Tabling Optimization for Contextual Abduction

Ridhwan Dewoprabowo   Ari Saptawijaya
Faculty of Computer Science
Universitas Indonesia
Depok, Indonesia
ridhwan.dewoprabowo@ui.ac.id   saptawijaya@cs.ui.ac.id

Tabling for contextual abduction in logic programming has been introduced as a means to store previously obtained abductive solutions in one context to be reused in another context. This paper identifies a number of issues in the existing implementations of tabling in contextual abduction and aims to mitigate the issues. We propose a new program transformation for integrity constraints to deal with their proper application for filtering solutions while also reducing the table memory usage. We further optimize the table memory usage by selectively picking predicates to table and by pragmatically simplifying the representation of the problem. The evaluation of our proposed approach, on both artificial and real world problems, shows that they improve the scalability of tabled abduction compared to previous implementations.

1 Introduction

The requirement for artificial intelligence (AI) to provide explanations in making critical decision becomes increasingly important due to concerns of accountability, trust, as well as ethics. Such an explainable AI is expected to be capable of providing justifications that are human-understandable. A form of reasoning for providing explanations to an observation, known as *abduction*, has been well studied in AI, particularly in knowledge representation and reasoning. It extends to logic programming, dubbed *abductive logic programming* [3], and it has a wide variety of usage, e.g., in planning, scheduling, reasoning of rational agents, security protocols verification, biological systems, and machine ethics. [1, 4, 6, 7, 8, 10].

Finding best explanations in abduction can be very costly, especially when the knowledge base is large and complex. In some cases, abductive solutions already obtained within one abductive context may be relevant to other contexts and thus, can be reused in those contexts. This idea of storing and reusing abductive solutions across different contexts, designated as *contextual abduction*, has been recently brought into abductive logic programming [13]. It benefits from tabling mechanism in logic programming [14], which is supported to different extents in a number of Prolog systems. Tabling for contextual abduction, TABDUAL, is implemented on top of XSB Prolog [15], and is realized by a transformation from abductive logic programs into tabled logic programs. The transformation makes use of the dual program transformation [2] to deal with abduction under negative goals.

While several implementation aspects have been considered in [13], TABDUAL may still suffer from excessive computational cost, particularly in terms of space due to tabling itself. This is especially true for an observation where many and large alternative explanations are found and naively tabled. Consequently, it may hamper contextual abduction to be completed, as it requires too many resources before being able to return a solution. To address this scalability issue, various features of tabling mechanism have been employed. In [11], TABDUAL is extended with answer subsumption [16] to store only subsuming abductive solutions and delivers to that effect only subset-minimal explanations to an observation. Using a real world case of chemoprevention [8] as a benchmark, answer subsumption improves TABDUAL in computing (minimal) abductive solutions for a query with more goals, as the number of solutions

F. Ricca, A. Russo et al. (Eds.): Proc. 36th International Conference
on Logic Programming (Technical Communications) 2020 (ICLP 2020)
EPTCS 325, 2020, pp. 137–150, doi:10.4204/EPTCS.325.20

is reduced. Nevertheless, scalability remains a challenge for harder queries (i.e., queries built from many goals) of the same benchmark. A way around this challenge is to call queries incrementally with respect to the number of their goals, making the most of tabled solutions from simpler queries (with less goals) and reusing them to answer harder ones.

In another exploration [5], a more recent tabling feature in XSB Prolog, viz., tabling with interned ground terms [19], is exploited to improve the scalability of TABDUAL. As abductive solutions are assumed to be ground, they can benefit from the succinct representation of interned ground terms and, at the same time, they are tabled for future reuse in other contexts. Unfortunately, in XSB Prolog, tabling with interned ground terms makes use of variant tabling, and thus cannot be combined with answer subsumption. It can therefore complement the variant of TABDUAL that employs answer subsumption, viz., to provide all solutions rather than only minimal ones. Indeed, experimenting with the same benchmark demonstrates that, when it is compared with the one without interning operation, it is superior in the case of returning all solutions, though it takes longer time to return the first solution. Nevertheless, this variant of TABDUAL still struggles in solving the hard queries of the same benchmark.

With the motivation of improving the scalability of TABDUAL, furthering the existing implementations that have exploited various tabling features, in this study we focus on the program transformation of TABDUAL. We look into the abductive solutions produced by the transformation and determine whether or not these solutions should be tabled. In so doing, we identify that integrity constraints should be handled carefully in order for TABDUAL to correctly get rid of unwanted solutions but without rejecting valid solutions. The structure of rules is also taken into account in deciding which predicates to table.

The contributions of the paper are therefore as follows. First, we formulate a new transformation for integrity constraints to warrant their correct application in filtering solutions. This new transformation no longer depends on the dual program transformation, but instead it utilizes subset checking as a way to ensure that abductive solutions adheres to the integrity constraints. Second, we introduce a mechanism to reduce the number of tabled solutions. This is realized by selectively choosing predicates to table, e.g., by inspecting the occurrence of abducibles in the body of rules. We design an artificial problem, whose size is parameterized by the number and the size of rules, to study the table memory usage of the proposed approach. The results show that, compared to the implementation where the present approach is not in place, the table memory usage is reduced as the size of the problem increases. However, as an implication, the number of inferences may also increase. There is therefore a trade-off between the table memory usage and the number of inferences, as expected. In the end, the scalability of TABDUAL with the present approach is evaluated on the same real world case as in the previous implementations. Together with answer subsumption, the new transformation for integrity constraints and the selection of tabled predicates successfully reduce the table memory usage compared to the implementation that only employs answer subsumption. Consequently, compared to the previous implementation with answer subsumption [11], the present approach manages to solve queries with more goals, even when queries are not called incrementally. We also exploit the representation of the problem and simplify the representation to further reduce the table memory usage.

The rest of the paper is organized as follows. We set up notations and recap contextual abduction in logic programming, in Section 2. The new transformation for integrity constraints and its empirical analysis are explained in Section 3. Section 4 discusses our proposed method for optimizing table memory usage by reducing tabled predicates and its evaluation on artificial and real world cases. We conclude, in Section 5, with future work. The implementation code for our improvement of TABDUAL is available on `https://github.com/RidhwanD/TabdualSC`.

2 Contextual Abduction in Logic Programming

A *logic program* $\mathcal{P}$ is a countable set of rules in the form $H \leftarrow L_1, \ldots, L_m$, $m \geq 0$ where L_i, $1 \leq i \leq m$ are literals (atoms or their negation). Ground terms (literals, rules, and programs) are defined as usual, without variables.

Abduction is a form of reasoning that aims at finding plausible explanations to a given observation. In logic programming, abduction is implemented by extending the logic program with abductive hypotheses or *abducibles* as candidates of explanation to an observation; the latter is given as a query. We recap below abductive reasoning in well-founded semantics [17] and closely follow the definitions in [2]. We use $\perp$ and U to denote *false* and *undefined*, respectively. In abductive reasoning, *integrity constraints* (ICs) are rules in the form of denial $\perp \leftarrow L_1, \ldots, L_m$. We define $\mathcal{AB}$ as the set of abducible predicates, i.e., predicates without rules. We define $\mathcal{AB}_g$ as the set containing ground abducible literals, i.e., for each $ab \in \mathcal{AB}$, we have that $ab(\bar{t})$, $not\ ab(\bar{t}) \in \mathcal{AB}_g$ for all $\bar{t}$ tuples of ground terms. We say that $not\ ab(\bar{t})$ is the complement of $ab(\bar{t})$ and vice versa.

Definition 2.1 (Abductive Framework) *An abductive framework F is a triple* $\langle \mathcal{P}, \mathcal{AB}, \mathcal{IC} \rangle$ *where* $\mathcal{P}$ *is a logic program such that no rule in* $\mathcal{P}$ *has an abducible from* $\mathcal{AB}$ *as its head, and* $\mathcal{IC}$ *is a set of integrity constraints.*

Definition 2.2 (Abductive Solution) *Given an abductive framework* $F = \langle \mathcal{P}, \mathcal{AB}, \mathcal{IC} \rangle$*, the set of abducibles* $\mathcal{S} \subset \mathcal{AB}_g$ *is an abductive solution for F if* $\perp$ *is false in well-founded model* $M_\mathcal{S}$ *of* $(\mathcal{P} \cup \mathcal{P}_\mathcal{S} \cup \mathcal{IC})$ *where* $\mathcal{P}_\mathcal{S}$ *is the smallest set of rules containing the fact A if* $A \in \mathcal{S}$ *and, for all* $A' \in \mathcal{AB}_g \backslash \mathcal{S}$ *and* A' *is not the complement of A, we have that* $A' \leftarrow$ *U. A set* $\mathcal{S}$ *is an abductive solution for query Q if* $M_\mathcal{S} \models Q$.

Example 2.1 *Given the abductive framework* $\langle \mathcal{P}, \mathcal{AB}, \mathcal{IC} \rangle$ *where* $\mathcal{AB} = \{q, r, t\}$ *and* $\mathcal{P}$ *is the program:*

$$p(X) \leftarrow q(0), q(1), s(X). \qquad s(X) \leftarrow not\ t(X). \qquad u(X) \leftarrow not\ p(X).$$

and $\mathcal{IC} = \{\perp \leftarrow q(X), r(X), \perp \leftarrow u(X)\}$. *Given a query* $Q = \{p(0)\}$*, its abductive solution is* $\mathcal{S} = \{q(0), q(1), not\ t(0), not\ r(0), not\ r(1)\}$.
Note that $\{q(0), q(1), not\ t(0), not\ r(0), not\ r(1), s(0), p(0), not\ u(0)\} \subset M_\mathcal{S}$ *and thus* $M_\mathcal{S} \models Q$.

Contextual abduction in TABDUAL [13] implements tabling mechanism for abduction in well-founded semantics to table abductive solutions obtained within one abductive context to be reused in other relevant contexts. Contexts can be the results obtained from previous queries or subgoals. Calling $s(0)$ before the query Q in Example 2.1 allows us to reuse its solution, i.e., $not\ t(0)$, in the execution of Q. Context can also be seen as a way to restrict the solutions by determining the initial hypothesis of a query. In Example 2.1, calling the query $p(0)$ with context $r(0)$, i.e., $r(0)$ is assumed true when we call the query, may reuse the already computed solution obtained from query $p(0)$ without context, i.e., $\{q(0), q(1), not\ t(0), not\ r(0), not\ r(1)\}$ by adding the context $r(0)$ into it. However, since $not\ r(0)$ is present in the solution due to the first IC, the query $p(0)$ with context $r(0)$ has no solution. Tabled abduction in TABDUAL involves a program transformation, that transforms abductive logic program to tabled logic program, and the abduction stage on top of the tabled logic program. The transformation introduces two extra arguments, for all predicates in the program, to serve as the input and output contexts. The program transformation process of TABDUAL includes:

1. Transformation for tabling abductive solution that accommodates storing and reusing abductive solutions by utilizing XSB Prologs tabling mechanism. For every rule r, we define the rule r_{ab} as its tabled predicate and moving the abducibles in its body as its input context. For each predicate p, we define a rule that permits reusing tabled abductive solutions in p_{ab} consistently in another context.

Example 2.2 *Given program $\mathcal{P}$ in Example 2.1. The results of transforming $p/1$ are as follow:*

- $p_{ab}(X,O) \leftarrow s(X,[q(0),q(1)],O)$.
- $p(X,I,O) \leftarrow p_{ab}(X,E), produce_context(O,I,E)$.

The transformation for $s/1$ and $u/1$ are defined similarly.

Here, $produce_context(O,I,E)$ is a TABDUAL system predicate that consistently produces output O from the input context I and a tabled solution E. It checks whether each abducible in E or its negation is already present in I. If that is not the case, then the abducible is added into the context. O is the resulting output context from this predicate. Otherwise, if the negation is present, then E and I are inconsistent and the predicate fails.

2. Transformation for generating dualized negation that enables TABDUAL to deal with abduction under negative goals. It employs the dual program transformation from ABDUAL [2] to deal with negative goals as positive literals. For every predicate $p(\bar{X})$, we define the rule $not_p(\bar{X},T_0,T_1)$ with $p^{*i}(\bar{X},T_{i-1},T_i), 1 \leq i \leq n$ as its body, where n is the number of rules with head p. Each $p^{*i}(\bar{X},T_{i-1},T_i)$ represents a rule that falsifies the original i-th rule of p.

Example 2.3 *Given program $\mathcal{P}$ in Example 2.1, the results of transformation for $p/1$ and the ICs are defined below. The transformation for $s/1$ and $u/1$ are defined similarly.*

- $p^{*1}(X,I,O) \leftarrow not_q(0,I,O)$.
- $p^{*1}(X,I,O) \leftarrow q(0,I,T), not_q(1,T,O)$.
- $p^{*1}(X,I,O) \leftarrow q(0,I,T), q(1,T,S), not_s(X,S,O)$.
- $not_p(X,I,O) \leftarrow p^{*1}(X,I,O)$.
- $false^{*1}(I,O) \leftarrow not_q(X,I,O)$.
- $false^{*1}(I,O) \leftarrow q(X,I,T), not_r(X,T,O)$.
- $false^{*2}(I,O) \leftarrow not_u(X,I,O)$.
- $not_false(I,O) \leftarrow false^{*1}(I,O), false^{*2}(I,O)$.

3. Transformation for inserting abducibles into context while also maintaining the consistency of the abductive context when an abducible is abduced. Each abducible $a(\bar{X})$ transforms into two rules, $a(\bar{X},I,O)$ and $not_a(\bar{X},I,O)$.

Example 2.4 *Given the set $\mathcal{AB}$ from Example 2.1. The transformation result for q are as follow:*

- $q(X,I,O) \leftarrow insert_abducible(q(X),I,O)$.
- $not_q(X,I,O) \leftarrow insert_abducible(not\ q(X),I,O)$.

The TABDUAL *system predicate $insert_abducible(A,I,O)$ non-redundantly adds abducible A into context I to obtain consistent context O. The transformation for abducibles r and t are defined similarly.*

4. Query transformation that ensures any abductive solution satisfies the ICs. Each query is transformed by adding input and output contexts to each subgoal. To warrant that ICs are not violated, $not_false(I,O)$ is added at the end. The output context O therefore serves as an abductive solution to the query, by checking whether I satisfies the ICs.

Example 2.5 *The query Q in Example 2.1 transforms into $?-p(0,[\,],T), not_false(T,O)$ and returns $O = [q(0),q(1),not\ t(0),not\ r(0)]$ as a solution. Calling the query Q with an input context $r(0)$ amounts to calling $?-p(0,[r(0)],T), not_false(T,O)$. Invoking subgoal $p(0,[r(0)],T)$ results in $T = [r(0),q(0),q(1),not\ t(0)]$. Invoking the second subgoal $not_false([r(0),q(0),q(1),not\ t(0)],O)$ fails as T violates the first IC.*

3 Transforming Integrity Constraints

In the original TABDUAL, ICs are maintained by first transforming them into their dual rule and conjoining it with the given query, as illustrated above. We highlight two problems concerning the dual program transformation for ICs. Firstly, the resulting $not_false(I,O)$ adds more abducibles to I for the abductive solution O to satisfy ICs under well-founded semantics (according to Definition 2.2). As a result, it may lead to a large number of generated abductive solutions, since an IC with multiple literals in the body produces several rules after transformation. Secondly, it may cause the abduction process to return incorrect solutions. For example, using the abductive framework in Example 2.1, the query $p(0)$ with the input context $r(1)$ returns the solution $[q(0), q(1), not\ t(0), not\ r(0), r(1)]$, despite violating the ICs. It happens since the call of $false^{*1}/2$ in line 6 of Example 2.3 attempts to abduce $q(X)$ and $not\ r(X)$ using the rules in Example 2.4. However, when attempting to abduce $q(X)$, $insert_abducible/3$ already unifies $q(X)$ with the atom $q(0)$ in the solution, and additionally $not\ r(0)$ is abduced. Consequently, $q(X)$ never unifies with $q(1)$ and the consistency checking for $X = 1$ is never performed. Thus, it returns the incorrect solution, where $q(1)$ and $r(1)$ belong to this solution.

The dual transformation of ICs may also risk rejection of correct solutions. Suppose that in Example 2.1, the rule $p/1$ is slightly modified into $p(X) \leftarrow q(0), not\ q(1), s(X)$. Given the query $Q = \{p(0)\}$, abduction fails even though $S = [q(0), not\ q(1), not\ t(0)]$ is actually a solution that does not violate the IC. In this case, it fails since calling $false^{*1}/2$ in line 5 of Example 2.3 attempts to abduce $not\ q(X)$. The $insert_abducible/3$ predicate in the rule on line 2 of Example 2.4, for the sake of consistency, checks whether the complement of $not\ q(X)$, viz., $q(X)$, exists in S. Unfortunately, the variable X in $q(X)$ is already unified to 0, and consequently it fails. Alternatively, calling $false^{*1}/2$ in line 6 of Example 2.3 also fails as it attempts to abduce $q(X)$ for similar reason. In this case, X is already unified with 1 when it checks its complement $not\ q(X)$.

A way to resolve these problems is by modifying the transformation for ICs such that it avoids adding more abducibles into a solution when checking for consistency. The idea is to initially compute the lists of abducibles that satisfy the ICs and to save each list using a new predicate $ic/1$. Then, in the abduction stage, every generated abductive solution is tested against those lists by subset checking. If none of these lists is a subset of the solution, then the abductive solution does not violate the ICs. To realize this idea, the definition of IC is modified into a rule of the form $\mathsf{U} \leftarrow L_1, \ldots, L_m$, i.e., abducing abducibles in the body is not forced and therefore they can be left undefined. In other words, this definition allows the body of an IC to be false or undefined [9], allowing us to define abductive solutions under the well-founded semantics as defined below.[1]

Definition 3.1 (Abductive Solution (Modified)) *Given an abductive framework $F = \langle \mathcal{P}, \mathcal{AB}, \mathcal{IC} \rangle$, the set of abducibles $\mathcal{S} \subset \mathcal{AB}_g$ is an abductive solution for F if U is undefined in well-founded model $M_{\mathcal{S}}$ of $(\mathcal{P} \cup \mathcal{P}_{\mathcal{S}} \cup \mathcal{IC})$ where $\mathcal{P}_{\mathcal{S}}$ is the smallest set of rules containing the fact A if $A \in \mathcal{S}$ and, for all $A' \in \mathcal{AB}_g \backslash \mathcal{S}$ and A' is not the complement of A, we have that $A' \leftarrow \mathsf{U}$. A set $\mathcal{S}$ is an abductive solution for query Q if $M_{\mathcal{S}} \models Q$.*

Definition 3.2 (Transformation for ICs) *For each $\mathsf{U} \leftarrow \mathcal{B}_i \in \mathcal{IC}$, we define $\mathcal{A}_i \subseteq \mathcal{B}_i$ as the set of abducibles in the body $\mathcal{B}_i$ of the i-th IC, $1 \leq i \leq m$. We define $\mathcal{IC}'$ as the set of ICs where the body is replaced by $\mathcal{B}'_i = \mathcal{B}_i - \mathcal{A}_i$:*

$$\mathsf{U} \leftarrow L_{11}, \ldots, L_{1n_1}. \quad \cdots \quad \mathsf{U} \leftarrow L_{m1}, \ldots, L_{mn_m}.$$

[1] The use of U in ICs is also adapted for abduction in the weak completion semantics [12].

The transformation of ICs in $\mathcal{IC}'$ results in the smallest set containing the rules:

$$U^* \leftarrow U^{*1}, \ldots, U^{*m}.$$

and

$$U^{*i} \leftarrow \alpha(L_{i1}), \ldots, \alpha(L_{in_i}), assert_IC(E_{in_i}).$$

where $assert_IC/1$ asserts the facts $ic/1$ to store the list of abducibles obtaining from the i-th IC's body and:

$$\alpha(L_{ij}) = \begin{cases} l_{ij}(\bar{t}_{ij}, E_{i(j-1)}, E_{ij}) & \text{if } L_{ij} = l_{ij}(\bar{t}_{ij}) \\ not_l_{ij}(\bar{t}_{ij}, E_{i(j-1)}, E_{ij}) & \text{if } L_{ij} = not\ l_{ij}(\bar{t}_{ij}) \end{cases}$$

$1 \leq j \leq n_i$, *with* $E_{i0} = \mathcal{A}_i$.

The predicate $assert_IC/1$ is a TABDUAL system predicate that is only utilized to store the facts of $ic/1$s. This predicate has no direct dependency with any of the tabled predicates. Thus, the use incremental tabling is not necessary. We call U* only once before any of the abduction processes, and in particular, the predicate $assert_IC/1$.

Example 3.1 *Using the same ICs in Example 2.1, the result of the new transformation for ICs is as follows:*

1. $U^* \leftarrow U^{*1}, U^{*2}$.
2. $U^{*1} \leftarrow assert_IC([q(X), r(X)])$.
3. $U^{*2} \leftarrow u(X, [\,], E), assert_IC(E)$.

We also modify the query transformation and add a mechanism for ICs testing on each subgoal to filter its solutions early, especially when we have multiple goals in a query.

Definition 3.3 (Query Transformation) *Given a query Q as $?-G_1, \ldots, G_m.$, we transform it into $\Delta(Q)$, namely $?-\delta(G_1), \ldots, \delta(G_m)$ where δ is defined as follows:*

$$\delta(G_i) = \begin{cases} g_i(\bar{t}_i, T_{i-1}, T_i), test_IC(T_i) & \text{if } G_i = g_i(\bar{t}_i) \\ not_g_i(\bar{t}_i, T_{i-1}, T_i), test_IC(T_i) & \text{if } G_i = not_g_i(\bar{t}_i) \end{cases}$$

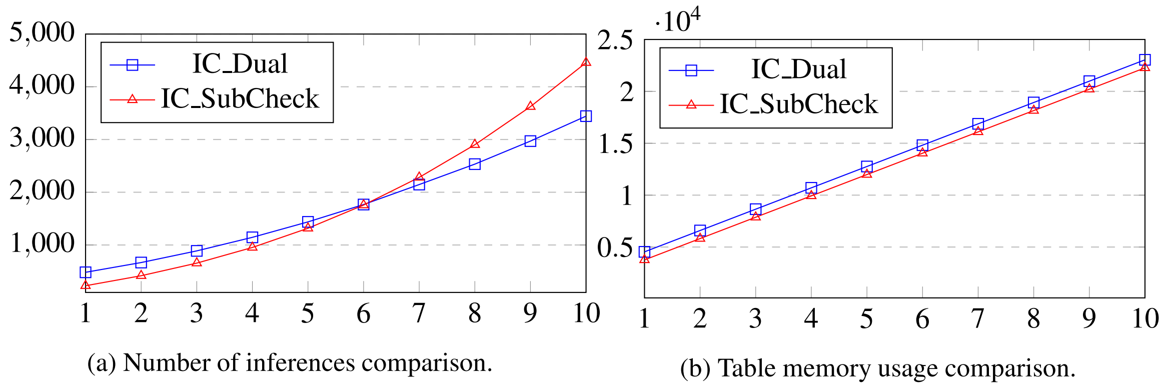
where $T_0 = [\,]$ or other given initial context. We define $test_IC/1$ as follows

$$test_IC(I) \leftarrow forall(ic(X), check_subset(X, I)).$$

where $check_subset(X, I)$ evaluates to $true$ if X is not a subset of I and $forall/2$ is a standard Prolog predicate.

Essentially, we test the solution in I against each list of abducibles asserted by the $ic/1$ facts. If any of those lists is a subset of a solution, then the solution violates the IC associated with that list of abducibles. Otherwise, the solution is accepted.

Example 3.2 *The query Q in Example 2.5 transforms into $?-p(0, [\,], O), test_IC(O)$ and returns only $\mathcal{S} = O = [q(0), q(1), not\ t(0)]$ as a solution. Note that $\{q(0), q(1), not\ t(0), s(0), p(0), not\ u(0)\} \subset M_{\mathcal{S}}$ and thus $M_{\mathcal{S}} \models Q$. The query $p(0)$ with input context $r(1)$ correctly returns $false$ since the query is not satisfiable. Lastly, if we modify the rule $p/1$ into $p(X) \leftarrow q(0), not\ q(1), s(X)$, the query Q correctly returns $[q(0), not\ q(1), not\ t(0)]$ as a solution instead of failing.*



(a) Number of inferences comparison.

(b) Table memory usage comparison.

Figure 1: Comparison of dual transformation and new transformation for ICs in Experiment 3.1.

Theorem 3.1 states that with the new transformation for ICs, the size of the whole program resulting from the TABDUAL transformation is linear in the size of the input program. Suppose $rules(\mathcal{P})$ denotes the number of rules in program $\mathcal{P}$, $|\mathcal{B}_{r_i}|$ denotes the number of literals in the body of rule r_i in $\mathcal{P}$, then, the size of $\mathcal{P}$ is defined as $size(\mathcal{P})$ $size(\mathcal{P}) = \Sigma_{i=1}^{rules(\mathcal{P})}(1 + |\mathcal{B}_{r_i}|)$.

Theorem 3.1 *Given an abductive logic program $\mathcal{P}$, the set of abducibles $\mathcal{AB}$, and $\tau(\mathcal{P})$ as the resulting program after applying the* TABDUAL *transformation. Then $size(\tau(\mathcal{P})) \leq 8 \cdot size(\mathcal{P}) + 4 \cdot |\mathcal{AB}| + 3$.*

We conduct experiments using SWI-Prolog on an artificial problem. The experiments aim to compare the table memory usage (in bytes) and the number of logical inferences between programs resulting from different transformation for ICs, viz., the one with the dual program transformation and with the newly proposed transformation with subset checking. The number of logical inferences defined the classical way, i.e., the number of calls performed or rule heads traversed by Prolog,

Experiment 3.1 *Given $\mathcal{AB} = \{a, b, c\}$ and $\mathcal{IC} = \{\bot \leftarrow p(X), not\ q(X).\}$. We generate n abductive frameworks $\langle \mathcal{P}_n, \mathcal{AB}, \mathcal{IC} \rangle$ where $\mathcal{P}_n$ is a logic program containing rules of the forms:*

- *Rules:* $p(X) \leftarrow a(X)$; $\quad q(X) \leftarrow b(X)$; $\quad t(X) \leftarrow c(X)$.
- *Rule:* $r \leftarrow a(1), \ldots, a(n), not\ q(1), \ldots, not\ q(n)$.
- *Rule:* $r \leftarrow p(1), \ldots, p(n)$.

We run the experiments w.r.t. various sizes of abductive solutions as a result of calling the query $?-r, t(1), \ldots, t(n)$, $1 \leq n \leq 10$. The rules of $r/0$ are introduced to provide rejected and accepted solutions against the IC. The purpose, as indicated by the query, is to examine the effect of removing solutions that violate the IC early before calling a sequence of n goals $t/1$. The size of the solutions, which varies over n, is used as a parameter for evaluating the effect of subset checking over different sizes of abductive solutions. The use of $t/1$ in the query is particularly to examine the effect of the new transformation w.r.t. the size of the query. Note that as n increases, the size of the query also increases, as well as the number of subset checking to perform. We define $p/1$ and $q/1$ to slightly add the complexity of the program, where they serve as intermediate predicates before abducing a and b. The IC has both positive and negative literals in the body to examine the effect of negative literals occurrences. Note that, in the dual transformation, the negative literals in the IC's body (here, $not\ q(X)$) are transformed to positive literals, whose predicates are tabled. On the other hand, using the new transformation for ICs, they remain negative and no tabling is involved.

The result presented in Figure 1a shows that the number of inferences of the new transformation for ICs using subset checking (*IC_SubCheck*) starts below that of the dual transformation (*IC_Dual*), but overtake the *IC_Dual* mode for $n \geq 6$. It shows that performing subset checking on each subgoal increases the number of inferences during abduction as expected. Even when we prune the solutions early, the number of inferences remains high. However, note that in this problem, the pruning is performed only once during the checking of the first subgoal, and thus does not largely affect the number of inference during abduction. Moreover, the dual transformation for ICs may incorrectly filters the abductive solutions, so the gap between these approaches may still be justified by this remark. Meanwhile, the result in Figure 1b depicts that the use of the dual transformation for ICs utilizes slightly more table memory, mainly because the predicate $q/1$, obtained from the dual of *not* $q(X)$ in the IC's body is tabled. It shows that the content of the IC's body may affect the table memory usage during abduction.

4 Optimizing Table Memory Usage

In this section, we address the scalability issue during the abduction stage, particularly when dealing with lots and large abductive solutions. We start by introducing a mechanism to reduce the number of tabled solutions by selectively choosing predicates to table. We consider subsequently the simplification of the problem representation to further reduce the table memory usage.

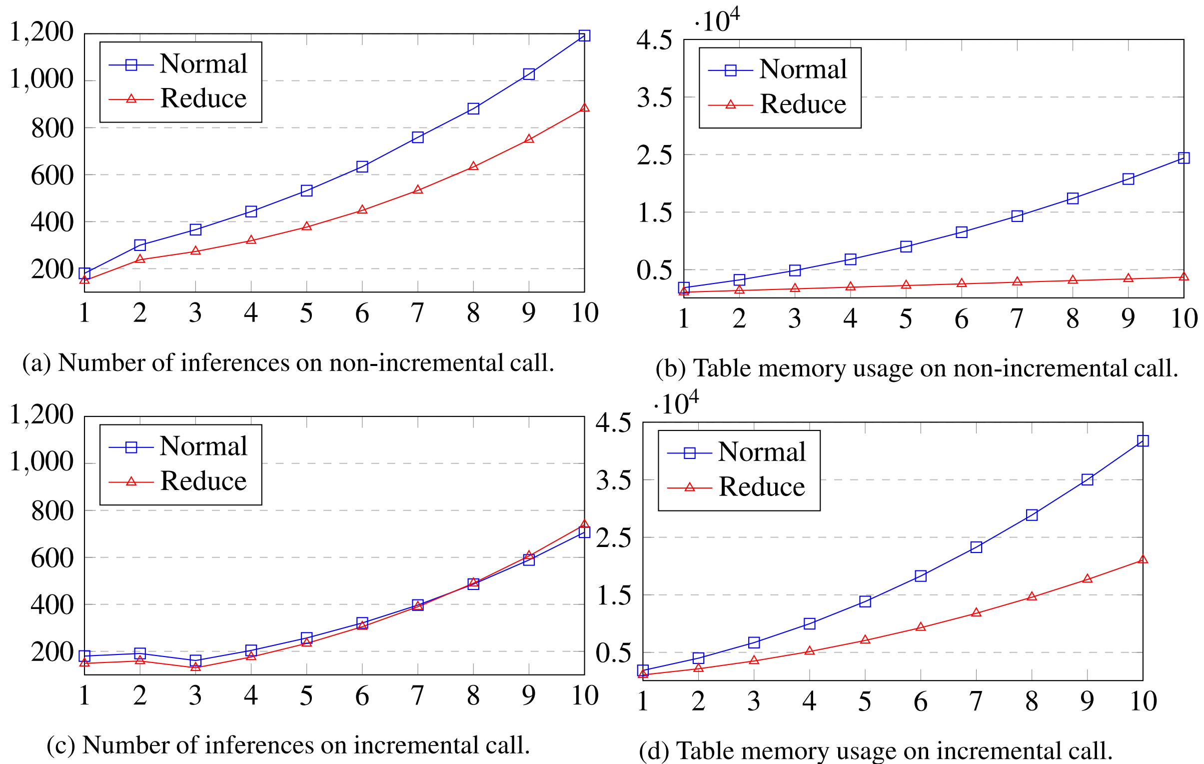
4.1 Reduced Tabled Predicates

Considering *every* rule in introducing predicates to table, as described in the transformation for tabling abductive solution (cf. Sect. 2), may certainly result in greedy space consumption, which in turn may hamper the abduction stage to be completed. We propose here a modification of such naive transformation by selecting tabled predicates based on the structure of the rule. That is, we do not table the predicates whose rules contain *only* abducibles or literals that are defined only by facts. Indeed, abducibles in the body can be easily added into the context. On the other hand, facts can be executed straightforwardly since they do not have body. Note that the other predicates whose rules do not meet the above condition still need to be tabled.

In realizing this modification, we add a procedure to check whether a predicate needs to be tabled based on the above condition. While this reduction is expected to effectively reduce table memory usage, the time and inference needed to process a query may increase depending on the size of the body of the non-tabled predicates' rules, viz., the number of literals in the body. Consider the program in Example 2.1. Using this new mechanism, the predicate $s/1$ is not tabled since it only has one rule whose body contains only the predicate $t/1$, which is an abducible. However, we still need to table the predicates $p/1$ since, even though its rule's body contains the abducible $q/1$, it also contains $s/1$ which is neither an abducible nor defined only by facts. The predicate $u/1$ is also tabled for similar reason. Note that, by avoiding tabling predicate $s/1$, the same inference is repeated every time $s/1$ is called.

4.2 Experiments on Artificial Problem

In this section we conduct experiments using SWI-Prolog to see the growth of the table memory usage (in bytes) and the number of inference calls on processing a query w.r.t. the size of the rule's body of the non-tabled predicates. We compare a program with reduced tabled predicates mechanism (*Reduce*) and the one without such reduction (*Normal*).



(a) Number of inferences on non-incremental call.

(b) Table memory usage on non-incremental call.

(c) Number of inferences on incremental call.

(d) Table memory usage on incremental call.

Figure 2: Number of inferences and table memory usage of Experiment 4.1.

Experiment 4.1 *Given $\mathcal{AB}_n$ as a set of n abducibles $\{a_1, \ldots, a_n\}$. We generate an abductive framework $\langle \mathcal{P}_n, \mathcal{AB}_n, \emptyset \rangle$ where $\mathcal{P}_n$ is a logic program containing rules of the forms:*

- *Facts $b_i(X)$. for $1 \leq i \leq n$.*
- *Rules $p_i(X) \leftarrow a_1(X), \ldots, a_i(X), b_1(X), \ldots, b_i(X)$. for $1 \leq i \leq n$.*
- *Rules $q_i(X) \leftarrow p_1(X), \ldots, p_i(X)$. for $1 \leq i \leq n$.*

For this experiment, we use $n = 10$.

We run these experiments for various size of the predicate p_i's body for $1 \leq i \leq n$ using the query $?-q_i(1)$ both in *Normal* and *Reduce* mode. The rules of $p_i/1$ are defined only by abducibles a_i and predicate $b_i/1$; the latter is defined only by facts. We define $q_i/1$ for the query that can reuse the solutions of $p_1/1$ to $p_i/1$ if they are tabled. The size of $q_i/1$'s body that varies w.r.t. i becomes a parameter to analyse the number of inferences and the table memory usage. For example, the query $q_2(1)$ will call the subgoal $p_1(1)$ and $p_2(1)$, while query $q_3(1)$ will additionally call $p_3(1)$. In *Normal* mode, i.e., without reducing the tabled predicates, each time the rule $p_i(X)$ is called, we table its solutions which can be reused in subsequent calls. However, in the *Reduce* mode, those results are not tabled and have to be recomputed.

We run these experiments on two different query calling cases. We define the *incremental call* of the queries, in which we make use of the previously tabled predicate $p_i(X)$ in normal mode. For instance, the call of query $q_2(1)$ makes use of the table produced during the call of query $q_1(1)$, which has previously been executed. On the other hand, in the *non-incremental call* of each query, the resulting table is

abolished after each query call. As an example, after the call of $q_1(1)$, we abolish the table before executing the next query, $q_2(1)$.

On Figure 2a, there is a decrease of inference calls on non-incremental call in *Reduce* mode compared to that of *Normal* mode. It shows that by not tabling some predicates, we can reduce the number of inferences, especially those on the tabling operations. In Figure 2b, the table memory usage of *Reduce* mode is significantly lower compared to that of *Normal* mode. The gap is larger as i increases, since, for example, query $q_{10}(1)$ creates the tables for $p_1(1)$ until $p_{10}(1)$ in addition to $q_{10}/1$. That is not the case on *Reduce* mode, since $p_i/1$ is not tabled, and the program only produces table for $q_{10}/1$. The graph thus shows only little increases over i.

On the incremental call, Figure 2c shows that the number of inferences for *Reduce* mode starts below the *Normal* mode but, at some point, overtakes that of *Normal* mode. It happens since, in the *Normal* mode, the previously tabled predicates are used on the next query call consecutively. For example, when calling the query $q_3(1)$, the program can reuse the table of $p_1(1)$ and $p_2(1)$ that is produced during the execution of the previous query $q_1(1)$ and $q_2(1)$. This is not the case for the *Reduce* mode since on every call, it must execute $p_i/1$ again as they are not tabled. Nevertheless, the table memory usage in Figure 2d for *Reduce* mode is still lower than that of *Normal* mode. The increase in table memory usage between Figure 2b and 2d happens since the table for all $q_i/1$ from the previous query execution is not deleted before the execution of the next query during the incremental call. While in non-incremental call, those tables are abolished between each call.

4.3 Experiments on Real World Problem

These experiments are performed on an abduction problem in chemoprevention [8]. Abduction is enacted to study genes that affect cancer cells activation or inactivation, with the knowledge on active or inactive cells and their propagation pathways. The problem itself is challenging since the program contains a large number of facts and rules with intricate relationships of genes and cancer cells.

4.3.1 Experimental Results

The abduction in these experiments are conducted with answer subsumption to compute only minimal explanations. Furthermore, we shall compare the scalability of the proposed approach in this paper and the previous work in [11] with answer subsumption only. In this evaluation, we use XSB Prolog since there is a difference on partial order mechanism between answer subsumption in XSB Prolog and SWI Prolog implementation. XSB Prolog can provide multiple minimal abductive solutions in case they are incomparable w.r.t. the subset operation, while SWI Prolog can only provide a single solution.

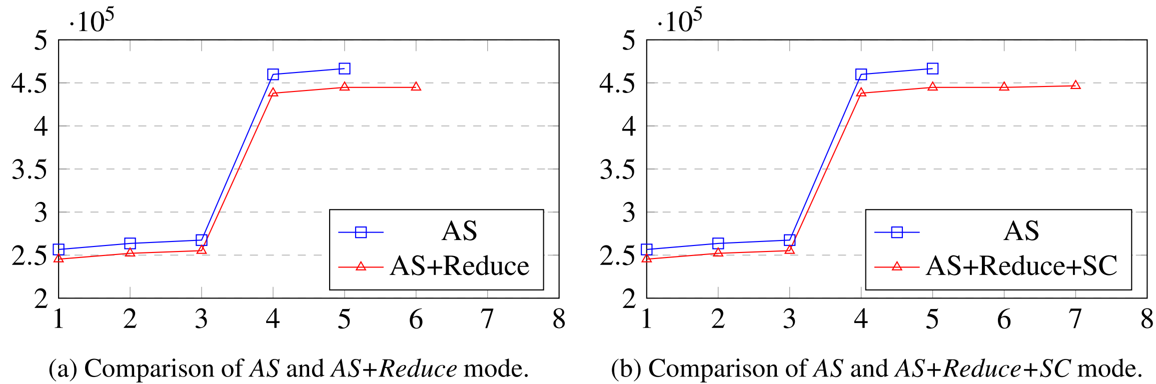
Experiment 4.2 *The main query comprises eight subgoals:*

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

The ICs of this problem is as follows:

1. $\leftarrow drug_induced(phase0, drug, Gene), drug_inhibited(phase0, drug, Gene).$
2. $\leftarrow drug_induced(phase0, drug, apoptosis).$

where $drug_induced/3$ *and* $drug_inhibited/3$ *are both abducibles.*



(a) Comparison of *AS* and *AS+Reduce* mode. (b) Comparison of *AS* and *AS+Reduce+SC* mode.

Figure 3: Comparison of table memory usage in Experiment 4.2

The objective is to analyse the effects on the table memory usage (in bytes) when we consider answer subsumption only (*AS*), a combination with reduced tabled predicates (*AS+Reduce*), as well as a further combination with the new transformation for ICs by subset checking (*AS* + *Reduce* + *SC*). We run the query, starting with only the first subgoal, and continuing to harder queries with more subgoals, until all eight subgoals are executed. However, unlike the experiment in [11], we do so non-incrementally, i.e., we abolish the table between each query execution. We also set the timeout for 25200 seconds (7 hours).

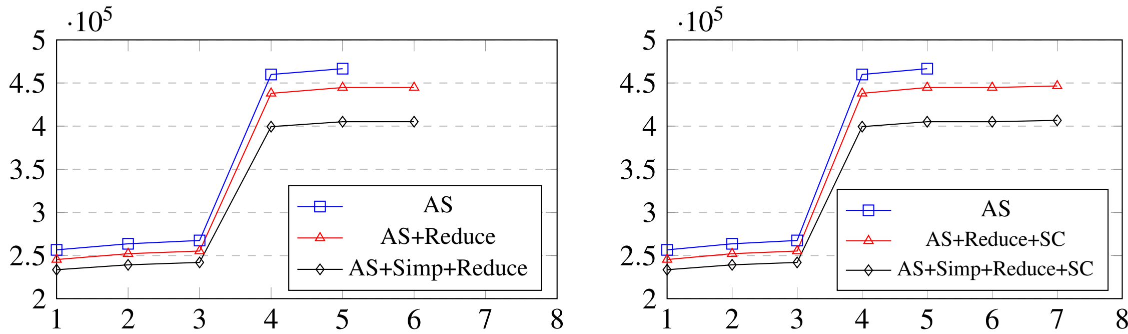
The comparison of the resulting table memory usage is presented in Figure 3. We see that, in Figure 3a, while [11] manages to finish all queries incrementally, they are still unable to finish the execution for more than five subgoals non-incrementally. By reducing the tabled predicates, we manage to execute six subgoals. Unfortunately, for this particular problem, the decrease of the table memory usage still cannot help in finishing all subgoals. In Figure 3b, using *AS+Reduce+SC* mode, we manage to complete seven subgoals. The reason is that the program in *AS* mode tries to add several sets of abducibles into the solutions while the program on *AS+Reduce+SC* does not, since it uses the new transformation for ICs. Thus, it produces smaller and less number of solutions compared to the program in *AS* mode, that produces at most twice as much solutions since the first IC in the problem contains two literals. We can also see that the table memory usage with the proposed approach is less than the *AS* mode. Even so, the difference is relatively small, which may indicate that non-tabled predicates are not called as often as the tabled ones.

4.3.2 Simplification of Domain-Specific Model Representation

Since the executions of all subgoals is still not possible, we explore another approach to try and finish all subgoals executions. We further simplify the representation of the problem by eliminating the first argument of the goals, viz., *phase0*, as there is no relationship between *phase0* and others in the given knowledge base. This argument can therefore safely be eliminated from the model representation. In the case where many phases of experiments occur, we can simply run the abductive program several times while adjusting the experiment-specific part of the program to each independent phase. The simplification is performed to decrease the table memory usage further.

Experiment 4.3 *Since we remove phase from the program, the main query is redefined as:*

$$active(aif), active(endo_g), inactive(caspase9), inactive(caspase6),$$
$$inactive(bcl2), inactive(caspase7), inactive(akt), inactive(xiap).$$



(a) Table memory usage comparison with dual transformation for ICs.

(b) Table memory usage comparison with new transformation for ICs.

Figure 4: Comparison of table memory usage between initial and simplified model representation.

The ICs are redefined accordingly.

The objective and scenario of these experiments are similar with Experiment 4.2. However, the query is adjusted as in Experiment 4.3. We also consider the effect on the representation simplification (*Simp*). Based on Figure 4, we can see that the the simplification of the model can successfully reduce the table memory usage further, since the arity of each predicate has been reduced by one, and thus reducing the amount of memory needed to table them. However, even with the decrease of table memory usage, the execution of eight subgoals is still not possible even with the new transformation for ICs on *AS+Simp+Reduce+SC* mode. It shows that the number of abductive solutions generated for eight subgoals may still be too numerous for the program to terminate within the allocated time of 25200 seconds.

5 Concluding Remarks

In this paper, we propose a new transformation for ICs to warrant their correct application in filtering solutions by utilizing subset checking as a way to ensure that the abductive solutions adheres to the ICs, instead of using the dual program transformation. However, this approach may potentially increase the number of inferences during abduction, as presented in the experiments. We also introduce a mechanism to reduce the number of tabled solutions and simplify the problem representation to further reduce the table memory usage. Our experiments show that by reducing the number of tabled predicates, we can decrease the table memory usage during the abduction stage. As a trade off, the number of inferences may increase if the body of the non-tabled predicate is large. Furthermore, by simplifying the representation of the problem, the table memory usage can be decreased further. However, this approach is problem specific, and requires an understanding of the problem domain in order to be implemented successfully. In our experiment on Abduction in Chemo-prevention, we also managed to finish the abduction process non-incrementally up to seven subgoals using the reduction and the new transformation for ICs. The previous results in [11] can only finish five subgoals non-incrementally. The results for eight subgoals are still unavailable due to the size of the abductive solutions generated during the abduction process.

The idea of tabling abductive solutions from one abductive context to be reused in other contexts (contextual abduction) is presented by ABDUAL [2], as a theory for abduction over Well-Founded semantics. In [13], the technique for the program transformation from abductive normal logic programs

into tabled logic programs is presented employing the dual transformation of ABDUAL. Several works to improve TABDUAL have been proposed in [5, 11] to reduce the tabling memory usage during the abduction process. However, both implementation utilize some tabling features and do not make any adjustment in the program transformation. Moreover, there are several issues regarding those implementations as we previously stated.

We use the same benchmark as in [5, 11] for evaluating our proposed approaches, which use the chemoprevention abductive program previously described in [8]. This abduction system is build on top of the A-system [18]. Given a query Q, the A-system computes an extension of the abducibles such that it entails Q and the ICs. We cannot compare our result directly to the result in [8]. One problem that prevents the execution of all subgoals is the large number and size of the abductive solutions. In [8], the size of the solutions are reduced by limiting the search depth of the A-system, showing that even in low depth searches, the quality of the provided results are satisfactory. Currently, we cannot replicate it in both SWI-Prolog and XSB-Prolog, since both have no functionality to limit search depth. Thus, it may be interesting to explore other ways to limit the size of the solutions, such as the bounded rationality approach in XSB-Prolog.

We may also attempt to selectively decrease the number of tabled solutions further, e.g., by removing redundant solutions. In this paper, we implement subset minimality for the answer subsumption criteria in selecting preferred solutions. Another approach that can be explored is by using minimal cardinality criteria. Furthermore, it may also be interesting to solve different challenging real-life problems using TABDUAL as other benchmarks. Lastly, the use of meta-interpreter which can be unfolded w.r.t. the interpreted program to generate an executable program free of the meta-interpreter overhead can also be examined further.

Acknowledgement This work was supported by the PUTI grant NKB-847/UN2.RST/HKP.05.00/2020 funded by the Directorate of Research & Development Universitas Indonesia

References

[1] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello & Paolo Torroni (2005): *Security Protocols Verification in Abductive Logic Programming: A Case Study*. In: International Workshop on Engineering Societies in the Agents World, Springer, pp. 106–124, doi:10.1007/11759683_7.

[2] José Júlio Alferes, Luís Moniz Pereira & Terrance Swift (2004): *Abduction in Well-founded Semantics and Generalized Stable Models via Tabled Dual Programs*. Theory and Practice of Logic Programming 4(4), pp. 383–428, doi:10.1017/S1471068403001960.

[3] Marc Denecker & Antonis Kakas (2002): *Abduction in Logic Programming*. In: Computational logic: Logic Programming and Beyond, Springer, pp. 402–436, doi:10.1007/3-540-45628-7_16.

[4] Joseph Gartner, Terrance Swift, Allen Tien, Carlos Viegas Damásio & Luís Moniz Pereira (2000): *Psychiatric Diagnosis from the Viewpoint of Computational Logic*. In: International Conference on Computational Logic, Springer, pp. 1362–1376, doi:10.1007/3-540-44957-4_91.

[5] Muhammad Okky Ibrohim & Ari Saptawijaya (2019): *Tabling with Interned Terms on Contextual Abduction*. Jurnal Ilmu Komputer dan Informasi 12(1), pp. 1–11, doi:10.21609/jiki.v12i1.569.

[6] Antonis C. Kakas & Antonia Michael (2001): *An Abductive-based Scheduler for Air-crew Assignment*. Applied Artificial Intelligence 15(3), pp. 333–360, doi:10.1080/08839510151063299.

[7] Robert Kowalski & Fariba Sadri (2011): *Abductive Logic Programming Agents with Destructive Databases*. Annals of Mathematics and Artificial Intelligence 62(1-2), pp. 129–158, doi:10.1007/s10472-011-9253-y.

[8] Sotiris Lazarou, Antonis C. Kakas, Christiana Neophytou & Andreas Constantinou (2013): *Automated Scientific Assistant for Cancer and Chemoprevention*. In: *IFIP International Conference on Artificial Intelligence Applications and Innovations*, 412, Springer, pp. 96–109, doi:10.1007/978-3-642-41142-7_11.

[9] Luís Moniz Pereira, Joaquim Nunes Aparício & José Júlio Alferes (1991): *Hypothetical Reasoning with Well Founded Semantics*. In: *Procs. 3rd Scandinavian Conference on Artificial Intelligence*, 12, IOS Press, p. 289. Available at https://run.unl.pt/bitstream/10362/64287/1/scai91-1.pdf.

[10] Luís Moniz Pereira & Ari Saptawijaya (2016): *Programming Machine Ethics*. 26, Springer, doi:10.1007/978-3-319-29354-7.

[11] Syukri Mullia Adil Perkasa, Ari Saptawijaya & Luís Moniz Pereira (2017): *Tabling in Contextual Abduction with Answer Subsumption*. In: *2017 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, IEEE, pp. 459–464, doi:10.1109/ICACSIS.2017.8355074.

[12] Emmanuelle-Anna Dietz Saldanha, Steffen Hölldobler, Carroline Dewi Puspa Kencana Ramli & Luis Palacios Medinacelli (2018): *A Core Method for the Weak Completion Semantics with Skeptical Abduction*. *Journal of Artificial Intelligence Research* 63, pp. 51–86, doi:10.1613/jair.1.11236.

[13] Ari Saptawijaya & Luís Moniz Pereira (2015): *Tabdual: a Tabled Abduction System for Logic Programs*. *IfCoLog Journal of Logics and Their Applications* 2(1), pp. 69–124. Available at http://www.collegepublications.co.uk/downloads/ifcolog00003.pdf#page=79.

[14] T. Swift (1999): *Tabling for Non-Monotonic Programming*. *Annals of Mathematics and Artificial Intelligence* 25(3-4), pp. 201–240, doi:10.1023/A:1018990308362.

[15] T. Swift & D. S. Warren (2012): *XSB: Extending Prolog with Tabled Logic Programming*. *Theory and Practice of Logic Programming* 12(1-2), pp. 157–187, doi:10.1017/S1471068411000500.

[16] Terrance Swift & David S Warren (2010): *Tabling with Answer Subsumption: Implementation, Applications and Performance*. In: *European Workshop on Logics in Artificial Intelligence*, Springer, pp. 300–312, doi:10.1007/978-3-642-15675-5_26.

[17] Allen Van Gelder, Kenneth A Ross & John S Schlipf (1991): *The Well-Founded Semantics for General Logic Programs*. *Journal of the ACM (JACM)* 38(3), pp. 619–649, doi:10.1145/116825.116838.

[18] Bert Van Nuffelen & Antonis Kakas (2001): *A-System: Declarative Programming with Abduction*. In: *International Conference on Logic Programming and Nonmonotonic Reasoning*, Springer, pp. 393–397, doi:10.1007/3-540-45402-0_29.

[19] David S Warren (2013): *Interning Ground Terms in XSB*. *Colloquium on Implementation of Constraint and Logic Programming Systems (CICLOPS 2013)*, pp. 74–86. Available at http://www.dcc.fc.up.pt/~ricroc/homepage/publications/2013-CICLOPS.pdf#page=80.