2, 2).
\end{align*}
\]

- **Transformation** ($\text{miss}/1$ abducible):
  
  Program $P$ plus
  
  \[
  \begin{align*}
    a(X) & \leftarrow \text{miss}(a(X)). \\
    b(X) & \leftarrow \text{miss}(b(X)). \\
    c(X, Y) & \leftarrow \text{miss}(c(X, Y)).
  \end{align*}
  \]

- **IC**: $\text{false} \leftarrow \text{not } a(5)$.

- **Solutions**:
  
  $[\text{miss}(a(5))], [\text{miss}(b(5))], [\text{miss}(b(5)), \text{miss}(c(X, X))].$
A Medical Dental Case

percussion_pain ← periapical_lesion.
percussion_pain ← fracture.

radiolucency ← periapical_lesion.

fracture ← horizontal_fracture.
elliptic_fracture_trace ← horizontal_fracture.
tooth_mobility ← horizontal_fracture.

fracture ← vertical_fracture.
decompression_pain ← vertical_fracture.
A Medical Dental Case

percussion_pain ← periapical_lesion.
percussion_pain ← fracture.

radiolucency ← periapical_lesion.

fracture ← horizontal_fracture.
elliptic_fracture_trace ← horizontal_fracture.
tooth_mobility ← horizontal_fracture.

fracture ← vertical_fracture.
decompression_pain ← vertical_fracture.

false ← not percussion_pain.
A Medical Dental Case

percussion_pain ← periapical_lesion.
percussion_pain ← fracture.

radiolucency ← periapical_lesion.

fracture ← horizontal_fracture.
elliptic_fracture_trace ← horizontal_fracture.
tooth_mobility ← horizontal_fracture.

fracture ← vertical_fracture.
decompression_pain ← vertical_fracture.

false ← not percussion_pain.
false ← tooth_mobility.
A Medical Dental Case

percussion_pain ← periapical_lesion.
percussion_pain ← fracture.

radiolucency ← periapical_lesion.

fracture ← horizontal_fracture.
elliptic_fracture_trace ← horizontal_fracture.
tooth_mobility ← horizontal_fracture.

fracture ← vertical_fracture.
decompression_pain ← vertical_fracture.

false ← not percussion_pain.
false ← tooth_mobility.

?- fracture([], T), not_false(T, O).
  O = [horizontal_fracture] X
percussion_pain ← periapical_lesion.  
percussion_pain ← fracture.

radiolucency ← periapical_lesion.

fracture ← horizontal_fracture.  
elliptic_fracture_trace ← horizontal_fracture.  
tooth_mobility ← horizontal_fracture.

fracture ← vertical_fracture.  
decompression_pain ← vertical_fracture.

false ← not percussion_pain.  
false ← tooth_mobility.

?− fracture([ ], T), not_false(T, O).
  ▶ O = [horizontal_fracture] ✗
  ▶ O = [vertical_fracture] ✓
A Medical Dental Case

\begin{align*}
\text{percussion\_pain} & \leftarrow \text{periapical\_lesion}. \\
\text{percussion\_pain} & \leftarrow \text{fracture}. \\
\text{radiolucency} & \leftarrow \text{periapical\_lesion}. \\
\text{fracture} & \leftarrow \text{horizontal\_fracture}. \\
\text{elliptic\_fracture\_trace} & \leftarrow \text{horizontal\_fracture}. \\
\text{tooth\_mobility} & \leftarrow \text{horizontal\_fracture}. \\
\text{fracture} & \leftarrow \text{vertical\_fracture}. \\
\text{decompression\_pain} & \leftarrow \text{vertical\_fracture}. \\
\text{false} & \leftarrow \text{not\ percussion\_pain}. \\
\text{false} & \leftarrow \text{tooth\_mobility}. \\
\text{?} & \leftarrow \text{fracture}([\text{vertical\_fracture}], T), \text{not\ false}([\text{vertical\_fracture}], O).
\end{align*}

\begin{itemize}
\item \text{O} = [\text{horizontal\_fracture}] \times
\item \text{O} = [\text{vertical\_fracture}] \checkmark
\item \text{O} = [\text{periapical\_lesion, vertical\_fracture}] \checkmark
\end{itemize}
Conclusions and Future Work

- Addressed the issue of tabling abductive solutions: TABDUAL
- Improved and extended TABDUAL towards more practical use
  - System predicate for abducible-based actions
  - System predicate for accessing ongoing abductive solutions
  - Other improvements: simpler facts transformation, dual by-need, ...
- Showed declarative debugging as abduction
- Applied TABDUAL to medical diagnosis
- Future work and application:
  - Perfecting implementation
  - Integrating TABDUAL with program updates and other logic programming features
  - Application to abductive moral decision making
Thank you!