Tabling in Contextual Abduction with Answer Subsumption

Syukri Mullia Adil Perkasa
Faculty of Computer Science
Universitas Indonesia
Email: syukri.mullia@ui.ac.id

Ari Saptawijaya
Faculty of Computer Science
Universitas Indonesia
Email: saptawijaya@cs.ui.ac.id

Luís Moniz Pereira
NOVA-LINCS Laboratory for
Computer Science and Informatics
Universidade Nova de Lisboa
Email: lmp@fct.unl.pt

***Abstract* — Abduction is a form of logical inference that seeks out best explanations for a given observation. Abduction has already been well studied in the field of computational logic, and logic programming in particular. In contextual abduction, explanations obtained within one context are also relevant in different contexts. In such contextual abduction, explanations thus can be reused with little cost. When abduction is realized in logic programming, one can reuse previously obtained explanations from one context to another by benefiting from a logic programming's feature called tabling. In this paper, we revisit tabling in contextual abduction and improve this technique with an advanced tabling feature of XSB Prolog, viz., *anwser subsumption*. The employment of answer subsumption in this technique is important, when one is interested in obtaining minimal explanations for an observation. It also helps avoid tabling too many and large explanations for a given observation, which may fail contextual abduction in practice, as it requires too many resources before being able to return a solution. We provide a protoytpe, TABDUAL$^+$, of this improved technique as a proof of concept. Our experiments, both in artificial and real world cases, show that TABDUAL$^+$ correctly returns minimal explanations, while the cost of their computation is greatly reduced.**

1. Introduction

Abductive reasoning, also known as *abduction*, is a form of logical inference that seeks out best explanations for a given observation.

Example 1.1. Consider the knowledge base below:

$$Grass\ is\ wet\ \textbf{\textit{if}}\ it\ rained. \quad (1)$$
$$Grass\ is\ wet\ \textbf{\textit{if}}\ sprinkler\ was\ on. \quad (2)$$
$$Shoes\ are\ wet\ \textbf{\textit{if}}\ grass\ is\ wet\ and\ it\ rained. \quad (3)$$

Given an observation that the shoes are wet, abduction produces two explanations (say, E_1 and E_2):

- The first explanation (E_1) is 'it rained', obtained by statements (1) then (3);
- The second one (E_2) is 'the sprinkler was on' and 'it rained', obtained by statements (2) then (3).

Notice that in abduction, *basic* explanations are required, in the sense that these explanations cannot be explained further. For instance, 'grass is wet' is not considered a basic explanation, as it can further be explained, either by statements (1) or (2).

Abduction has already been well studied in the field of computational logic, and logic programming in particular, for a few decades by now [1], [2], [3]. Abduction in logic programming, known as *abductive logic programming*, offers a formalism to express and solve problems declaratively and has been used widely in a variety of areas, e.g., decision-making, diagnosis, planning, belief revision, and hypothetical reasoning [4], [5], [6], [7], [8].

In abduction, it is often the case that explanations (commonly termed as *abductive solutions*) obtained within one context are also relevant in a different context; a concept known as *contextual abduction*. These relevant abductive solutions can actually be reused with little cost. Continuing Example 1.1, after explaining 'shoes are wet', one may want to find the explanation of the same observation but in the context of hypothesizing that 'the sprinkler was off'. In finding the explanation of the latter observation (within that specific context), the abductive solutions obtained for the former observation (viz., E_1 and E_2) can be reused. Here, the context that 'the sprinkler was off' eliminates E_2 from possible solutions (due to a contradiction about the sprinkler), thus leaving one explanation as the only abductive solution (obtained by extending E_1 with the newly hypothesized context), viz., 'it rained' and 'the sprinkler was off' .

In logic programming, the technique of reusing solutions to a goal (in general, not necessarily for abductive reasoning) is commonly known as *tabling*, and it has currently been featured in some Prolog systems. Conceptually, tabling is also applicable to abduction too: in this case by reusing previously obtained abductive solutions from one abductive context in another. This technique, called *tabling in contextual abduction*, has been proposed by Saptawijaya and Pereira [3], whose concept and implementation aspects were detailed, wrapped into a system called TABDUAL, and implemented in XSB Prolog [9].

Despite its advantage of reusing previously obtained abductive solutions, TABDUAL may suffer from excessive computational cost, particularly when many and large al-

ternative explanations for an observation are tabled. This is typically the case when one deals with large scale of abduction problems. Several implementation aspects have been introduced in [3] to alleviate this issue. In this paper, we further improve TABDUAL with an advanced tabling feature of XSB Prolog, viz., *anwser subsumption* [10], resulting in a prototype: TABDUAL$^+$.

Answer subsumption allows tabling of only preferred answers (abductive solutions) to a query (an observation). That is, any subsumed answers in a table are deleted and only the subsuming answer (which is the one preferred) is stored in the table. Adding answer subsumption into TABDUAL does not only help in dealing with excessive computational cost, but it has also one important consequence: answer subsumption permits abduction to deliver *minimal* explanations, or most general explanations, criteria often applied in generating explanations [4]. For instance, in Example 1.1, delivering E_1 as the explanation for the wet shoes may be preferred to E_2, if the minimality criterion for explanations are imposed. This means that, under this criterion, there is no need to table E_2, but only E_1.

We evaluate the benefit of answer subsumption using TABDUAL$^+$, both in artificial and real world problems. The artifical problem is designed to evaluate the scalability of TABDUAL$^+$ with respect to the total table space used due to answer subsumption. The result shows that as the size of the problem increases, the total table space used is significantly reduced when abductive solutions are tabled with answer subsumption. For the real world case, an abduction problem concerning chemoprevention is considered [11]. While tabling without answer subsumption fails in returning explanations (abductive solutions) for more than half cases of this problem, TABDUAL$^+$ with its answer subsumption successfully delivers minimal explanations for all cases of this real world problem very efficiently, both in terms of execution time and the total table space used.

The rest of the paper is organized as follows. Section 2 presents necessary background on logic programming and abductive logic programming. The technique of tabling in contextual abduction is subsequently described, in Section 3. Section 4 details TABDUAL$^+$ as an extension of tabling in contextual abduction with answer subsumption. Experiments using TABDUAL$^+$ and their results are discussed in Section 5. The paper concludes with future work, in Section 6.

2. Preliminaries

This section presents background concepts on logic programming and abduction in logic programming, those necessary to understand the rest of the paper.

2.1. Logic Program

A *term* over alphabet $\mathcal{A}$ in language $\mathcal{L}$ is defined recursively as either a variable (conventionally written as a capital letter), a constant, or an expression with the form of $f(t_1, \ldots, t_n)$ where f is a function symbol of $\mathcal{A}$ and $t_i s$ are terms. An *atom* over $\mathcal{A}$ is defined as an expression with the form of $p(t_1, \ldots, t_n)$ where p is a predicate symbol of $\mathcal{A}$ and $t_i s$ are terms. The notation of p/n is used to denote a predicate symbol p having arity n. A literal is either an atom a (called positive literal) or its negation $not\ a$ (called default literal). A term (respectively, atom and literal) is *ground* if it does not contain variables.

A (*normal*) *logic program* is a countable set of *rules* with the form of:

$$H \leftarrow L_1, \ldots, L_m$$

where H is an atom, $m \geq 0$, and $L_i s (1 \leq i \leq m)$ are literals. The comma in the rule is read as a conjunction. A rule with the form $H \leftarrow$ is called a *fact*, and written just H instead.

Example 2.1. Logic program P below represents sentences in Example 1.1:

$$g \leftarrow r.$$
$$g \leftarrow sp.$$
$$sh \leftarrow g, r.$$

where 'grass is wet', 'it rained', 'sprinkler was on', and 'shoes are wet' are denoted as g, r, sp, and sh, respectively.

2.2. Abductive Logic Programming

Abduction in logic programming is realized by extending logic programs with abductive hypotheses, called *abducibles*. Moreover, specific rules in the form of denial, called *integrity constraints* are introduced to express some restrictions that have to be fulfilled when performing abduction.

An *abducible* is an atom Ab or its negation Ab^* (syntactically an atom, but denoting $not\ Ab$), respectively named *positive* and *negative abducible*, whereas an *integrity constraint* is a rule of the form:

$$\bot \leftarrow L_1, \ldots, L_m.$$

where $\bot/0$ is a reserved predicate symbol (denoting $false$) in $\mathcal{L}$, $m \geq 1$, and $L_i s$ $(1 \leq i \leq m)$ are literals.

The logic program, abducibles, and integrity constraints are wrapped into a structure called *abductive framework* $\langle P, \mathcal{AB}, \mathcal{IC} \rangle$, where $\mathcal{AB}$ is the set of abducible predicates and their corresponding arity, P is a logic program over $\mathcal{L}$ such that there is no rule in P whose head is an abducible, and $\mathcal{IC}$ is the set of integrity constraints. Note that an observation in abduction is analogous to a query goal in logic programming.

Example 2.2. Recall Example 2.1, where we can consider and abductive framework $\langle P, \{r/0, sp/0\}, \emptyset \rangle$:

- P is the logic program in Example 2.1 that represents the problem in Example 1.1;
- The set $\mathcal{AB}$ consists of abducible predicates r and sp, all with arity 0;
- The set $\mathcal{IC}$ of integrity constraints is $\emptyset$: the problem has no restriction (integrity constraint) imposed in accepting the abductive solutions.

Given a query, abduction amounts to find abductive solutions, i.e., finding *consistent* truth values for abducibles that make both the program rules satisfied and the query true, while satisfying the integrity constraints.

Abduction in logic programming can be accomplished by a top-down query-oriented procedure for finding its solution, by need. The correctness of this top-down computation requires a semantics to be relevant, because it avoids computing the whole model. Instead, it suffices to use only rules that are relevant to the query. The *Well-Founded Semantics* enjoys this property, i.e., it allows finding only relevant abducibles and their truth value through the top-down query-oriented procedure.

Example 2.3. Recall the abductive framework $\langle P, \{r/0, sp/0\}, \emptyset \rangle$ from Example 2.2. Given query sh (i.e., the observation 'shoes are wet') on this abductive framework, the top-down query-oriented procedure returns two abductive solutions, viz., $[r]$ and $[sp, r]$ (referring to E_1 and E_2 of Example 1.1, respectively).

3. Tabling in Contextual Abduction

It is often the case that abductive solutions obtained within one context are also relevant in a different context; a concept known as *contextual abduction*.

Example 3.1. Recall Example 2.3. Suppose that after issuing query sh (and obtaining its two solutions: $[r]$ and $[sp, r]$), the same query is posed, but now within the given abductive context $[sp^*]$ ('the sprinkler was off (*not* on)'). One may benefit from *reusing* (rather than recomputing) the previously obtained solutions $[r]$ and $[sp, r]$ of the same query, but now taking into account the abductive context $[sp^*]$. As abductive solutions should be consistent, $[r, sp^*]$ is thus the only solution for query $[sh]$ within the abductive context sp^*. No consistent abductive solution can be obtained from $[sp, r]$ within the context $[sp^*]$.

In logic programming, absent from abduction, *tabling* is a technique by reusing solutions to a goal rather than recomputing them, by storing a goal and its answers obtained from query evaluation. One may therefore benefit from tabling for reusing previously obtained abductive solutions, as illustrated in Example 3.1. But, in practice, an abductive solution to a goal Q is not immediately amenable to be stored in a table, because such solution is typically tied to Q's abductive context. The technique for reusing previously obtained abductive solutions from one abductive context to another by making use of tabling is called *tabling in contextual abduction*, proposed in [3]. To fully realize this idea, [3] provides a prototype named TABDUAL, developed using XSB Prolog, a Prolog system that is based on the Well-Founded Semantics. TABDUAL consists of several program transformations:

1) *Transformation for tabling abductive solutions* that accommodates storing and reusing abductive solutions by making use of XSB's tabling feature.
2) *Transformation for producing dualized negation* that enables TABDUAL to deal with abduction under negative goals, which is based on the *dual program transformation* of ABDUAL [12].
3) *Transformation for inserting abducibles* into an abductive context that helps realize contextual abduction itself.
4) *Transformation of a query* that warrants any abductive solution to satisfy the integrity constraints.

Note that only the *transformation for tabling abductive solutions* is discussed in this section, as it is the one subject to the extension with answer subsumption. For the discussion of the other three transformations, the reader is referred to [3].

In the transformation for tabling abductive solutions, every rule in the program is transformed, producing a rule of a tabled predicate with one additional argument as the entry of its tabled abductive solution. The abductive solution is obtained by relaying the ongoing abductive solution through the rule's subgoals. Example 3.2 illustrates this transformation and its important ingredients.

Example 3.2. Recall P in Example 2.1. The rule $sh \leftarrow g, r$ is transformed into following two rules:

$$sh_{ab}(E) \leftarrow g([r], E). \quad (4)$$
$$sh(I, O) \leftarrow sh_{ab}(E), produce\_context(O, I, E). \quad (5)$$

with $sh_{ab}/1$ being the *tabled predicate*. Rule (4) shows that the tabled abductive solution E of sh_{ab} is obtained from subgoal g given the input abductive context $[r]$. The tabled predicate sh_{ab} is used in rule (5), showing that its abductive solution E can be reused (since it is tabled) for a given *input abductive context* I of sh, resulting in the *output abductive context* (abductive solution) O, via the TABDUAL system predicate $produce\_context(O, I, E)$. This system predicate is responsible for producing a consistent abductive solution O from the input abductive context I and the tabled solution E. Note that in XSB, a tabled predicate has to be explicitly declared. For example, the directive below declares sh_{ab} as a tabled predicate.

$$:- table\ sh_{ab}/1.$$

Tabling this predicate causes both solutions $[r]$ and $[sp, r]$ of goal sh (cf. Example 2.3) being stored in a table.

4. Adding Answer Subsumption: TABDUAL$^+$

We start this section by describing an advanced XSB's tabling feature: *answer subsumption*. Then, the prototype TABDUAL$^+$ is detailed. Similar to its predecessor, TABDUAL$^+$ is also implemented using XSB Prolog.

4.1. Answer Subsumption

In XSB Prolog, tabling is by default performed using a technique called *answer variance*, i.e., an answer A of a goal is added to table T only if A is not a *variant*, i.e., identical

up to variable renaming, of some other answer already in T. For instance, if $p(A, B, A)$ is an answer to goal G added to table T, and G still has two other answers, e.g., $p(I, O, I)$ and $p(1, 2, 3)$, then only $p(1, 2, 3)$ is additionally added to T. In this case, $p(I, O, I)$ is not added as it is a variant of $p(A, B, A)$, which has already been tabled.

While answer variance is powerful enough to allow tabling to compute a program over Well-Founded Semantics, some other techniques to add an answer to a table is also provided in XSB Prolog. One of those techniques is called *partial order answer subsumption*. Using partial order answer subsumption, an answer A would be added to table T only if A is maximal compared to other answer in T according to a given partial order relation $>_o$. Moreover, if A is added, all answers that A subsumes are removed from T. We illustrate below the use of partial order answer subsumption.

Example 4.1. Suppose we are interested in counting the *minimum* number of edges in between two vertices in a unweighted graph. Predicate $d(X, Y, N)$ denotes that N is the number of edges between vertices X and Y. It is defined as follows:

$$d(P, Q, 1) \ :- \ e(P, Q).$$
$$d(P, R, N) \ :- \ d(P, Q, M), e(Q, R), N \ is \ M \ + \ 1.$$

where $e(P, Q)$ denotes that there is an edge from vertex P to Q.

As mentioned above, partial order answer subsumption keeps only those answers that are maximal according to a given partial order $>_o$. Since we are interested in the minimum number of edges, the relation $>_o$ is defined as: $d(P_1, Q_1, N_1) >_o d(P_2, Q_2, N_2)$ if $P_1 = P_2$, $Q_1 = Q_2$, and $N_1 < N_2$.

In XSB Prolog, tabling with partial order answer subsumption for the case illustrated in Example 4.1 is specified using the directive below:

$$:- \ table \ d(\_, \_, po((<)/2)).$$

This directive tells XSB that $d/3$ will be stored in the table using partial order answer subsumption (denoted by po) for its third argument with $</2$ as its partial order relation. Note that $</2$ is XSB's built-in predicate.

4.2. TABDUAL$^+$

TABDUAL$^+$ is a prototype that implements all four program transformations (cf. Section 3) plus answer subsumption. The employment of answer subsumption in TABDUAL$^+$ is important when one is interested in obtaining minimal explanations for an observation. It also helps avoid tabling too many and large explanations of a given observation, which may fail contextual abduction in practice, as it requires too many resources before being able to return a solution.

Figure 1 depicts the architecture of TABDUAL$^+$. TABDUAL$^+$ consists of two phases in general:

- The first phase is **program transformation**. Input program will be transformed into its corresponding output program based on the idea of program transformations in Section 3. These four transformations are accomplished using predicate $transform(F)$, where F is the file name of the input program. The output program resulting from this phase is needed for the next phase.
- The second phase is **abduction** itself. In this phase, abduction is performed by posing query Q to the output program using predicate $ask(Q)$ (i.e., with empty input abductive context), or using predicate $ask(Q, I)$ if the query is paired with a specific input abductive context I. When performing abduction, TABDUAL$^+$ interacts with its system predicates (e.g., $produce\_context/3$) and XSB's tabling mechanism to compute abductive solutions to the query given.

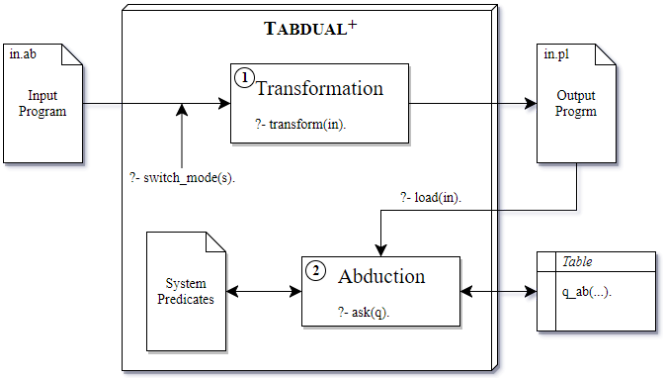

Figure 1. TABDUAL$^+$ Program Flow.

TABDUAL$^+$ also provides three modes of transformation for which users can flexibly opt. These three modes include *transformation without tabling*, *transformation with tabling*, and *transformation with tabling and answer subsumption*, coded into n, t, and s, respectively. Users can switch between modes using predicate $switch\_mode(M)$, where $M \in \{n, t, s\}$. The only difference among these modes is the directive that declares tabled predicates, which determines how these predicates are to be stored in a table. The required directives are automatically added in the program transformation phase, depending on the selected mode. We exemplify below required directives by considering predicate sh of our running example:

- Mode n: since tabling is not utilized, there is no tabling directive declared.
- Mode t: the directive below is required to declare that predicate $sh_{ab}/1$ is a tabled predicate:

$$:- \ table \ sh_{ab}/1.$$

- Mode s: $sh_{ab}/1$ is declared as a tabled predicate with partial order answer subsumption using the following directive:

$$:- \ table \ sh_{ab}(po(subset/2)).$$

In mode s, predicate $subset(X, Y)$ (i.e., X is a subset of Y) is defined as the partial order relation for delivering minimal abductive solutions stored in the table.

5. Experiments

We first present, in Example 5.1, several interactions in TABDUAL$^+$ to illustrate contextual abduction with and without answer subsumption.

Experiment 5.1. Recall the abductive framework $\langle P_1, \{r/0, sp/0\}, \emptyset\rangle$ from Example 2.2. Using TABDUAL$^+$, the following scenario is performed after issuing $switch\_mode(t)$ for the transformation phase:

1) To find explanations of 'the shoes are wet', query $ask(sh)$ is given. TABDUAL$^+$ produces two answers as expected, viz., $[r]$ ('it rained') as the first explanation, and $[sp, r]$ ('the sprinkler was on' and 'it rained') as its second one.
2) To find explanations of 'the shoes are wet' given that 'the sprinkler was off' (as its abductive context), query $ask(sh, [not\ sp])$ is given. Now, there is only one answer delivered by TABDUAL$^+$, viz., $[r]$. (cf. Example 3.1).

When the mode is altered to s (viz., $switch\_mode(s)$), TABDUAL$^+$ produces (and tables) only one answer $[r]$, as the minimal explanation, for query $ask(sh)$. Note, $[r]$ is a subset of the other solution $[sp, r]$. This is in line with our discussion of minimal explanation for this example in Section 1.

Alternatively to finding explanations by restricting to a specific abductive context (cf. Scenario 2 above), one may extend the abductive framework with integrity constraints. Suppose $\langle P_1, \{r/0, sp/0\}, \emptyset\rangle$ is now extended by adding an integrity constraint $\bot \leftarrow r, sp$, i.e., we consider the abductive framework $\langle P_1, \{r/0, sp/0\}, \{\bot \leftarrow r, sp\}\rangle$. Asking query $ask(sh)$ in TABDUAL$^+$, without answer subsumption, resulting in only one answer, viz., $[r]$.

The next experiments were conducted to evaluate TABDUAL$^+$'s scalability and efficiency, in terms of table space used and execution time, with respect to answer subsumption. In Experiment 5.2, a set of artificial cases are devised for evaluating TABDUAL$^+$'s scalability. On the other hand, in Experiment 5.3, the effectiveness and efficiency of TABDUAL$^+$ is assessed by a real world problem on cancer and chemoprevention.

Experiment 5.2. Given an integer n, defined $\mathcal{AB}_n$ as a set of n abducibles, $\{a_1, \ldots, a_n\}$. A *generator* G_n is built to produce a program P_n with the following specifications. For each $A \in pow(\mathcal{AB}_n)\backslash\{\emptyset\}$, the rule below is produced:

$$p \leftarrow seq(A).$$

where $pow(X)$ is a function that returns the powersets of a set X and $seq(A)$ returns a sequence of abducibles from the set A of abducibles. Using this generator, the abductive framework $\langle P_n, \mathcal{AB}_n, \emptyset\rangle$ is considered in this experiment.

TABLE 1. SCALABILITY WITH RESPECT TO ANSWER SUBSUMPTION

n	Total table space used (*byte*)	
	Without Answer Subsumption	With Answer Subsumption
1	78880	78816
2	80424	80304
3	81720	81280
4	84184	83024
5	91568	88952
6	101608	95808
7	120928	108872
8	158936	134032
9	230232	179864
10	385544	283480
11	680720	475192
12	1281160	868680
13	2481904	1655256
14	4883272	3228328
15	9685880	6374264
16	19299096	12673736

We run this experiment with various generators G_n, where $1 \leq n \leq 16$. The total table space used for query $ask(p)$, with and without answer subsumption, is reported in Table 1. In Table 1, the table space used by TABDUAL$^+$ is reduced significantly when answer subsumption is activated. Compared to its counterpart (without answer subsumption), the space used grows slower as n is increased, despite only a small amount of reduction in the beginning (small value of n). It thus indicates that answer subsumption results in better scalability compared to its counterpart. This is particularly true for the cases where a huge amount of space is required to store many and large explanations.

Experiment 5.3. This experiment concerns itself with an abduction problem on cancer and chemoprevention [11]. Therein, abduction is employed as a modeling approach to study genes that affect activation or inactivation of cancer cells, given the knowledge about cells that are active or inactive. Their interest is not on measuring the efficiency of abduction (as we do it here), but rather on showing the appropriateness of abduction in modeling such a problem.

The problem introduced in [11] is challenging as the knowledge base consists of a large number of facts and rules describing the influence between genes and cancer cells. The main query for TABDUAL$^+$ consists of eight subgoals:

$$active(phase0, aif), active(phase0, endo\_g),$$
$$inactive(phase0, caspase9), inactive(phase0, caspase6),$$
$$inactive(phase0, bcl2), inactive(phase0, caspase7),$$
$$inactive(phase0, akt), inactive(phase0, xiap).$$

Here, $active(Phase, Gene)$ and $inactive(Phase, Gene)$ denote that the gene $Gene$ is known to be active (inactive, respectively) in a particular experiment $Phase$. The solutions to the above query pertain to which drugs induced or inhibited, in an experiment, given the activation or inactivation of the specified genes.

In order to exercise the potential of tabling abductive soltions, this query is invoked gradually, starting from only invoking the first subgoal ($ask(active(phase0, aif))$), the first two subgoals (query $ask((active(phase0, aif), active(phase0, endo\_g)))$), and eventually all subgoals are invoked. The result of this experiment is reported in Table 2.

TABLE 2. EFFECTIVENESS AND EFFICIENCY WITH RESPECT TO ANSWER SUBSUMPTION

Number of subgoals	Execution time (*ms*)		Space usage (*byte*)	
	Without *Answer Sub.*	With *Answer Sub.*	Without *Answer Sub.*	With *Answer Sub.*
1	80	64	1545592	265544
2	52	36	2713648	276696
3	12	8	2752112	284232
4	*time out*	55004	*time out*	552600
5	-	8	-	564976
6	-	0	-	564976
7	-	2	-	568264
8	-	72	-	646808

The results shows that TABDUAL$^+$ in its answer subsumption mode successfully returns all (minimal) abductive solutions for all subgoals (and fewer ones, obviously). On the other, without answer subsumption, the computation stops after the first three subgoals, exceeding the alloted time limit (15 minutes). Interestingly, the execution time for executing the query with the first four subgoals reaches 55 seconds. This is expected, as the fourth subgoal ($inactive(phase0, caspase6)$) is known to be the hardest to solve.[1] Arguably, solving this fourth subgoal may be independent from that of the first three subgoals: solutions are newly computed and not tabled yet. Interestingly, the subsequent five to eight subgoals are solved almost instantly. Apparently, in this case recomputation of the solutions is avoided by benefiting from tabled solutions of previous subgoals' invocations. Furthermore, the space used by answer subsumption when computing the solutions of the first four (and more) subgoals does not significantly increase, with the exception of computing the solutions for the first four subgoals (for the reason mentioned above).

6. Conclusions and Future Work

We revisit tabling in contextual abduction that permits tabling of abductive solutions obtained in one context, so they can be reused in other abductive contexts. We improve this technique with an advanced tabling feature, viz., answer subsumption. As a proof of concept, a prototype TABDUAL$^+$ is implemented in XSB Prolog. TABDUAL$^+$ consists of two stages: program transformation and abduction. It also provides three different modes: no tabling, tabling without answer subsumption, and with answer subsumption. Our experiments show that answer subsumption successfully delivers minimal explanations, a criteria often applied in abduction, while promoting TABDUAL$^+$'s scalability. Moreover, it also helps in dealing with too many and large explanations for a given observation, a common situation in real world problems. Such a situation may hinder successful non-enhanced contextual abduction.

In future, it is interesting to explore another XSB's advanced feature, viz., *tabling with interned terms*, which supports a succinct representation of ground terms. Note, abducibles are typically assumed ground. This special representation, combined with tabling, helps to table and to retrieve answers more efficiently. Unfortunately, interned terms cannot be combined with answer subsumption, at least for the current XSB's version (3.7.x). Still, it is worth further exploring this feature, alternative to answer subsumption, and to study the extra efficiency gained from it.

1. Personal communication with one of the authors of [11].

Acknowledgements

S. M. A. Perkasa and A. Saptawijaya acknowledge the PITTA research grant 397/UN2.R3.1/HKP.05.00/2017 from Directorate Research and Community Services, Universitas Indonesia. L. M. Pereira acknowledges the support from Fundação para a Ciência e a Tecnologia (FCT/MEC) NOVA LINCS PEst UID/CEC/04516/2013.

References

[1] A. C. Kakas, R. A. Kowalski, and F. Toni, "Abductive logic programming," *Journal of Logic and Computation*, vol. 2, no. 6, pp. 719–770, 1992.

[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. Saptawijaya and L. M. Pereira, "Tabdual: a tabled abduction system for logic programs," *IfCoLog Journal of Logics and their Applications*, vol. 2, no. 1, pp. 69–123, 2015.

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

[5] J. de Castro and L. Pereira, "Abductive validation of a power-grid expert system diagnoser," *Innovations in Applied Artificial Intelligence*, pp. 838–847, 2004.

[6] 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.

[7] J. Gartner, T. Swift, A. Tien, C. V. Damásio, and L. M. Pereira, "Psychiatric diagnosis from the viewpoint of computational logic," in *Computational Logic (CL 2000)*, pp. 1362–1376, Springer Berlin Heidelberg, 2000.

[8] L. M. Pereira and A. Saptawijaya, *Programming Machine Ethics*, vol. 26. Springer, 2016.

[9] T. Swift and D. S. Warren, "XSB: Extending Prolog with Tabled Logic Programming," *Theory and Practice of Logic Programming*, vol. 12, no. 1-2, pp. 157–187, 2012.

[10] T. Swift and D. S. Warren, "Tabling with Answer Subsumption: Implementation, Applications and Performance," in *European Workshop on Logics in Artificial Intelligence*, pp. 300–312, Springer, 2010.

[11] S. Lazarou, A. Kakas, C. Neophytou, and A. Constantinou, "Logical Modeling of Cancer and Chemoprevention," *Learning and Discovery in Symbolic Systems Biology*, p. 36, 2012.

[12] 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. 04, pp. 383–428, 2004.