Inspection Points and Meta-Abduction In Logic Programs

Luís Moniz Pereira    Alexandre Pinto
Centre for Artificial Intelligence - CENTRIA
Universidade Nova de Lisboa

INAP’09, 5-7 November  Évora, Portugal
Summary Motivation

- With Abductive Logic Programs (ALPs), when finding an abductive solution for a query, one may want to check too whether some other literals become true (or false) as a consequence, i.e., are side-effects determined by the abductive solution found.
- For efficiency, we need to inspect those side-effects without having to produce a complete model.
- We show how this type of reasoning requires a new mechanism, not provided by others already available.
- To achieve it, we present the concept of Inspection Point in ALPs, and show, how to employ it to investigate side-effects of interest (the inspection points).
Abduction, or inference to the best explanation, is a reasoning method that chooses hypotheses that, if true, would best explain observed evidence.

In LPs, abductive hypotheses—abducibles—are named literals which have no rules

- Motivation
- Inspection Points: side-effects checking
- Declarative Semantics of IPs
- Implementation of Inspection Points
- Comparisons, Conclusions, Future Work
Motivation (1/2)

- ALPs are used for modeling abductive problems
- Abductive query-answering is a backward-chaining process. Finding side-effects of abductive assumptions is forward-chaining
- Full-fledged forward-chaining => too many (often irrelevant) conclusions are derived
- Better solution is selective forward-chaining: user specifies conclusions of interest; only those are computed by forward-chaining
- We simulate selective forward chaining by backward-chaining from the focused on conclusions — the Inspection Points — with abduction turned off
Motivation (2/2)

Example:

\[
\begin{align*}
\text{← thirsty, not drink.} & \quad \text{drunk ← drink_bear.} \\
\text{thirsty.} & \quad \text{unsafe_drive ← inspect(drunk).} \\
\text{drink ← drink_bear.} & \quad \text{use_glass ← drink.} \\
\text{drink ← drink_water.} & \quad \text{wet_glass ← use_glass.}
\end{align*}
\]

- We use \textit{inspect}(drunk) in the rule for \textit{unsafe_drive} to check its truth as a consequence of abductions made, not to enforce it
- The IC ← not unsafe_drive would force the abduction of drink_bear
- Truth value of wet_glass and use_glass is irrelevant, so whole model computation is wasteful
- This \textit{unsafe_drive}'s rule allows for all abductive solutions (drink_water and drink_bear): unsafe_drive is just a side-effect
Another example:

\[
\begin{align*}
dust & \leftarrow \text{cleaning\_day}, \text{inspect}(\text{not sound\_alarm}) \\
sound\_alarm & \leftarrow \text{temperature\_rise} \\
sound\_alarm & \leftarrow \text{faulty\_alarm} \\
evacuate & \leftarrow \text{sound\_alarm} \\
& \quad \leftarrow \text{not cleaning\_day}
\end{align*}
\]

- On \textit{cleaning\_days} the cleaning team must dust the Nuclear Power Plant, but only if the alarm is not sounding.

- In order to dust we must check (inspect) that the alarm is not sounding ≠ enforcing the alarm is not sounding.

- Checking (inspecting) does not allow any abductions to take place, as opposed to “regular” abductive query-answering.
Inspection Points—Declarative Semantics

- IPs declarative Semantics given by program transformation
- Only SMs of transform where each \textit{abduced}(x) is enjoined by abducible \( x \) are intended models of original program \( P \)
- Transform consists of \( P \), plus even loops for any abducible \( x \) and for \textit{abduced}(x), and:

1. all the rules obtained by the rules in \( P \) by systematically replacing:
   - \textit{inspect}(not \( L \)) with not \textit{inspect}(\( L \));
   - \textit{inspect}(\( a \)) or \textit{inspect}(\textit{abduced}(\( a \))) with \textit{abduced}(\( a \))
     if \( a \) is an abducible, and keeping \textit{inspect}(\( a \)) otherwise.

2. for every rule \( A \leftarrow L_1, \ldots, L_t \) in \( P \), the additional rule:
   \( \textit{inspect}(A) \leftarrow L'_1, \ldots, L'_t \) where for every \( 1 \leq i \leq t \):
   \[
   L'_i = \begin{cases} 
   \text{i} & \text{if } L_i \text{ is an abducible} \\
   \text{inspect}(X) & \text{if } L_i \text{ is } \textit{inspect}(X) \\
   \text{inspect}(L_i) & \text{otherwise}
   \end{cases}
   \]
Example:

\[
\begin{align*}
    x &\leftarrow a, \text{inspect}(y), b, c, \text{not } d &
    y &\leftarrow \text{inspect}(\text{not } a) \\
    z &\leftarrow d &
    y &\leftarrow b, \text{inspect}(\text{not } z), c
\end{align*}
\]

Transformed program = original + the following rules:

\[
\begin{align*}
    \text{inspect}(x) &\leftarrow \text{abduced}(a), \text{inspect}(y), \text{abduced}(b), \text{abduced}(c), \text{not } \text{abduced}(d) \\
    \text{inspect}(z) &\leftarrow \text{abduced}(d) \\
    \text{inspect}(y) &\leftarrow \text{not } \text{abduced}(a) \\
    \text{inspect}(y) &\leftarrow \text{abduced}(b), \text{not } \text{inspect}(z), \text{abduced}(c)
\end{align*}
\]

For a query on \(x\), the unique abductive SM for the transformed program – where each \(\text{abduced}(L)\) is matched by \(L\) in the SM – is:

\[
\{x, y, a, b, c, \text{abduced}(a), \text{abduced}(b), \text{abduced}(c), \text{inspect}(x), \text{inspect}(y)\}
\]
Implementation of Inspection Points

- IPs are implemented as add-on to ABDUAL — ALP meta-interpreter for top-down querying — where predicate consume/1 stands for abduced/1.

- When solving a query, if meta-interpreter finds inspect(L) it:
  1. increments the inspect_counter variable (initialized to 0)
  2. runs a “normal” query on literal L
  3. decrements the inspect_counter

- When meta-interpreter finds abducible x:
  - if inspect_counter is zero, adds x to abductions list — if list consistent
  - if inspect_counter is greater than zero, adds consume(x) to abductions list

- After query completes, checks if all consume(x) match with x in abductions list
Implementation of Inspection Points

Running example (abducibles are $a, b, c, d$):

- $x \leftarrow a, \text{inspect}(y), b, c, \text{not } d$
- $y \leftarrow \text{inspect(\text{not } a)}$
- $z \leftarrow d$
- $y \leftarrow b, \text{inspect(\text{not } z)}, c$

Abductive query: $?- x.$

<table>
<thead>
<tr>
<th>Current atom</th>
<th>Current rule</th>
<th>inspect_counter</th>
<th>Abduced Lits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Implementation of Inspection Points

Running example (abducibles are $a, b, c, d$):

$$x \leftarrow a, \text{inspect}(y), b, c, \text{not} d$$

$$z \leftarrow d$$

$$y \leftarrow \text{inspect}(\text{not } a)$$

$$y \leftarrow b, \text{inspect}(\text{not } z), c$$

Abductive query: $?- x.$

<table>
<thead>
<tr>
<th>Current atom</th>
<th>Current rule</th>
<th>inspect_counter</th>
<th>Abduced Lits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$x \leftarrow a, \text{inspect}(y), b, c, \text{not } d$</td>
<td>0</td>
<td>[]</td>
</tr>
</tbody>
</table>
Implementation of Inspection Points

Running example (abducibles are \(a, b, c, d\)):

\[
x \leftarrow a, \text{inspect}(y), b, c, \text{not } d \quad \text{y} \leftarrow \text{inspect(\text{not } a)}
\]

\[
z \leftarrow d \quad \text{y} \leftarrow b, \text{inspect(\text{not } z)}, c
\]

Abductive query: \(?- x.\)

<table>
<thead>
<tr>
<th>Current atom</th>
<th>Current rule</th>
<th>inspect_counter</th>
<th>Abduced Lits</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>(x \leftarrow a, \text{inspect}(y),) (b, c, \text{not } d)</td>
<td>0</td>
<td>([a])</td>
</tr>
</tbody>
</table>
Implementation of Inspection Points

Running example (abducibles are $a, b, c, d$):

$x \leftarrow a, \text{inspect}(y), b, c, \text{not } d$

$y \leftarrow \text{inspect}(\text{not } a)$

$z \leftarrow d$

$y \leftarrow b, \text{inspect}(\text{not } z), c$

Abductive query: ?- $x$.

<table>
<thead>
<tr>
<th>Current atom</th>
<th>Current rule</th>
<th>inspect_counter</th>
<th>Abduced Lits</th>
</tr>
</thead>
<tbody>
<tr>
<td>inspect($y$)</td>
<td>$x \leftarrow a, \text{inspect}(y), b, c, \text{not } d$</td>
<td>1</td>
<td>[a]</td>
</tr>
</tbody>
</table>
Implementation of Inspection Points

Running example (abducibles are $a, b, c, d$):

$x \leftarrow a, \text{inspect}(y), b, c, \text{not } d$

$y \leftarrow \text{inspect(}\text{not } a\text{)}$

$z \leftarrow d$

$y \leftarrow b, \text{inspect(}\text{not } z\text{), } c$

Abductive query: $? - x$.

<table>
<thead>
<tr>
<th>Current atom</th>
<th>Current rule</th>
<th>inspect_counter</th>
<th>Abduced Lits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>$y \leftarrow \text{inspect(}\text{not } a\text{)}$</td>
<td>1</td>
<td>[a]</td>
</tr>
</tbody>
</table>

FAIL!
Implementation of Inspection Points

Running example (abducibles are \( a, b, c, d \)):
\[
\begin{align*}
x & \leftarrow a, \text{inspect}(y), b, c, \text{not} \ d \\
z & \leftarrow d
\end{align*}
\]
\[
\begin{align*}
y & \leftarrow \text{inspect}(\text{not} \ a) \\
y & \leftarrow b, \text{inspect}(\text{not} \ z), c
\end{align*}
\]

Abductive query: \(- x.\)

<table>
<thead>
<tr>
<th>Current atom</th>
<th>Current rule</th>
<th>inspect_counter</th>
<th>Abduced Lits</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y )</td>
<td>( y \leftarrow b, \text{inspect}(\text{not} \ z), c )</td>
<td>1</td>
<td>( [a] )</td>
</tr>
</tbody>
</table>
Implementation of Inspection Points

Running example (abducibles are $a, b, c, d$):

$x \leftarrow a, \text{inspect}(y), b, c, \text{not } d$

$y \leftarrow \text{inspect}(\text{not } a)$

$z \leftarrow d$

$y \leftarrow b, \text{inspect}(\text{not } z), c$

Abductive query: $?- x.$

<table>
<thead>
<tr>
<th>Current atom</th>
<th>Current rule</th>
<th>inspect_counter</th>
<th>Abduced Lits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>$y \leftarrow b,$</td>
<td>1</td>
<td>[a,consume(b)]</td>
</tr>
<tr>
<td></td>
<td>$\text{inspect}(\text{not } z), c$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Implementation of Inspection Points

Running example (abducibles are $a$, $b$, $c$, $d$):

$x \leftarrow a, \text{inspect}(y), b, c, \text{not } d$

$z \leftarrow d$

$y \leftarrow \text{inspect(not } a)$

$y \leftarrow b, \text{inspect(not } z), c$

Abductive query: $?- x.$

<table>
<thead>
<tr>
<th>Current atom</th>
<th>Current rule</th>
<th>inspect_counter</th>
<th>Abduced Lits</th>
</tr>
</thead>
<tbody>
<tr>
<td>inspect(not $z$)</td>
<td>$y \leftarrow b, \text{inspect(not } z), c$</td>
<td>2</td>
<td>[a,consume(b)]</td>
</tr>
</tbody>
</table>
### Implementation of Inspection Points

Running example (abducibles are $a$, $b$, $c$, $d$):

- $x \leftarrow a, \text{inspect}(y), b, c, \neg d$
- $y \leftarrow \text{inspect}(\neg a)$
- $z \leftarrow d$
- $y \leftarrow b, \text{inspect}(\neg z), c$

Abductive query: ?- $x$.

<table>
<thead>
<tr>
<th>Current atom</th>
<th>Current rule</th>
<th>inspect_counter</th>
<th>Abduced Lits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z$</td>
<td>$z \leftarrow d$</td>
<td>2</td>
<td>[$a$, consume($b$)]</td>
</tr>
</tbody>
</table>
Implementation of Inspection Points

Running example (abducibles are $a, b, c, d$):

\[
x \leftarrow a, \text{inspect}(y), b, c, \text{not } d
\]
\[
y \leftarrow \text{inspect(\text{not } a)}
\]
\[
z \leftarrow d
\]
\[
y \leftarrow b, \text{inspect(\text{not } z)}, c
\]

Abductive query: ?- $x$.

<table>
<thead>
<tr>
<th>Current atom</th>
<th>Current rule</th>
<th>inspect_counter</th>
<th>Abduced Lits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d$</td>
<td>$z \leftarrow d$</td>
<td>2</td>
<td>[a, consume(b), consume(\text{not } d)]</td>
</tr>
</tbody>
</table>
Implementation of Inspection Points

Running example (abducibles are a, b, c, d):
\[
x \leftarrow a, \text{inspect}(y), b, c, \text{not } d
\]
\[
y \leftarrow \text{inspect}(\text{not } a)
\]
\[
z \leftarrow \text{not } d
\]
\[
y \leftarrow b, \text{inspect}(\text{not } z), c
\]

Abductive query: \(?- x\).

<table>
<thead>
<tr>
<th>Current atom</th>
<th>Current rule</th>
<th>inspect_counter</th>
<th>Abduced Lits</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>( y \leftarrow b, \text{inspect}(\text{not } z), c )</td>
<td>1</td>
<td>([a, \text{consume}(b), \text{consume}(\text{not } d), \text{consume}(c)])</td>
</tr>
</tbody>
</table>
Implementation of Inspection Points

Running example (abducibles are $a$, $b$, $c$, $d$):

$x \leftarrow a, \text{inspect}(y), b, c, \text{not } d$

$z \leftarrow d$

Abductive query: ?- $x$.

<table>
<thead>
<tr>
<th>Current atom</th>
<th>Current rule</th>
<th>inspect_counter</th>
<th>Abduced Lits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>$x \leftarrow a, \text{inspect}(y), b, c, \text{not } d$</td>
<td>0</td>
<td>[a,\text{consume}(b), \text{consume}(\text{not } d), \text{consume}(c), b]</td>
</tr>
</tbody>
</table>
Running example (abducibles are $a, b, c, d$):

$x \leftarrow a, \text{inspect}(y), b, c, \text{not } d$

$y \leftarrow \text{inspect}($not $a)$

$z \leftarrow d$

$y \leftarrow b, \text{inspect}(\text{not } z), c$

Abductive query: $?- x$.

<table>
<thead>
<tr>
<th>Current atom</th>
<th>Current rule</th>
<th>inspect_counter</th>
<th>Abduced Lits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c$</td>
<td>$x \leftarrow a, \text{inspect}(y), b, c, \text{not } d$</td>
<td>0</td>
<td>$[a, \text{consume}(b), \text{consume}(\text{not } d), \text{consume}(c), b, c]$</td>
</tr>
</tbody>
</table>
Implementation of Inspection Points

Running example (abducibles are a, b, c, d):

\[ x \leftarrow a, \text{inspect}(y), b, c, \text{not } d \]
\[ y \leftarrow \text{inspect}(\text{not } a) \]
\[ z \leftarrow d \]
\[ y \leftarrow b, \text{inspect}(\text{not } z), c \]

Abductive query: \(?- x.\)

<table>
<thead>
<tr>
<th>Current atom</th>
<th>Current rule</th>
<th>inspect_counter</th>
<th>Abduced Lits</th>
</tr>
</thead>
<tbody>
<tr>
<td>not d</td>
<td>[ x \leftarrow a, \text{inspect}(y), b, c, \text{not } d ]</td>
<td>0</td>
<td>[a,consume(b), consume(not d), consume(c),b,c,not d]</td>
</tr>
</tbody>
</table>

Post-processing: checking if all ‘consume(L)’ match with ‘L’, in the Abduced Lits.

Abduced Lits has: consume(b), consume(not d), consume(c), and b, c, not d \(\Rightarrow\) Match OK: query succeeds!
• HyProlog (Christiansen & Dahl) system allows for inspection, but only on abducibles
• IFF (Sadri & Toni) abductive proof procedure only allows inspecting negative abducibles
• We introduced a new general mechanism of side-effect inspection of any literals
• Gave its declarative semantics and implemented it in ABDUAL
• Possible application: planning/acting interleaving in multi-agent systems – check action pre-conditions before acting, without redoing action
Thank you!