Jurnal Ilmu Komputer dan Informasi (Journal of a Science and Information). 12/1 (2019), 1-11
DOI: _http://dx.doi.org/10.21609/jiki.v12i1.569

TABLING WITH INTERNED TERMS ON CONTEXTUAL ABDUCTION

Muhammad Okky Ibrohim and Ari Saptawijaya*

Faculty of Computer Science, Universitas Indonesia, Kampus UI, Depok, 16424, Indonesia

Corresponding author. Email: saptawijaya@cs.ui.ac.id

Abstract

Abduction (also called abductive reasoning) is a form of logical inference which starts with an observation and is followed by finding the best explanations. In this paper, we improve the tabling in contextual abduction technique with an advanced tabling feature of XSB Prolog, namely *tabling with interned terms*. This feature enables us to store the abductive solutions as interned ground terms in a global area only once so that the use of table space to store abductive solutions becomes more efficient. We implemented this improvement to a prototype, called as TABDUAL$^{+\text{INT}}$. Although the experiment result shows that tabling with interned terms is relatively slower than tabling without interned terms when used to return first solutions from a subgoal, tabling with interned terms is relatively faster than tabling without interned terms when used to returns all solutions from a subgoal. Furthermore, tabling with interned terms is more efficient in table space used when performing abduction both in artificial and real world case, compared to tabling without interned terms.

Keywords: *abduction, logic programming, contextual abduction, tabling, interned terms*

Abstrak

Abduction (disebut juga *abductive reasoning*) adalah suatu bentuk inferensi logika yang digunakan untuk mencari penjelasan dari suatu observasi yang diberikan. Makalah ini membahas teknik lanjut dari *tabling* pada *contextual abduction* melalui pemanfaatan fitur *tabling* dengan *interned terms* pada XSB Prolog. Dalam hal ini, fitur *tabling* dengan *interned terms* digunakan untuk menyimpan *abductive solutions* sebagai suatu *interned ground terms* pada suatu area global, sedemikian sehingga penggunaan *table space* menjadi lebih efisien. Teknik lanjut dengan pemanfaatan fitur ini dikemas dalam suatu prototipe TABDUAL$^{+\text{INT}}$. Meskipun hasil eksperimen menunjukkan bahwa *tabling* dengan *interned terms* sedikit lebih lambat dari *tabling* tanpa *interned terms* dalam mencari solusi pertama dari suatu *subgoal*, *tabling* dengan *interned terms* relatif lebih cepat dari *tabling* tanpa *interned terms* ketika digunakan untuk mendapatkan semua solusi dari suatu *subgoal*. Lebih lanjut, *tabling* dengan *interned terms* lebih efisien dalam hal penggunaan *table space* saat digunakan untuk proses *abduction*, baik dalam permasalahan artifisial maupun riil jika dibandingkan dengan *tabling* tanpa *interned terms*.

Kata Kunci: *abduction, pemrograman logika, contextual abduction, tabling, interned terms*

1. Introduction

In logic programming, the study of abduction started in the late 80s as a new reasoning paradigm to address some of the limitations of deductive reasoning in classical logic. Abduction has already been well studied to resolve various problems in artificial intelligence and other areas of computer science. For example, abduction can resolve scheduling of maintenance [1], air-crew assignment [2], and system diagnoser problems [3].

Unlike the deductive reasoning, the premises in abduction do not guarantee the conclusion. Abduc-

tion is a form of logical inference used to find the best explanations for given observation.

Example 1. Consider the knowledge base as follow:

always present in class ***if*** *healthy* (1)

getting good grades ***if*** *study hard* (2)

getting good grades ***if*** *always present in class and be lucky* (3)

Suppose an observation of *getting good grades* is given, abduction will produces two explanation. The first explanation (E_1) is *study hard*, which obtained by statements (2), while the second explanation (E_2) is *healthy and be lucky*, which obtained by statements (1) then (3). Note that in abduction, an explanation must be a basic explanation (there is no statement further explaining the explanation). For example, *always present in class* are not considered as basic explanations. This is because *always present in class* can further be explained by statement (1). Furthermore, in abduction, the explanations commonly termed as abductive solutions.

In abduction, one often meets the case where abductive solutions obtained within one abductive context are also relevant and may be reused in a different context; this is called contextual abduction. In this case, the abductive solutions can be stored, so they may be reused in another context later. Recall Example 1, after explaining *getting good grades*, suppose the same observation with a context of hypothesizing that *do not study hard*. To find the explanations of this latter observation, the abductive solutions obtained by previous observations (which are E_1 and E_2) can be reused. In this case, the context of *do not study hard* will eliminate E_1 from possible explanations because there is contradiction about *study hard*. Therefore, for this new observation, abduction will produce an explanation by extending E_2 with the newly hypothesized context, which is *healthy, be lucky, and do not study hard*. Note that in contextual abduction, the context will be a part of the explanations.

In [4], Saptawijaya and Pereira proposed a technique called *tabling in contextual abduction*, whose detailed concept and implementation aspect wrapped into a system called TABDUAL, and implemented in XSB Prolog [5]. In general of logic programming (not only for abductive reasoning), tabling is a technique of reusing solutions of a goal. Here, tabling in contextual abduction is a technique of reusing abductive solutions obtained by previous observations in another abductive context.

The TABDUAL system was then improved by Perkasa, et al. [6], and wrapped into a system called TABDUAL$^+$. They improved the TABDUAL system by adding an advanced tabling feature of XSB Prolog, namely *answer subsumption* [7]. Their experiment result shows that tabling with answer subsumption is more efficient than normal tabling (tabling without answer subsumption) in table space used, both in artificial and real world case. Furthermore, tabling with answer subsumption successfully returns minimal explanations for every subgoal of real word case, while normal tabling cannot. Unfortunately, the tabling with answer subsumption cannot give all explanation from a given observation, since the answer subsumption is just storing the minimum solutions from a goal. Therefore, in those experiments, tabling with answer subsumption just gives a minimum explanation from a given observation.

Depending on problems, different requirements for explanations may apply. For some problems, showing only the existence of an explanation to an observation (or an action to satisfy a goal) is desired. In this case, finding a single solution suffices. In other cases, all explanations of an observation (e.g., in finding all possible causes of a disease) are required. Here, one prefers to enumerate all solutions. Moreover, while finding minimum explanations becomes a criterion in abduction for 'the best explanation', in general, other criteria may apply to satisfactoriness and plausibility of the explanations [8].

In this paper, we improved the TABDUAL$^+$ system by adding another advanced tabling feature of XSB Prolog, namely *tabling with interned terms*. Note that we do not extend it as a feature of XSB Prolog but we employ it instead for abduction, following up our own previous research results [6]. While the idea of tabling is similar to that of caching, tabling with interned terms emphasizes the representation of terms (an important data structure in Prolog) to improve the efficiency in storing them for their future retrieval. More precisely, tabling with interned terms is a tabling technique which supports a succinct representation of ground terms such that all interned terms are stored in a global area and each term is stored only once, with all instances of a given interned term (or subterm) pointing to that one stored representation [9]. Since each term can only be stored in a table once, tabling with interned term can be more efficient than normal tabling in table space used. Here, tabling with interned terms is relevant if used to store the abductive solutions for tabling in contextual abduction problem, since abducibles are typically assumed as ground terms. The improvement of TABDUAL$^+$ by adding tabling with interned terms then implemented in XSB Prolog and wrapped as TABDUAL$^{+\text{INT}}$.

To evaluate the benefit of tabling with interned terms, we use TABDUAL$^{+\text{INT}}$ to resolve an artificial and real world abduction problems. The experiment result shows that tabling with interned terms is slightly slower than normal tabling when used to return first solutions from a subgoal. However, when used to returns all solutions from a subgoal, tabling with interned terms is relatively faster than normal tabling. Furthermore, tabling with interned terms gives a more efficient in table space used compared with normal tabling, both in artificial and real world case.

In this paper, we discuss the logic program and abductive logic programming as a background for this research, presented in Section 2. We also discuss the technique of tabling in contextual abduction in Section 3. The detail of TABDUAL$^{+\text{INT}}$ is presented in Section 4. Section 5 will discuss the result experiments of TABDUAL$^{+\text{INT}}$. The paper concludes with future work, in Section 6.

2. Preliminaries

This section discusses the logic program and abductive logic programming, as a background for this paper.

2.1. Logic Program

A *logic program* is a set of sentences in logical form, expressing facts or rules about some problem domain. Suppose an alpabhet $\mathcal{A}$ from some language $\mathcal{L}$ given, where $\mathcal{A}$ denoting a disjoint countable set of contants, function symbols, predicate symbols, and also a set of variable symbols. Here, a variable conventionally is written as a capital letter. A term in $\mathcal{A}$ is defined recursively either as a variable, a constant, or an expression in the form of $f(t_1, \ldots, t_n)$, where f is a function symbol in $\mathcal{A}$ and t_i with $n \in \mathbb{N}$ are terms. Term that does not contain any free variables called as *ground term*. Next, an atom over $\mathcal{A}$ is defined as an expression in the form of $p(t_1, \ldots, t_n)$, where p is a predicate symbol in $\mathcal{A}$. The form of p/n is denoted as a predicate symbol p with n-arity.

Formally, a (normal) logic program is a countable set of rules in the form of $H \leftarrow L_1, \ldots, L_m$ where H is head of rule and $L_1, \ldots, L_m$ is a body of a rule. H is an atom; while L_i with $m \in \mathbb{N}$ are literals, either an atom a (called positive literal) or its negation $not\ a$ (called default literal). The comma in the rule is read as a conjunction. When the head of a rule is empty, the rule is called as *integrity constraint* (will be explained later in Section 2.2). Furthermore, in the logic program, a rule without a body is called as a fact, simply denoted with H. The example of a logic program can be seen in Example 2.

Example 2. Recall Example 1, the knowledge base in Example 1 can be represented in the logic program P as follow:

$always\_present \leftarrow healthy.$
$good\_grades \leftarrow study\_hard.$
$good\_grade \leftarrow always\_present, lucky.$

2.2. Abductive Logic Programming

Abduction in logic programming or *Abductive Logic Programming* (ALP) is an extension of logic programming to perform abductive reasoning [10]. ALP is used to solve a goal (an observation) by giving solutions (abductive solutions) in the form of a set of abductive hypothesis, namely *abducible*. An abducible is an atom a (named positive abducible) or its negation a^* (named negative abducible). Moreover, in ALP may contain specific rules to express restriction which must be fulfilled when performing abduction, called *integrity constraint*.

In ALP, the logic program, abducibles, and integrity constraints are wrapped into a triple (P, A, IC), called ALP theory or *abductive framework*, where P is logic program over $\mathcal{L}$ such that no rule in P whose head is an abducible, A is a set of abducible predicates with their corresponding arity, and IC is a set of integrity constraint. An integrity constraint in ALP is a rule in the form whose head is *false*, formally written as $\bot \leftarrow L_1, \ldots, L_m$, where $\bot$ denotes false. An example of abductive framework can be seen in Example 3.

Example 3. Recall Example 2, we can consider an abductive framework $(P, study\_hard/0, healthy/0, lucky/0, \emptyset)$ such that:

- P is the logic program in Example 2 that represents the knowledge base in Example 1;
- A is the set of abducible predicates consisting of $study\_hard, healthy$, and $lucky$, which their all arity is 0;
- The integrity constraint IC is $\emptyset$, which represents this framework has no restriction which must be fulfilled when performing abduction.

In ALP, an observation is analogous to query for a goal. Formally, the abduction phase for a query in ALP given in the following definition [1].

Definition 1. Given an abductive framework (P, A, IC), an abductive solution for a query Q is a set of abducibles $\Delta \subseteq A$ such that:

- $P \cup \Delta \models Q$,
- $P \cup \Delta \models IC$,

- $P \cup \Delta$ consistent.

Definition 1 is a generalization of normal logic program (both in terms of syntax and semantics), where $\models$ is an entailment relation. An abductive solutions Δ represents a set of statements which make Q hold. Here, an abductive solution Δ must consistent with statements (both rules and facts) in logic program P, that is no abductive solutions such that contradictory with rules and facts in logic program P.

Next, IC is a set of restrictions which make an abducible in A do not apply as the abductive solution. In this definition, $P \cup \Delta \models IC$ means that $P \cup \Delta \cup IC$ must consistent. For example, if we extend the abductive framework in Example 3 by give an integrity constraint: *its impossible to be healthy but rarely exercise*, denoted as $\bot \leftarrow healthy, rarely\_exercise$, and given the fact that $rarely\_exercise$ is true in P; so when we give a query $good\_grades$, the abduction phase will give one solution $[study\_hard]$ (referring to E_1 in Example 1). Here, $[healthy, lucky]$ (referring to E_2 in Example 1)) is not an abductive solution, since $healthy$ will make $P \cup \Delta \models IC$ is not consistent.

3. Tabling in Contextual Abduction

Abduction may benefit from the tabling technique, a feature offered in several Prolog systems, to store abductive solutions. Therefore, these solutions may be reused in another abductive context (viz., explanations that become the context in explaining an observation); thus avoiding unnecessary recomputation. This concept is known as *contextual abduction*.

For example, recall abductive framework in Example 3 and given a query $good\_grades$ which produce two abductive solutions $[study\_hard]$ and $[healthy,\ lucky]$ (respectively referring to E_1 and E_2 in Example 1), and then suppose the same query within an abductive context $[study\_hard^*]$ (which represents a context, where the student *do not study hard*). If we store the abductive solutions $[study\_hard]$ and $[healthy, lucky]$ from the previous goal, we can reuse those abductive solutions for the same query while also considering the additional abductive context $[study\_hard^*]$ as well. Since abductive solutions must be consistent with the logic program and integrity constraint, the abduction phase will produce the only $[healthy, lucky, study\_hard^*]$ for query $good\_grades$ within abductive context $[study\_hard^*]$. Here, $[study\_hard]$ cannot be an abductive solution because of inconsistency within the context $[study\_hard^*]$.

From this illustration, it can be seen that the benefit of tabling for storing the abductive solutions in abduction phase. But, in practice, an abductive solution from a goal Q cannot be stored directly in a table, since those solutions are related to the abductive context of Q. To solve this problem, Saptawijaya and Pereira [4] introduced a technique called *tabling in contextual abduction*.

Tabling in contextual abduction is a technique of reusing abductive solutions obtained by previous observations in another abductive context. This technique is implemented in XSB Prolog [5], wrapped in a prototype system called TABDUAL. The TABDUAL system consists of several program transformations, viz., *transformation for tabling abductive solutions, transformation for producing dualized negation, transformation for inserting abducibles,* and *transformation of a query*. In the subsequent paragraphs, we provide the motivation and a simple example for each transformation. For the fundamental theorem of TABDUAL transformation, including the soundness and completeness of TABDUAL, the reader is referred to [4].

Transformation for tabling abductive solutions is used to transform a predicate that will be stored in table. In this transformation, 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 solutions. The example of this transformation can be seen in Example 4.

Example 4. Recall P in Example 2, the rules:

$$good\_grades \leftarrow study\_hard.$$

$$good\_grades \leftarrow always\_present, lucky.$$

are transformed into three following rules:

$$good\_grades_{ab}([study\_hard]). \tag{4}$$

$$good\_grades_{ab}(E) \leftarrow always\_present([lucky], E). \tag{5}$$

$$good\_grades(I, O) \leftarrow good\_grades_{ab}(E), produce\_context(O, I, E). \tag{6}$$

where $good\_grades_{ab}/1$ is a tabled predicate. Rule (4) shows that the abductive solution E stored in table from predicate $good\_grades_{ab}/1$ is obtained from subgoal $study\_hard$ and also obtained from subgoal $always\_present$ with input context $[lucky]$ (rule (5)). Furthermore, rule (6), shows that the abductive solution E from predicate $good\_grades_{ab}/1$ can be reused (since stored in a table) with some *input abductive context* I such that produced *output abductive context* O via

system predicate $produce\_context(O, I, E)$. This system predicate is also checking the consistency when producing abductive solution O from the input abductive context I and the tabled solution E. When implemented in XSB, a tabled predicate must be explicitly declared. For example, the directive below declares $good\_grades_{ab}/1$ as a tabled predicate:

$$:- \ table\ good\_grades_{ab}/1.$$

Tabling this predicate will make the abductive solutions of goal $good\_grades$ (which represents *getting good grades* in Example 1), which are $[study\_hard]$ and $[healthy, lucky]$ (respectively reffering to E_1 and E_2 in Example 1) being stored in a table.

To deal with abduction under negative goals, TABDUAL transforms each rule for producing dualized negation, by implementing *dual program transformation* of ABDUAL [11]. The main purpose using dual program transformation (simply called *dual transformation*) is to get the solutions from a negative goal $not\ G$ without negated all abductive solutions of G. Dual transformation will produce two different rules (*layer*). *First layer dual rule* from a predicate p in logic program P is a rule not_p, which is defined to falsify p (which is $not\ p$), at once bringing the input context from every subgoal of p. The subgoals from first layer dual rule is the *second layer dual rule* p^{*i} which defined by falsifying the i^{th} body of predicate p in the logic program P. The example of this transformation can be seen in Example 5.

Example 5. Recall P in Example 2, the rules

$$good\_grades \leftarrow study\_hard.$$

$$good\_grades \leftarrow always\_present, lucky.$$

are transformed into two layers of dual rules as follow:

- First layer dual rule:

$$\begin{aligned} not\_good\_grades(T_0, T_2) \leftarrow \\ good\_grades^{*1}(T_0, T_1), \\ good\_grades^{*2}(T_1, T_2). \end{aligned} \tag{7}$$

- Second layer dual rule:

$$\begin{aligned} good\_grades^{*1}(I, O) \leftarrow \\ study\_hard^{*}(I, O). \end{aligned} \tag{8}$$

$$\begin{aligned} good\_grades^{*2}(I, O) \leftarrow \\ not\_always\_present(I, O). \end{aligned} \tag{9}$$

$$good\_grades^{*2}(I, O) \leftarrow lucky^{*}(I, O). \tag{10}$$

where T_i with $0 \leq i \leq 2$ is a fresh variable provided as an abductive context. TABDUAL implements dual rules transformation by need. In this case, the second layer dual rules not formed in transformation phase. Instead, they are formed in abduction phase when it is required for execution and then stored in a *trie* [12]. This is done to avoid too big a cost from transforming the rules which are actually not needed in abduction phase.

Further transformation in TABDUAL is a transformation for inserting abducibles. In this transformation, every abducible a in an abducible set A is transformed into a rule which can update the abductive context with the transformed abducible. Example 6 will illustrate this transformation.

Example 6. Recall abductive framework in Example 3, the abducible $study\_hard$ and $lucky$ is transformed to following rules:

$$\begin{aligned} study\_hard(I, O) \leftarrow \\ insert\_abducible(study\_hard, I, O). \end{aligned} \tag{11}$$

$$\begin{aligned} study\_hard^{*}(I, O) \leftarrow \\ insert\_abducible(study\_hard^{*}, I, O). \end{aligned} \tag{12}$$

$$\begin{aligned} lucky(I, O) \leftarrow \\ insert\_abducible(lucky, I, O). \end{aligned} \tag{13}$$

$$\begin{aligned} lucky*(I, O) \leftarrow \\ insert\_abducible(lucky^{*}, I, O). \end{aligned} \tag{14}$$

where $insert\_abducible(Ab, I, O)$ is TABDUAL system predicate used to add an abducible Ab into input abductive context I such that it produced output abductive context O. Furthermore, this predicate also preserve consistency of output abductive context O when the system added some abducible Ab.

As a consequence of the three transformation program which has been described previously, a query in abduction phase is also transformed by the TABDUAL system. Transformation of a query in TABDUAL is consists of positive goal transformation and negative goal transformation. Furthermore, the transformed query must satisfy all integrity constraints. This is done by adding a goal $not\_\bot/2$ that state dual rule from integrity constraint. Example 7 will illustrate this transformation.

Example 7. Recall abductive framework in Example 3, suppose $good\_grades$ with an empty input abductive context as query goal, the query is transformed into:

$$?- \ good\_grades([\,], T), not\_\bot(T, O)$$

4. Tabling with Interned Terms on TABDUAL$^+$

This section discusses *tabling with interned term*, an advanced tabling technique feature in XSB Prolog that allows us to store a term in succinct representation. This feature is used to improve the TABDUAL$^+$, implemented in XSB Prolog and wrapped in a prototype called as TABDUAL$^{+\text{INT}}$.

4.1. Tabling with Interned Terms

By default, tabling in XSB Prolog is performed by using variant tabling [12]. In XSB Prolog, there is an advanced tabling technique for variant tabling, namely *tabling with interned terms*. Since tabling with interned terms is performed by using variant tabling, this feature cannot be combined with answer subsumption tabling.

Interned terms is special representation for ground terms (the term which has no free variable in their argument), not only simple ones but may be structured (or complex) terms (viz., those built recursively from a function symbol). This representation is also sometimes known as *hash-consing* representation, since this representation using a hash for indexing the terms which tabled. Illustration of an interned ground term can be seen in Figure 1. In tabling with interned terms, all interned terms will be stored in a global area and each term is stored only once, with all instances of given interned subterms pointing to that one stored representation. This operation called *interning*, that allows for a more succinct representation of sets of ground terms including their subterms. Interning a ground term also makes that ground term not need to be copied into and out of tables, where this makes tabling with interned terms may save significant space compared to tabling without interned terms.

Based on Figure 1, it can be seen that an argument (subterm) $g/1$ of a term $f/2$ is stored in an array, and then their representation is stored in a hash table. Hash table consist of subterms which are indexed by term's arity. Since all terms (with their subterms) which stored as interned terms are indexed, this makes tabling with interned terms is more efficient when doing call and answering, compared to when we stored it as regular terms.

4.2. TABDUAL$^{+\text{INT}}$: TABDUAL$^+$ with Interned Terms

In 2017, Perkasa, et al. [6] proposed TABDUAL$^+$, a prototype that implements all four program transformation that has been described in Section 3 plus *answer subsumption*. Answer subsumption is an advanced XSB's tabling technique that allows users to store minimum solutions from a goal into a table. By means of a partial order relation R, partial order answer subsumption is achieved by adding an answer A into table T only if A is maximal compared to other answers in T according to a given partial order relation R. Furthermore, by adding A into table T, all answers that A subsumes are removed from T. Answer subsumption thus avoids too big a cost of table space due to storing too many solutions. The architecture of TABDUAL$^+$ can be seen in Figure 2.

As can be seen in Figure 2, TABDUAL$^+$ consists of two phases. The first phase is *program transformation*. In this phase, the input program will be transformed into its corresponding output based on TABDUAL program transformation (Section 3). Before users transform an input program, they can select the desired transformation mode. Here, TABDUAL$^+$ provides three transformation modes, which are *transformation without tabling, transformation with tabling,* and *transformation with tabling and answer subsumption,* which respectively coded into $n, t,$ and s. Users can switch between modes using predicate $switch\_mode(M)$, where $M \in \{n, t, s\}$. Afterward, to transform an input program F, users must use predicate $transform(F)$ such that resulting output program which needed for the next phase.

The second phase in TABDUAL$^+$ is *abduction* itself. To do an abduction phase, users can pose some query Q using predicate $ask(Q)$ (if users want to get abductive solutions of goal Q without any input abductive context) or using predicate $ask(Q, I)$ (if users want to get abductive solutions of goal within some input abductive context I). When performing abduction, TABDUAL$^+$ interacts with its system predicates and XSB's tabling mechanism to compute abductive solutions to the query given.

We then improved the TABDUAL$^+$ system by adding another advanced tabling feature, namely *tabling with interned terms*. This improvement is done to make an efficient tabling in contextual abduction in the term of table space used without losing any explanation from a goal, since tabling with interned terms allows us to store the all abductive solutions in a global area only once.

Tabling with interned terms is relevant to store abductive solutions for tabling in contextual abduction problem since abducibles are typically assumed as ground terms. To take advantage of tabling with interned terms for tabling in contextual abduction problem, a tabled abductive solution predicate must be declared as intern. For example, if we want to table an abductive solu-

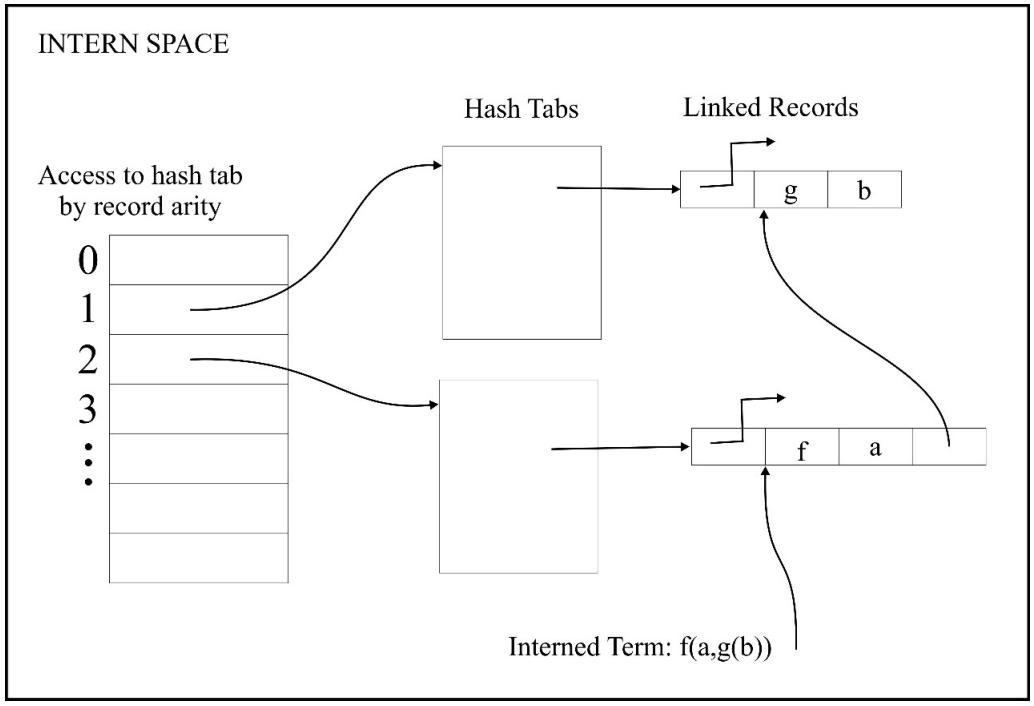

Figure 1. Storage of interned ground term $f(a, g(b))$ [9]

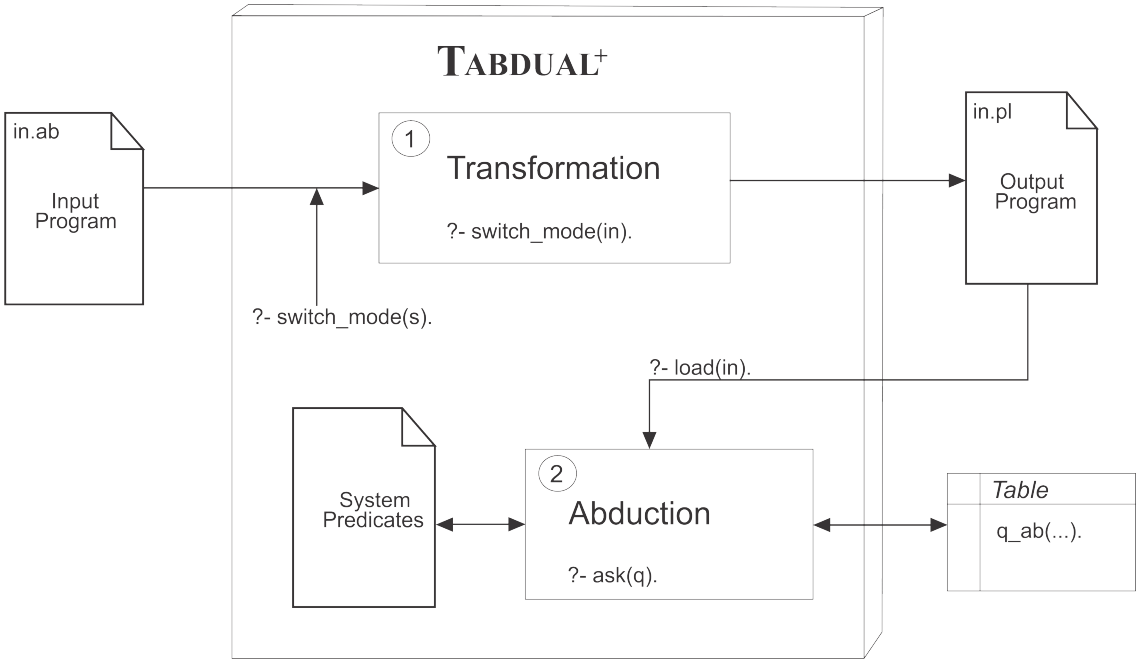

Figure 2. TABDUAL+ program flow [6]

tion predicate $good\_grades_{ab}/1$, we must declare $good\_grades_{ab}/1$ using directive:

$$:- \ table\ good\_grades_{ab}/1\ as\ intern.$$

If a tabled predicate declared as intern, when a call happens, all arguments are automatically interned before the call is looked up in the ta-

ble. Furthermore, while in return, every answer is interned before being added to the table. The TABDUAL$^+$ system that has been added with interned terms is also implemented in XSB Prolog, wrapped in a prototype system which called TABDUAL$^{+\text{INT}}$. This prototype system is available in _https://github.com/okkyibrohim/tabdual-plus-int.

Similarly with TABDUAL$^+$, TABDUAL$^{+\text{INT}}$ consists of two phases in performing abduction, viz., transformation phase and abduction phase. We build TABDUAL$^{+\text{INT}}$ simply by adding *transformation with tabling and interned terms* mode in TABDUAL$^+$, such that TABDUAL$^{+\text{INT}}$ provides four transformation modes, which are *transformation without tabling* (called 'no tabling'), *transformation with tabling, but without answer subsumption and interned terms* (called 'normal tabling'), *transformation with tabling and answer subsumption* (called 'tabling with answer subsumption'), and *transformation with tabling and interned terms* (called 'tabling with interned term'), which respectively coded into $n, t, s,$ and i. Therefore, now users can switch between modes using predicate $switch\_mode(M')$ where $M' \in \{n, t, s, i\}$. The thing that differentiates those four modes is the directive for declaring tabled abductive solutions predicates, which determines how the predicates are to be stored in a table. In transformation phase, the required directives are automatically added, based on the selected mode. Notice that for mode n, no tabling directive is required. Example 8 illustrates how the required directives are added for mode $t, s,$ and i.

Example 8. Consider an abductive solutions predicate $good\_grades_{ab}/1$ of our running example, the required directives below will be automatically added based on selected mode:

- mode *t*: the system automatically adds the directive below to declare that predicate $good\_grades_{ab}/1$ as a tabled predicate:
  $:- \ table\ good\_grades_{ab}/1.$
- mode *s*: the system declares $good\_grades_{ab}/1$ as a tabled predicate with *partial order answer subsumption* using predicate $subset/2$ as its partial order relation between abductive solutions by adding the directive below [6]:
  $:- \ table\ good\_grades_{ab}(po(subset/2)).$
- mode *i*: the system automatically adds the directive below to declare that intern ground term formed by the predicate $good\_grades_{ab}/1$:
  $:- \ table\ good\_grades_{ab}/1\ as\ intern.$

5. Experiments and Discussions

To evaluate the benefit of tabling with interned terms, we conduct two experiments using TABDUAL$^{+\text{INT}}$. Every experiment in this paper was done on a personal computer with Windows 10 64bit OS, Intel® Core i5 CPU @3.20 GHz, and 8 GB of memory. The XSB version used in this experiment is XSB v.3.7 (Clan MacGregor).

In both experiments, the efficiency of TABDUAL$^{+\text{INT}}$ is evaluated in terms of table space used and execution time. To evaluate the efficiency of TABDUAL$^{+\text{INT}}$ in the term of table space used, we used a set of an artificial case given in [6], presented in Experiment 1. Furthermore, we used a real world abduction problem on cancer and chemoprevention given in [13] to evaluate the efficiency of TABDUAL$^{+\text{INT}}$ in the term of table space used and execution time when performing abduction that contains a large knowledge base.

Experiment 1. Given A_n as a set of n abducibles $\{a_n, \ldots, a_n\}$ with $n \in \mathbb{Z}^+$. A *generator* G_n is introduced to provide an abductive framework $(P_n, A_n, \emptyset)$ where P_n is a logic program which contains rule $p \leftarrow seq(Ab)$ for each $Ab \in pow(Ab) \setminus \{\emptyset\}$. Here, $pow(A_n)$ refers to the power set of A_n, while $seq(Ab)$ is a functions that returns a sequence of abducibles from the abducible set Ab. For example, given A_2, the generator G_2 produces three rules in P_2 as follows:

$$p \leftarrow a_1.$$

$$p \leftarrow a_2.$$

$$p \leftarrow a_1, a_2.$$

Next, the transformation for tabling abductive solutions (cf. rules (4) and (6) of Example 4) transforms those three rules into the following rules, where $p_{ab}/1$ is a tabled predicate whose terms are interned:
$:- \ table\ p_{ab}/1\ as\ intern.$

$$p_{ab}([a_1]).$$

$$p_{ab}([a_2]).$$

$$p_{ab}([a_1, a_2]).$$

$$p(I, O) \leftarrow p_{ab}(E), produce\_context(O, I, E).$$

Note that the abductive solutions E in the tabled predicate p_{ab} (i.e., lists $[a_1]$, $[a_2]$, and $[a_1, a_2]$) are actually (ground) structured terms.

We run this experiment for various generators G_n for $1 \leq n \leq 14$ using query $ask(p)$, both in normal tabling mode and tabling with interned terms mode. This query is called once for every generator G_n, and collects every abductive solution O of $p([\ ], O)$ for each n. Without interning, the repetitive call of $p_{ab}(E)$ in collecting every abductive solution of p causes the sublist (every suffix of

the list) of every abductive solution to be copied into the table. With interning, only a pointer to an interned list (abductive solution) is copied into the table. By varying n of G_n in this experiment, by interning the tabled abductive solutions, we expect to see significant table space reduction as the value of n grows. The result of this experiment can be seen in Table 1. Furthermore, the plot between the value of n and the table space reduction can be seen in Figure 3.

TABLE 1
COMPARISON OF TABLE SPACE USED BETWEEN NORMAL TABLING AND TABLING WITH INTERNED TERMS IN ARTIFICIAL CASE

n	Total table space used (*byte*)		Reduction (%)
	Normal Tabling	Tabling with Interned Term	
1	48312	48272	0.083
2	49060	48960	0.203
3	50164	49994	0.338
4	52012	51808	0.392
5	55220	54568	1.181
6	61308	59696	2.629
7	73092	69560	4.832
8	96268	88896	7.658
9	142556	127696	10.424
10	234348	206976	11.680
11	413988	359368	13.194
12	779740	670768	13.975
13	1511236	1293504	14.408
14	2974172	2538912	14.635

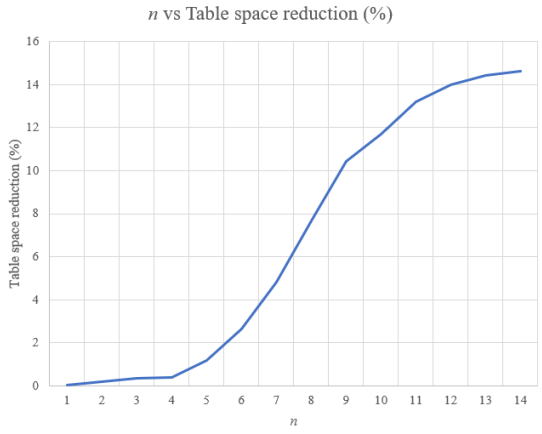

Figure 3. Plot between the value of n and the table space reduction

From Table 1, it can be seen that tabling with interned terms (in this case, for predicate p_{ab}/1) can reduce the table space used for all various generators G_n. This indicates that tabling with interned terms is more efficient in table space used compared to normal tabling. This can be particularly true in another case where a huge amount of space is required to store many and large explanations.

Experiment 2. We are interested in using TABDUAL$^{+\text{INT}}$ in real world abduction problem. The problem in this experiment is about cancer and chemoprevention, introduced in [13]. This abduction problem is employed as modeling approach to study genes that affect the activation or inactivation of cancer cells, given the logic program contains a knowledge base about inactive or inactive cells.

This problem is challenging as the knowledge base describing the influence between genes and cancer cells were consists of a large number of facts and rules. Here, the main query for TABDUAL$^{+\text{INT}}$ consists of eight subgoals, which are:

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

where $active(Phase, Gene)$ and $inactive(Phase, Gene)$ respectively denote that the gene $Gene$ is known to be active and inactive for an experiment $Phase$. The solutions for all query related to whether a drug is induced or inhibited in an experiment if given the activation or inactivation of the specified genes.

All query given in this experiment will be invoked gradually. This is done to know the potential of tabling abductive solutions. We start from only invoking the first subgoal which is $ask(active(phase0, aif))$ and then first two subgoals which is $ask((active(phase0, aif), active(phase0, endo\_g))$, and finally all subgoals are invoked. The result of this experiment can be seen in Table 2.

TABLE 2
COMPARISON OF TABLE SPACE USED BETWEEN NORMAL TABLING AND TABLING WITH INTERNED TERMS IN REAL CASE

Number of subgoals	Execution time (*seconds*)		Space usage (*byte*)	
	Normal Tabling	Tabling with Interned Terms	Normal Tabling	Tabling with Interned Terms
1	3.72	4.64	28404932	11075412
2	3.66	4.58	55576452	21786592
3	0.84	1.03	55787492	21823824
4–8	*time out*	*time out*	*time out*	*time out*

Based on Table 2, we can see that tabling with interned terms can significantly reduce the table space used when performing contextual abduction in this real world abduction problem. Tabling with interned terms can reduce the table space used up to 61%. Unfortunately, both normal tabling and tabling

with interned terms cannot compute the solutions of abduction goals after first three goals because of *time out* (in this experiment we set 20 minutes for the limit of execution time). This is because the fourth subgoal $(inactive(phase0, caspase6))$ is known to be the hardest to solve [6]: solving this fourth subgoal may be independent of the first three subgoals (solutions are newly computed and not tabled yet). Notice that the execution time in this experiment refers to getting the first explanation from a subgoal. This experiment result also shows that the execution time when using tabling with interned terms is relatively slower than using normal tabling (about 20%). This is because tabling with interned terms must intern a term before storing it in a table, such that the tabling with interned terms requires more time compared to when using normal tabling. Moreover, if we are just interested to get the first explanation from a subgoal, both tabling modes will do more tabling than reuse it. Therefore, the special representation for ground terms in tabling with interned terms is not beneficial in this case.

However, if we compare normal tabling and tabling with interned terms in checking (only checking, not returning) *all* abductive solutions of a subgoal using TABDUAL^{+int} system predicate $checksol(G)$ where G is the first subgoal in this experiment, i.e., $checksol(active\ \ (phase0, aif))$; tabling with interned terms is faster than normal tabling (22.58 *seconds* when using tabling with interned terms and 26.77 *seconds* when using normal tabling). As it checks for all abductive solutions, the repetitive calls to the tabled predicate benefit from reusing solutions in the table. Being tabled as interned terms, the solutions are indexed and these solutions interact with the indexed lookup of calls (and answers) in the table. This is not the case for normal tabling (without interning): the terms that form the solutions have to be completely traversed. Therefore, tabling with interned terms can return all tabled abductive solutions of a goal faster than normal tabling. Unfortunately, both normal tabling and tabling with interned terms can only check all explanations for the first subgoal, but cannot check all solutions for more than one subgoal because of memory overload (shortage of space).

6. Conclusions and Future Works

Tabling in contextual abduction is a technique that allows us to store an abductive context obtained in one context such that they can be reused in different relevant context. This technique is improved by Perkasa et al. [6] by adding the answer subsumption tabling feature and wrapped it in a system called TABDUAL^{+}, implemented in XSB Prolog. In this paper, we improved the TABDUAL^{+} by adding another advanced tabling feature in XSB, which is interned terms, in order to make an efficient tabling in contextual abduction without losing any explanation from a goal. This improvement is also implemented in XSB Prolog and wrapped in a system called TABDUAL$^{+\text{INT}}$. TABDUAL$^{+\text{INT}}$ consists two phases viz., program transformation and abduction, and provides four different modes which are no tabling, normal tabling (tabling without answer subsumption and interned term), tabling with answer subsumption, and tabling with interned terms.

All experiments in this paper show that tabling with interned terms is more efficient than normal tabling in table space used. Both Experiment 1 and Experiment 2 shows that tabling with interned terms can reduce the table space used compared to normal tabling. Experiment 2 shows that tabling with interned terms is relatively faster than normal tabling when used to check all explanations of a subgoal. Unfortunately, when it is used to solve a real world abduction problem (Experiment 2) which have too many explanations from some subgoals, both normal tabling and tabling with interned terms failed to solve all query goals given because of out of memory.

For future works, there are several ways that may solve this problem. The implementation of systems predicates of TABDUAL$^{+\text{INT}}$ such as $produce\_context/3$ and $insert\_abducible/3$ can be improved. When $produce\_context/3$ maintains the consistency between the input abductive context I and the abductive solutions entry E, the system checks each element of abductive solutions E one by one against the context I. If the elements of abductive solutions E is split into two lists (positive and negative abductive solutions list) and they are sorted (base on their predicate name), the consistency checking process will be more efficient. This can also be applied to the system predicate $insert\_abducible/3$, by splitting the abducible list in the input abductive context I into positive and negative abducible lists.

In TABDUAL$^{+\text{INT}}$, the dual transformation is performed by need. The application of the dual transformation by need in TABDUAL$^{+\text{INT}}$ indirectly increases cost in the abduction phase when for solving a negative goal, both in execution time and table space used (since the dual transformation by need requires TABDUAL$^{+\text{INT}}$ storing dual rules in a trie). It is interesting to perform the dual transformation with schema once for all during the transformation phase so that the abduction phase becomes faster. On the other hand, this schema may increase the execution

time significantly in the transformation phase for a large knowledge base (consisting huge number of rules).

Furthermore, to avoid the space shortage problem, more research needs to be done in selecting any predicate whose abductive solutions to be stored in the table automatically, in such a way that the table space used can be reduced. This can be achieved by carefully considering the abduction problem, where this requires the expertise of the knowledge engineer. Such a careful consideration from a domain expert is also important in addressing the cases of contextual abduction that may benefit from tabling with interned terms in the long run.

Acknowledgements

The authors acknowledge Antonis C. Kakas in providing the real world case of chemoprevention for this paper. Muhammad Okky Ibrohim acknowledges to Syukri M. A. Perkasa for his help in dealing with implementation issues.

References

[1] M. Denecker and A. C. Kakas, "Abduction in logic programming," in *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part I*. London, UK, UK: Springer-Verlag, 2002, pp. 402–436.

[2] A. C. Kakas and A. Michael, "An abductive-based scheduler for air-crew assignment," *Journal of Applied Artificial Intelligence*, vol. 15, pp. 333–360, 2001.

[3] J. F. Castro and L. M. Pereira, "Abductive validation of a power-grid expert system diagnose," in *Innovations in Applied Artificial Intelligence IEA/AIE*, vol. 3029, 2004, pp. 838–847.

[4] A. Saptawijaya and L. M. Pereira, "TABDUAL: a tabled abduction system for logic programs," *IfCoLog Journal of Logics and their Applications*, vol. 2(1), pp. 69–123, 2015.

[5] T. Swift and D. S. Warren, "XSB: Extending prolog with tabled logic programming," *Theory and Practice of Logic Programming*, vol. 12(1-2), pp. 157–187, 2012.

[6] S. M. A. Perkasa, A. Saptawijaya, and L. M. Pereira, "Tabling in contextual abduction with answer subsumption," in *Proceedings 9th International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, 2017, pp. 459–464.

[7] T. Swift and D. S. Warren, "Tabling with answer subsumption: Implementation, applications and performance," in *European Workshop on Logics in Artificial Intelligence JELIA*, vol. 6341, 2010, pp. 300–312.

[8] I. Douven, "Abduction," in *The Stanford Encyclopedia of Philosophy*, summer 2017 ed., E. N. Zalta, Ed. Metaphysics Research Lab, Stanford University, 2017.

[9] D. S. Warren, "Interning ground terms in XSB," in *Colloquium on Implementation of Constraint and Logic Programming Systems (CICLOPS 2013)*, 2013.

[10] A. C. Kakas, R. A. Kowalski, and F. Toni, "Abductive logic programming," *Journal of Logic and Computation*, vol. 2(6), pp. 719–770, 1993.

[11] 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(4), pp. 383–428, 2004.

[12] T. Swift, D. S. Warren, K. Sagonas, J. Freire, P. Rao, B. Cui, E. Johnson, L. Castro, R. F. Marques, D. Saha, S. Dawson, and M. Kifer, *The XSB System Version 3.8x Volume 1: Programmers Manual*, 2017. [Online]. Available: _http://xsb.sourceforge.net/manual1/manual1.pdf

[13] S. Lazarou, A. Kakas, C. Neophytou, and A. Constantinou, "Logical modeling of cancer and chemoprevention," in *Workshop on Learning and Discovery in Symbolic Systems Biology (LDSSB 2012)*, 2012, pp. 36–54.