Towards Practical Tabled Abduction Usable in Decision Making

Ari SAPTAWIJAYA [1] and Luís Moniz PEREIRA
Centro de Inteligência Artificial (CENTRIA)
Departamento de Informática, Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
Email: ar.saptawijaya@campus.fct.unl.pt, lmp@fct.unl.pt

Abstract. Abduction as reasoning paradigm has been much explored in Artificial Intelligence, but not yet taken up by decision making as much as its potential warrants. Indeed, abduction permits the generation of hypothetical knowledge based scenarios, about which one can then equate decisions. One reason for this state of affairs is that abduction is difficult to implement efficaciously, even by experts, which entails that abductive systems are not readily available for decision making. Our concept of tabled abduction mitigates this, in the abductive logic programming system TABDUAL.
The contribution of this paper is three-fold: (1) We discuss some TABDUAL improvements towards its more practical use, particularly in decision making, (2) We show that declarative debugging can be viewed as abduction in logic programming, thus showing another potential of abduction for decision making, and (3) We describe how TABDUAL can be applied in decision making and examine its benefit.

Keywords. tabled abduction, abductive logic programming, declarative debugging, decision making.

1. Introduction

Reasoning by abduction is a well-known method in AI, studied at length in computational logic, and logic programming in particular, for quite a while [1–4]. In logic programming, it offers a formalism that, combined with other reasoning methods, affords us a declarative way to express and solve a variety of problems in a diversity of areas, e.g. in decision-making, diagnosis, planning, belief revision, and hypothetical reasoning (cf. [5–9]). Though many Prolog systems have matured enough to become of much practical use, abduction is not yet offered in them as a staple reasoning method, on account of the special and additional complexities involved. We have strived to change this state of affairs.

Finding hypothetical assumptions (i.e. abductive solutions) that lend support to a desired goal, or finding best (even satisficing) possible abductive explanations for observed evidence, can be very costly. Now, it can be the case that abductive solutions found in one context can then be appropriated in a different context and reused at little extra cost.

[1]Affiliated with Fakultas Ilmu Komputer at Universitas Indonesia, Depok, Indonesia.

In the absence of abduction, goal solution reuse in logic programming systems is by now a rather staple feature, which resorts to a tabling mechanism for effectiveness. However, abductive solution reuse in not readily amenable to tabling, exactly because solutions go together with an abduction context. It raises new problems on how to reuse them in distinct but compatible contexts, while catering to all varieties of loops in logic programs, complicated now by abduction.

The concept and realization of tabled abduction implementation for abductive normal logic programs was introduced recently to address this problem [10], which will be reviewed in Section 2. It is realized via a program transformation and implemented in a prototype, TABDUAL, using XSB-Prolog. The transformation makes use of the theory of the dual transformation [1], which allows to efficiently handle the problem of abduction under negative goals, by introducing the dual positive counterparts for them. Abduction is then performed on the transformed program without the need of a meta-interpreter.

We explore here TABDUAL's potential in decision making. First, we introduce a new TABDUAL construct for the purpose, adding to its features. It helps decision making on the basis of available hypothetical scenarios and permits modular mixes of abductive and non-abductive program parts. These and other improvements towards its more practical use are discussed in Section 3. Second, we revisit declarative debugging, a tool of interest to decision making, in Section 4, previously characterized as belief revision [11, 12], and show that it can be viewed as (tabled) abduction. Third, we describe how TABDUAL can be applied in decision making and examine its benefit, illustrating it with an example from medical diagnosis, in Section 5.

2. Tabled Abduction

First we recap some background definitions in logic programs. We then discuss how abduction can be accomplished in logic programs and recapitulate tabled abduction.

A *logic rule* has the form $H \leftarrow B_1, \ldots, B_m, not\ B_{m+1}, \ldots, not\ B_n$, where $n \geq m \geq 0$ and H, B_i with $1 \leq i \leq n$ are atoms. In a rule, H is called the head of the rule and $B_1, \ldots, B_m, not\ B_{m+1}, \ldots, not\ B_n$ its body. We employ '*not*' to denote default negation. The atom B_i and its default negation $not\ B_i$ are named positive and negative *literals*, respectively. When $n = 0$, we say the rule is a *fact* and render it simply as H. The atoms *true* and *false* are, by definition, respectively true and false in every interpretation. A rule in the form of a denial, i.e. with empty head, or equivalently with *false* as head, is an *integrity constraint* (IC). A *logic program* (LP) is a set of logic rules, where non-ground rules (i.e. rules containing variables) stand for all their ground instances. In this work we focus on *normal logic programs*, i.e. those whose heads of rules are positive literals or empty. As usual, we write p/n to denote predicate p with arity n.

2.1. Abduction in Logic Programs

In summary, abduction, or inference to the best explanation (a common designation of one of its uses in the philosophy of science [13, 14]), is a reasoning method, of interest to scenario building in decision making [15, 16], whereby one chooses those hypotheses that would, if true, best explain the observed evidence – while meeting any attending ICs – or that would satisfy some query. In LPs, abductive hypotheses (or *abducibles*) are named literals of the program which have no rules, and whose truth value is not initially assumed. Abducibles may have arguments, but they must be ground on the occasion of their abduction. An *abductive normal logic program* is a normal logic program that allows for abducibles, or their negations, to appear in the body of rules. Note that the negation '*not a*' of an abducible a does not refer to its default negation, as abducibles have no rules, but instead to the explicitly assumed hypothetical negation of a.

The truth value of abucibles may be independently assumed *true* or *false*, via either their positive or negated form, as the case may be, in order to produce an abductive solution to a query, that is a consistent set of assumed hypotheses that support it. An *abductive solution* to a query is a consistent set of abducible instances or their negations that, when substituted by their assigned truth value everywhere in the program P, affords us with a model of P (for the specific semantics used on P), which satisfies both the query and the ICs – a so-called *abductive model*.

Abduction in LPs can naturally be accomplished by a top-down query-oriented procedure to find an (abductive) solution to a query (by need, i.e. as abducibles are encountered), where the abducibles in the solution are leaves in its procedural query-rooted call-graph, i.e. the graph recursively engendered by the procedure calls from literals in bodies of rules to heads of rules, and thence to the literals in the rule's body.

2.2. Tabled Abduction in TABDUAL

The concept of tabled abduction is illustrated and summarized in the following examples. Consider an abductive logic program, taken from [10]:

Example 1. Program P_1: $q \leftarrow a. \quad s \leftarrow b, q. \quad t \leftarrow s, q.$
where a and b are abducibles. Suppose three queries: q, s, and t, are individually launched, in that order. The first query, q, is satisfied simply by taking $[a]$ as the abductive solution for q, and tabling it. Executing the second query, s, amounts to satisfying the two subgoals in its body, i.e. abducing b followed by invoking q. Since q has previously been invoked, we can benefit from reusing its solution, instead of recomputing, given that the solution was tabled. That is, query s can be solved by extending the current ongoing abductive context $[b]$ of subgoal q with the already tabled abductive solution $[a]$ of q, yielding $[a,b]$. The final query t can be solved similarly. Invoking the first subgoal s results in the priorly registered abductive solution $[a,b]$, which becomes the current abductive context of the second subgoal q. Since $[a,b]$ subsumes the previously obtained abductive solution $[a]$ of q, we can then safely take $[a,b]$ as the abductive solution to query t. This example shows how $[a]$, as the abductive solution of the first query q, can be reused from an abductive context of q (i.e. $[b]$ in the second query, s) to another context (i.e. $[a,b]$ in the third query, t). In practice the body of rule q may contain a huge number of subgoals, causing potentially expensive recomputation of its abductive solutions and thus such unnecessary recomputation should be avoided.

Tabled abduction is realized in a prototype, TABDUAL, which involves a program transformation of abductive logic programs. Abduction can then be enacted on the transformed program directly, without the need of a meta-interpreter. Example 1 already indicates two key ingredients of the transformation:

1. *Abductive context*, which relays the ongoing abductive solution from one subgoal to subsequent subgoals, as well as from the head to the body of a rule, via

input and *output* contexts, where abducibles can be envisaged as the terminals of parsing.

2. *Tabled predicates*, which table the abductive solutions for predicates defined in the input program.

Example 2. The rule $t \leftarrow s, q$ from Example 1 is transformed into two rules:

$$t_{ab}(E) \leftarrow s([\,],T), q(T,E). \qquad\qquad t(I,O) \leftarrow t_{ab}(E), produce(O,I,E).$$

Predicate $t_{ab}(E)$ is the tabled predicate which tables the abductive solution of t in its argument E. Its definition, in the left rule, follows from the original definition of t. Two extra arguments, that serve as input and output contexts, are added to the subgoals s and t in the rule's body. The left rule expresses that the tabled abductive solution E of t_{ab} is obtained by relaying the ongoing abductive solution in context T from subgoal s to subgoal q in the body, given the empty input abductive context of s (because there is no abducible in the body of the original rule of t). The rule on the right expresses that the output abductive solution O of t is obtained from the the solution entry E of t_{ab} and the given input context I of t, via TABDUAL system predicate $produce(O,I,E)$, that checks consistency. The other rules in Example 1 are transformed following the same idea.

An abducible is transformed into a rule that inserts it into the abductive context. For instance, the abducible a from Example 1 is transformed into: $a(I,O) \leftarrow insert(a,I,O)$, where $insert(a,I,O)$ is a TABDUAL system predicate which inserts a into the input context I, resulting in the output context O, while also checking consistency. The negation *not a* of the abducible a is transformed similarly, except that it is renamed into *not_a* in the head (as we consider normal logic programs): $not\_a(I,O) \leftarrow insert(not\ a,I,O)$.

The TABDUAL program transformation employs the *dual transformation* [1], which makes negative goals 'positive', thus permitting to avoid the computation of all abductive solutions, and then negating them, under the otherwise regular negative goals. Instead, we are able to obtain one abductive solution at a time, as when we treat abduction under positive goals. The dual transformation defines for each atom A and its set of rules R in a program P, a set of dual rules whose head *not_A* is true if and only if A is false by R in the employed semantics of P. Note that, instead of having a negative goal *not A* as the rules' head, we use its corresponding 'positive' one, *not_A*. Example 3 illustrates only the main idea of how the dual transformation is employed in TABDUAL and omits many details, e.g. checking loops in the input program, all of which are referred in [10].

Example 3. Suppose program P_2 contains two rules of $p/1$ (along with $q/1$ and $r/2$):

$$p(1) \leftarrow a(1). \qquad\qquad p(X) \leftarrow q(Y), r(X,Y).$$

where $a/1$ is abducible. The TABDUAL transformation will create a set of dual rules for $p/1$ which falsify $p/1$ with respect to its two above rules, i.e. by falsifying both the first rule *and* the second rule, expressed by predicate $p^{*1}/3$ and $p^{*2}/3$, respectively:

$$not\_p(X,I,O) \leftarrow p^{*1}(X,I,T), p^{*2}(X,T,O).$$

In TABDUAL, the above rule is known as the first layer of the dual transformation. The second layer contains the definitions of $p^{*1}/3$ and $p^{*2}/3$.

For $p^{*1}/3$, the first rule of p is falsified when either its argument is not equal to 1 or, if equal to 1, its body is falsified (i.e. the negation of $a(1)$ is abduced), so we have:

$$p^{*1}(X,I,I) \leftarrow X \neq 1. \qquad p^{*1}(1,I,O) \leftarrow not\_a(1).$$

In the first rule of $p^{*1}/3$, the content of the context I is simply relayed from the input to the output context. The abduction of *not* $a(1)$, i.e. by invoking the subgoal $not\_a(1)$ in the body of the second rule, is achieved via the rule of $not\_a/1$.

For $p^{*2}/3$, the second rule of p is falsified when either its argument does not unify with X, or its body is falsified (by negating one subgoal at a time). Since variable X must be unified by any term, the first alternative can be dropped. Thus, we have:

$$p^{*2}(X,I,O) \leftarrow not\_q(Y,I,O). \qquad p^{*2}(X,I,O) \leftarrow q(Y,I,T), not\_r(X,Y,T,O).$$

In the second rule of $p^{*2}/3$, besides negating the second subgoal r, the preceeding positive subgoal q is kept, since it may help instantiate variable Y when the second subgoal not_r is subsequently invoked. The abductive context in the body is relayed as usual.

Finally, TABDUAL transforms integrity constraints like any other rules, and top-goal queries are always launched by also satisfying integrity constraints. This transformation is illustrated, in Section 5, using an example from medical diagnosis.

3. Improvements on TABDUAL

TABDUAL has still much room for improvement. We extend here its features, by introducing a new system predicate that is relevant for decision making. We also mention other general improvements, which advance TABDUAL towards its more practical use.

3.1. Picking up Abducible-based Actions

In decision making under hypothetical reasoning, given an observation, one is typically confronted with several possible scenarios. These scenarios are characterized by the explanatory abducibles, and decisions are made on their basis. We illustrate it with a simple example.

Example 4. Suppose that an agent observes some smoke and its action decision with regard to this situation depends on the cause of the smoke. In case it is triggered by fire, the agent reacts by calling firefighters. But if it is explained by the presence of tear gas, then the agent better seeks police protection.

Top-goal queries for decision making can be enacted by means of a new TABDUAL system predicate $do(Act, Abds, Obs)$. It explains the given observation *Obs* by the abductive solution *Abds*, and subsequently picks the action *Act* based on the given definition of $decide/2$ in the agent's belief.

The situation can be modeled in TABDUAL as follows:

$smoke \leftarrow fire. \qquad smoke \leftarrow tear\_gas.$
$beginProlog.$
$decide(call\_firefighters, Abds) \leftarrow member(fire, Abds).$
$decide(police\_protection, Abds) \leftarrow member(tear\_gas, Abds).$
$endProlog.$

where $fire$ and $tear\_gas$ are abducibles. Note that *beginProlog* and *endProlog* are TABDUAL's identifiers to distinguish non-abducitve program part from the abductive one. All rules between these two identifiers are considered usual Prolog programs and are not subject to the TABDUAL transformation.

For Example 4, we have the top-goal query $do(Act, Abds, smoke)$, which gives us two scenarios and guides us with actions to take: $Act = call\_firefighters$ for $Abds = [fire]$ and $Act = police\_protection$ for $Abds = [tear\_gas]$.

3.2. Recalling Other Improvements

Besides predicate $do/3$, TABDUAL is also equipped with some other improvements mentioned below. These improvements, which support TABDUAL better for its practical use, are detailed elsewhere [17]:

- TABDUAL provides a system predicate to access the ongoing abductive solutions. This feature is useful and important for manipulating abductive solutions dynamically, e.g. to prefer some explanations, or eliminate so-called *nogood* combinations (those known to violate constraints). Like predicate $do/3$, it also permits modular mixes of abductive and non-abductive program parts.
- Predicates comprised of facts are transformed much simpler, i.e. they do not follow the general TABDUAL transformation applied for rules as explained in Section 2.2. The reason is that facts do not induce any abduction, and such simpler transformation helps deal with abductive logic programs having large factual data, which is often the case in many real world problems.
- Dual rules are constructed by need, thus avoiding a complete dual transformation for every defined atom in the program in advance during the transformation phase. The latter approach, though conceptually correct, is unpractical, as real-world problems typically consist of a huge number of rules, and the transformation may take ages and hinder abduction to take place. The transformed program still contains the same first layer of the dual transformation, but its second layer is defined by a rule, which will be interpreted by the TABDUAL system on-the-fly, during abduction, to produce the concrete definitions of the second layer. Extra computation load that may occur during the abduction phase, due to the by-need construction of dual rules, can be reduced by tabling the already constructed generic dual rules. Therefore, when such dual rules are later needed, they are available for reuse and thus their recomputation can be avoided.

4. Declarative Debugging as Abduction: a Preparatory Tool for Decision Making

When decision making rests upon knowledge that is expressed declaratively (e.g. in logic programs), declarative debugging becomes an important tool to ease the debugging of errors that may infiltrate the knowledge representation, in preparation for and maintenance of correct decision making. Declarative debugging of normal logic programs has been characterized before as belief revision [11, 12]. We revisit in this section two cases of declarative debugging, those of incorrect solutions and of missing solutions, and show that they can be viewed and implemented as abduction.

We start by considering the debugging of definite logic programs.

Example 5. Take a buggy program P_4 [11]:

$$a(1). \qquad a(X) \leftarrow b(X), c(Y,Y).$$
$$b(2). \qquad b(3). \qquad c(1,X). \qquad c(2,2).$$

Incorrect Solutions Suppose that $a(3)$ is an incorrect solution. To debug its cause, the program is first changed using the simple transformation introduced in [12], i.e. by adding default literal $not\ incorrect(i, [X_1, \ldots, X_n])$ to the body of each i-th rule of P_4, to defeasibly assume their correctness by default, where n is the rule's arity and X_is, for $1 \leq i \leq n$, its head arguments. This yields program P_4':

$$a(1) \leftarrow not\ incorrect(1, [1]). \qquad a(X) \leftarrow b(X), c(Y,Y), not\ incorrect(2, [X]).$$
$$b(2) \leftarrow not\ incorrect(3, [2]). \qquad b(3) \leftarrow not\ incorrect(4, [3]).$$
$$c(1,X) \leftarrow not\ incorrect(5, [1,X]). \qquad c(2,2) \leftarrow not\ incorrect(6, [2,2]).$$

In terms of abduction, one can envisage $incorrect/2$ as an abducible. To express, while debugging, that $a(3)$ is an incorrect solution, we add to P_4' an IC: $\leftarrow a(3)$. We run TABDUAL on P_4', which returns three solutions as the possible sufficient causes of the incorrect solution:

$$[incorrect(2, [3])], [incorrect(4, [3])], [incorrect(5, [1, 1]), incorrect(6, [2, 2])].$$

Missing Solutions Suppose $a(5)$ should be a solution of P_4, which is missing. To find this bug, P_4 is transformed [11], by adding for each predicate p/n the rule:

$$p(X_1, \ldots, X_n) \leftarrow missing(p(X_1, \ldots, X_n)).$$

For P_4, it is transformed into P_4'' that contains all rules from P_4 plus three new rules:

$$a(X) \leftarrow missing(a(X)). \quad b(X) \leftarrow missing(b(X)). \quad c(X,Y) \leftarrow missing(c(X,Y)).$$

Similarly as before, $missing/1$ can be viewed as an abducible. But now, to express that we miss $a(5)$ as a solution, we add to P_4'' an IC: $\leftarrow not\ a(5)$. TABDUAL returns the three abductive solutions on P_4'' as the causes of missing solution $a(5)$ in P_4:

$$[missing(a(5))], [missing(b(5))], [missing(b(5)), missing(c(X,X))].$$

Differently from [11, 12] where minimal solutions are targeted, TABDUAL also returns non-minimal solution $[missing(b(5)), missing(c(X,X))]$. In this case, TABDUAL allows one to choose those solutions that are sufficient so far, or interesting enough, and to continue searching for more solutions if needed, just like Prolog does. Finding minimal abductive solutions is not always desired – here bugs may well not be minimal. And in general, minimality can be deferred to a later step and paid for only if needed. Examination of side-effects is often more important than minimality [15].

Finally, in case of debugging normal logic programs, the two above debugging transformations are adroitly summed into one, as illustrated in Example 6. TABDUAL takes care of further transforming the default *not*'s into positive atoms by dualizing them, to make the program a definite one.

Example 6. Consider program P_5:

$$a \leftarrow not\ b. \qquad a \leftarrow c. \qquad b.$$

We obtain P_5^* by applying the two transformations:

$$a \leftarrow not\ b, not\ incorrect(1). \qquad a \leftarrow c, not\ incorrect(2). \qquad b \leftarrow not\ incorrect(3).$$
$$a \leftarrow missing(a). \qquad b \leftarrow missing(b). \qquad c \leftarrow missing(c).$$

Suppose we want to explain the causes of missing solution a, then we add to P_5^* an IC: $\leftarrow not\ a$. Running TABDUAL on P_5^*, we obtain three abductive solutions:

$$[incorrect(3)], [missing(a)], [missing(c)]$$

which correctly enunciate the three possible causes of the problem.

5. Evaluating TABDUAL in Decision Making: a Medical Case

Next, we show how TABDUAL is applied to medical diagnosis, adapted from [18].

Example 7. A patient shows up at the dentist with signs of pain upon teeth percussion but without tooth mobility. The expected causes for the observed signs are periapical lesion, horizontal fracture, and vertical fracture of the root and/or crown.

An abductive logic program representing a partial medical knowledge base of the practitioner is as follows:

$percussion\_pain \leftarrow periapical\_lesion$
$percussion\_pain \leftarrow fracture$

$radiolucency \leftarrow periapical\_lesion$

$fracture \leftarrow horizontal\_fracture$
$elliptic\_fracture\_trace \leftarrow horizontal\_fracture$
$tooth\_mobility \leftarrow horizontal\_fracture$

$fracture \leftarrow vertical\_fracture$
$decompression\_pain \leftarrow vertical\_fracture$

$\leftarrow not\ percussion\_pain$
$\leftarrow tooth\_mobility$

where $periapical\_lesion$, $horizontal\_fracture$, and $vertical\_fracture$ are abducibles. The integrity constraints indicate that the practitioner must conclude $percussion\_pain$ but not $tooth\_mobility$ since these are the symptoms of the patient that requires explanation.

Suppose that during examination, the practitioner suspects that there is a fracture. This corresponds to query $fracture$ with its corresponding transformed top-goal: $fracture(I,T), not\_false(T,O)$. Notice that abductive context T, which is the ongoing abductive solution to $fracture$ is relayed to not_false in order to satisfy integrity constraints. Recall that integrity constraints are transformed like any other rules (cf. Section 2.2). In particular, predicate $not\_false/2$ is defined, by the dual transformation, as follows:

$not\_false(I,O) \leftarrow false^{*1}(I,T), false^{*2}(T,O).$
$false^{*1}(I,O) \leftarrow percussion\_pain(I,O).$
$false^{*2}(I,O) \leftarrow not\_tooth\_mobility(I,O).$

The first subgoal $fracture$ gives two abductive solutions: $T = [horizontal\_fracture]$ and $T = [vertical\_fracture]$. The second subgoal not_false, constrains these two solutions further:

- It eliminates $[horizontal\_fracture]$, due to rule $false^{*2}/2$. The latter rule, which eventually abduces *not* $horizontal\_fracture$, makes the first abductive solution inconsistent.
- With respect to $[vertical\_fracture]$, the integrity constraint results in two final abductive solutions: $O = [periapical\_lesion, vertical\_fracture]$ and $O = [vertical\_fracture]$. Notice that, in the derivation of $false^{*1}$ to eventually obtain $O = [vertical\_fracture]$, TABDUAL allows reusing the tabled abductive solution from the subgoal $fracture$, thus showing the benefit of tabled abduction in this problem.

We have also evaluated TABDUAL in terms of performance and scalability, where we consider four distinct TABDUAL variants (of the same underlying implementation), obtained by separately factoring out its important features. We particularly examined two benchmarks: the well-known N-queens problem, where abduction is used to find safe board configurations of N queens, and an example of declarative debugging. The evaluation of the first benchmark aims at evaluating the scalability of TABDUAL, concentrating on tabling of nogoods of subproblems, i.e. tabling conflictual configurations of queens (essentially, tabling the ongoing abductive solutions). The result is promising, that TABDUAL indeed benefits from tabling ongoing abductive solutions: as constraints become more complex, its performance consistently surpasses that of its non-tabling counterpart. The second benchmark focuses on the relative worth of the dual transformation by need in TABDUAL, with respect to both the transformation and the abduction time. Though the dual transformation by need requires more time during abduction, which is to be expected due to the on-the-fly dual rules construction, it saves the transformation time significantly and thus the total time as a whole. The evaluation details can be found in [19].

6. Conclusion and Future Work

We have improved TABDUAL, adding a novel feature by introducing a new system predicate to facilitate decision making, apart from other improvements. We believe that these improvements will gear up TABDUAL better for practical applications, namely in decision making. We also illustrated that declarative debugging, a tool of interest to perfecting knowledge-based decision making, can be viewed as abduction, and hence readily implemented in an abduction system like TABDUAL. Finally, we applied TABDUAL to medical diagnosis and showed how it can benefit from tabled abduction. An issue that we have touched upon in the TABDUAL's evaluation, reported elsewhere [19], is tabling nogoods of subproblems, and how it may improve performance, with good scalability with problem complexity. As for the future, we look forward to applying TABDUAL, integrating it with other logic programming features (such as updating and uncertainty), to moral decision making, where we have already made a start [20, 21].

Acknowledgements Ari Saptawijaya acknowledges the support of FCT-Portugal, grant SFRH/BD/72795/2010.

References

[1] J. J. Alferes, L. M. Pereira, and T. Swift, "Abduction in well-founded semantics and generalized stable models via tabled dual programs," *Theory and Practice of Logic Programming*, vol. 4, no. 4, pp. 383–428, 2004.

[2] T. Eiter, G. Gottlob, and N. Leone, "Abduction from logic programs: semantics and complexity," *Theoretical Computer Science*, vol. 189, no. 1-2, pp. 129–177, 1997.

[3] A. Kakas, R. Kowalski, and F. Toni, "The role of abduction in logic programming," in *Handbook of Logic in Artificial Intelligence and Logic Programming* (D. Gabbay, C. Hogger, and J. Robinson, eds.), vol. 5, Oxford U. P., 1998.

[4] M. Denecker and A. C. Kakas, "Abduction in logic programming," in *Computational Logic: Logic Programming and Beyond*, Springer Verlag, 2002.

[5] A. C. Kakas and A. Michael, "An abductive-based scheduler for air-crew assignment," *J. of Applied Artificial Intelligence*, vol. 15, no. 1-3, pp. 333–360, 2001.

[6] J. F. Castro and L. M. Pereira, "Abductive validation of a power-grid expert system diagnoser," in *Procs. 17th Intl. Conf. on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEA-AIE'04)*, vol. 3029 of *LNAI*, pp. 838–847, Springer, 2004.

[7] J. Gartner, T. Swift, A. Tien, C. V. Damásio, and L. M. Pereira, "Psychiatric diagnosis from the viewpoint of computational logic," in *Procs. 1st Intl. Conf. on Computational Logic (CL 2000)*, vol. 1861 of *LNAI*, pp. 1362–1376, Springer, 2000.

[8] R. Kowalski and F. Sadri, "Abductive logic programming agents with destructive databases," *Annals of Mathematics and Artificial Intelligence*, vol. 62, no. 1, pp. 129–158, 2011.

[9] R. Kowalski, *Computational Logic and Human Thinking: How to be Artificially Intelligent*. Cambridge U. P., 2011.

[10] L. M. Pereira and A. Saptawijaya, "Abductive logic programming with tabled abduction," in *Procs. 7th Intl. Conf. on Software Engineering Advances (ICSEA)*, pp. 548–556, ThinkMind, 2012.

[11] L. M. Pereira, C. V. Damásio, and J. J. Alferes, "Debugging by diagnosing assumptions," in *Automatic Algorithmic Debugging*, vol. 749 of *LNCS*, pp. 58–74, Springer, 1993.

[12] L. M. Pereira, C. V. Damásio, and J. J. Alferes, "Diagnosis and debugging as contradiction removal in logic programs," in *Progress in Artificial Intelligence*, vol. 727 of *LNAI*, Springer, 1993.

[13] J. R. Josephson and S. G. Josephson, *Abductive Inference: Computation, Philosophy, Technology*. Cambridge U. P., 1995.

[14] P. Lipton, *Inference to the Best Explanation*. Routledge, 2001.

[15] L. M. Pereira and A. M. Pinto, "Side-effect inspection for decision making," in *Procs. 1st KES Intl. Symposium on Intelligent Decision Technologies (KES-IDT'09)*, vol. 199 of *Springer Studies in Computational Intelligence*, pp. 139–150, 2009.

[16] B. Brogaard, "A Peircean Theory of Decision," *Synthese*, vol. 118, no. 3, pp. 383–401, 1999.

[17] A. Saptawijaya and L. M. Pereira, "Implementing tabled abduction in logic programs." Submitted to Doctoral Symposium on Artificial Intelligence (SDIA), Available at `http://centria.di.fct.unl.pt/~lmp/publications/online-papers/implementing_tabdual.pdf`, 2013.

[18] L. M. Pereira, P. Dell'Acqua, A. M. Pinto, and G. Lopes, "Inspecting and preferring abductive models," in *The Handbook on Reasoning-Based Intelligent Systems* (K. Nakamatsu and L. C. Jain, eds.), pp. 243–274, World Scientific Publishers, 2013.

[19] A. Saptawijaya and L. M. Pereira, "Towards practical tabled abduction in logic programs." Submitted to 16th Portuguese Conference on Artificial Intelligence (EPIA), Available at `http://centria.di.fct.unl.pt/~lmp/publications/online-papers/abduction_tabling.pdf`, 2013.

[20] L. M. Pereira and A. Saptawijaya, "Modelling Morality with Prospective Logic," in *Machine Ethics* (M. Anderson and S. L. Anderson, eds.), pp. 398–421, Cambridge U. P., 2011.

[21] T. A. Han, A. Saptawijaya, and L. M. Pereira, "Moral reasoning under uncertainty," in *Procs. of The 18th Intl. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR-18)*, vol. 7180 of *LNCS*, pp. 212–227, Springer, 2012.