Abductive Logic Programming with Tabled Abduction

Luís Moniz Pereira, Ari Saptawijaya

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

ICSEA 2012 – Lisbon
20 November 2012

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 (IC):**
      No clothes are 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, ...

Tabled Abduction: Motivation

$$P_1: \qquad q \leftarrow \mathbf{a}. \qquad r \leftarrow \mathbf{b}, q. \qquad p \leftarrow r, q.$$

- Abducibles: $\{\mathbf{a}, \mathbf{b}\}$
- 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
  - Table [a] as solution to ?-q.
- Solutions reuse in distinct context!

Goal	Solutions
q	[**a**]

Tabled Abduction: Motivation

$$P_1: \qquad q \leftarrow \mathbf{a}. \qquad r \leftarrow \mathbf{b}, q. \qquad p \leftarrow r, q.$$

- Abducibles: $\{\mathbf{a}, \mathbf{b}\}$
- 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
  - Table [a] as solution to ?-q.
- Solutions reuse in distinct context!
  - ?-r: reuse solution q with context [b], but

Goal	Solutions
q	[**a**]
r	[**a**, **b**]

Tabled Abduction: Motivation

$$P_1: \quad q \leftarrow \mathbf{a}. \quad r \leftarrow \mathbf{b}, q. \quad p \leftarrow r, q.$$

- Abducibles: $\{\mathbf{a}, \mathbf{b}\}$
- 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
  - Table [a] as solution to ?-q.
- Solutions reuse in distinct context!
  - ?-r: reuse solution q with context [b], but
  - ?-p: reuse solution q with r's solution ([a, b]) as its context.

Goal	Solutions
q	[**a**]
r	[**a**, **b**]
p	[**a**, **b**]

Program Transformation: Tabling Solutions

- Table abductive solution entry
  - XSB-Prolog tabling
- P_1 : $q \leftarrow \mathbf{a}.$ $r \leftarrow \mathbf{b}, q.$ $p \leftarrow r, q.$
  - Table $q^{ab}/1$, $r^{ab}/1$, and $p^{ab}/1$

$$q^{ab}([\mathbf{a}]).$$
$$r^{ab}(E) \leftarrow q([\mathbf{b}], E).$$
$$p^{ab}(E) \leftarrow r([\,], T), q(T, E).$$

  - Re-uptake context-independent solutions from "*ab*" tables into different contexts

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

  - $prod/3$: produces consistent output abduction result

Program Transformation: Dealing with "*not*"

- P_2 : $p \leftarrow \mathbf{a}, not\ q.$ $q \leftarrow \mathbf{a}, \mathbf{b}.$ $q \leftarrow \mathbf{c}.$
  - Abductive solutions of *not q*: compute first *all* abductive solutions for q, before negate them?
  - Finding solutions *incrementally*.
- Dual rules via dual transformation
  - Replace default literal *not q* to *not_q*

    $$p^{ab}(E) \leftarrow not\_q([\mathbf{a}], E).$$
  - Provide dual rules: *not_q*

$$not\_q(I, O) \leftarrow not\_q_1(I, T), not\_q_2(T, O).$$
$$not\_q_1(I, O) \leftarrow not\_a(I, O).$$
$$not\_q_1(I, O) \leftarrow not\_b(I, O).$$
$$not\_q_2(I, O) \leftarrow not\_c(I, O).$$

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*'.

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

    - Detect such loops by maintaining an ancestor list (with just negative "*not_*" literals)

$$not\_p \quad \rightsquigarrow$$

      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

$$P_4 : \quad p \leftarrow q. \quad q \leftarrow not\ p.$$

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*'.

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

    - Detect such loops by maintaining an ancestor list (with just negative "*not_*" literals)

$$not\_p \rightsquigarrow not\_q \rightsquigarrow$$
ancestor: [ ] [*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 : $p \leftarrow q.$ $q \leftarrow p.$
  - 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*'.

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

    - Detect such loops by maintaining an ancestor list (with just negative "*not_*" literals)

	not_p	$\rightsquigarrow$	not_q	$\rightsquigarrow$	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

$$P_4 : \quad p \leftarrow q. \quad q \leftarrow not\ p.$$

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 $not\_false(I, I)$ in the program.
- In case of negative query, made it "positive".
- Example: `?-`*not p*.
- This query is called as a top goal:

$$?\text{-}not\_p([\,], T), not\_false(T, O).$$

Comparison with ABDUAL and NegABDUAL

	ABDUAL	NegABDUAL	**TABDUAL**
Tabled solutions reuse	✗	✗	✓
Dual transformation	✓	✓	✓
Meta-interpreter	✓	✓	✗
Programs with variables	✗	✓	✓
Constructive negation	✗	✓	✗

Conclusions and Future Work

- Addressed the issue of tabling abductive solutions
- Achieved via program transformation
  - Table abductive solutions
  - Deal with negative literals
  - Deal with loops
  - Deal with programs and queries containing variables
- Future work:
  - Perfecting implementation
  - Evaluation TABDUAL
  - Application of TABDUAL
  - Migrating core features into an engine-level
    - Tabling abduction entries
    - Hiding data structures, e.g. the ancestor list

Thank you!