Default Logic and Bounded Treewidth*

Johannes K. Fichte[1] Markus Hecher[1]
Irina Schindler[2]

[1]: TU Wien, Austria, {fichte,hecher}@dbai.tuwien.ac.at
[2]: Leibniz Universität Hannover, Germany, schindler@thi.uni-hannover.de

December 30, 2017

Abstract

In this paper, we study Reiter's propositional default logic when the treewidth of a certain graph representation (semi-primal graph) of the input theory is bounded. We establish a dynamic programming algorithm on tree decompositions that decides whether a theory has a consistent stable extension (Ext). Our algorithm can even be used to enumerate all generating defaults (EnumSE) that lead to stable extensions. We show that our algorithm decides Ext in linear time in the input theory and triple exponential time in the treewidth (so-called *fixed-parameter linear* algorithm). Further, our algorithm solves EnumSE with a pre-computation step that is linear in the input theory and triple exponential in the treewidth followed by a linear delay to output solutions.

1 Introduction

Reiter's *default logic (DL)* is one of the most fundamental formalisms to non-monotonic reasoning where reasoners draw tentative conclusions that can be retracted based on further evidence [19, 16]. DL augments classical logic by rules of default assumptions (*default rules*). Intuitively, a default rule expresses "in the absence of contrary information, assume ...". Formally, such rule is a triple $\frac{p:j}{c}$ of formulas p, j, and c expressing "if prerequisite p can be deduced and justification j is never violated then assume conclusion c". For an initial set of facts, beliefs supported by default rules are called an extension of this set of facts. If the default rules can be applied consistently until a fixed point, the extension is a maximally consistent view (*consistent stable extension*) with respect to the facts together with the default rules. In DL stable extensions involve the construction of the deductive closure, which can be generated from the conclusions of the defaults and the initial facts by means of so-called generating defaults. However, not every generating default leads to a stable extension. If a generating default leads to a stable extension, we call it a stable default set. Our problems of interest are deciding whether a default theory has a consistent stable extension (Ext), output consistent stable default sets (CompSE), counting the number of stable default sets (#SE), and enumerating all stable default sets (EnumSE). All these problems are of high worst case complexity, for example, the problem Ext is Σ_2^{p}-complete [14].

Parameterized algorithms [7] have attracted considerable interest in recent years and allow to tackle hard problems by directly exploiting certain structural properties present in input instances (the *parameter*). For example, Ext can be solved in polynomial time for input theories that allow for small backdoors into

*The work has been supported by the Austrian Science Fund (FWF), Grants Y698 and P26696, and the German Science Fund (DFG), Grant ME 4279/1-1. The first two authors are also affiliated with the Institute of Computer Science and Computational Science at University of Potsdam, Germany. The final publication will be available at Springer proceedings of LATA 2018.

tractable fragments of DL [11]. Another parameter is treewidth, which intuitively measures the closeness of a graph to a tree. Ext can also be solved in linear time for input theories and a (non-elementary) function that depends on the treewidth of a certain graph representation of the default theory (incidence graph) [17]. This result relies on logical characterization in terms of a so-called MSO-formula and Courcelle's theorem [5]. Unfortunately, the non-elementary function can become extremely huge and entirely impractical [15]. More precisely, the result by Meier et al. [17] yields a function that is at least quintuple exponential in the treewidth and the size of the MSO-formula. This opens the question whether one can significantly improve these runtime bounds. A technique to obtain better worst-case runtime bounds that often even allows to practically solve problem instances, which have small treewidth, are dynamic programming (DP) algorithms on tree decompositions [4, 9, 10]. In this paper, we present such a DP algorithm for DL, which uses a slightly simpler graph notation of the theory (semi-primal graph).

Contributions. We introduce DP algorithms that exploit small treewidth to solve Ext and CompSE in time triple exponential in the semi-primal treewidth and linear in the input theory. Further, we can solve #SE in time triple exponential in the semi-primal treewidth and quadratic in the input theory. Our algorithm can even be used to enumerate all stable default sets (EnumSE) with a pre-computation step that is triple exponential in the semi-primal treewidth and linear in the input theory followed by a linear delay for outputting the solutions (Delay-FPT [6]).

2 Default Logic

We assume familiarity with standard notions in computational complexity, the complexity classes P and NP as well as the polynomial hierarchy. For more detailed information, we refer to other standard sources [18, 12, 8]. For parameterized (decision) problems we refer to work by Cygan et al. [7].

A *literal* is a (propositional) variable or its negation. The *truth evaluation* of (propositional) formulas is defined in the standard way [16]. In particular, $\theta(\bot) = 0$ and $\theta(\top) = 1$ for an assignment θ. Let f and g be formulas and $X = \mathrm{Vars}(f) \cup \mathrm{Vars}(g)$. We write $f \vDash g$ if and only if for all assignments $\theta \in 2^X$ it holds that if the assignment θ satisfies f, then θ also satisfies g. Further, we define the *deductive closure of* f as $\mathrm{Th}(f) := \{\, g \in \mathcal{P} \mid f \vDash g \,\}$ where $\mathcal{P}$ is the family that contains all formulas. In this paper, whenever it is clear from the context, we may use sets of formulas and a conjunction over formulas equivalently. In particular, we let for formula f and a family $\mathcal{M}$ of sets of variables be $\mathrm{Mod}_{\mathcal{M}}(f) := \{M \mid M \in \mathcal{M}, M \vDash f\}$. We denote with Sat the problem that asks whether a given formula f is satisfiable.

We define for formulas p, j, and c a *default rule* d as a triple $\frac{p:j}{c}$; p is called the *prerequisite*, j is called the *justification*, and c is called the *conclusion*; we set $\alpha(d) := p$, $\beta(d) := j$, and $\gamma(d) := c$. The mappings α, β and γ naturally extend to sets of default rules. We follow the definitions by Reiter [19]. A *default theory* $\langle W, D\rangle$ consists of a set W of propositional formulas (knowledge base) and a set of default rules.

Definition 1. *Let $\langle W, D\rangle$ be a default theory and E be a set of formulas. Then, $\Gamma(E)$ is the smallest set of formulas such that: (i) $W \subseteq \Gamma(E)$ (ii) $\Gamma(E) = \mathrm{Th}(\Gamma(E))$, and (iii) for each $\frac{p:j}{c} \in D$ with $p \in \Gamma(E)$ and $\neg j \notin E$, it holds that $c \in \Gamma(E)$. E is a* stable extension *of $\langle W, D\rangle$, if $E = \Gamma(E)$. An extension is* inconsistent *if it contains $\bot$, otherwise it is called* consistent*. The set $G = \{\, d \mid \alpha(d) \in E, \neg\beta(d) \notin E, d \in D \,\}$ is called the set of* generating defaults *of extension E and default theory D.*

The definition of stable extensions allows inconsistent stable extensions. However, inconsistent extensions only occur if the set W is already inconsistent where $\langle W, D\rangle$ is the theory of interest [16, Corollary 3.60]. In consequence, (i) if W is consistent, then every stable extension of $\langle W, D\rangle$ is consistent, and (ii) if W is inconsistent, then $\langle W, D\rangle$ has a stable extension. For Case (ii) the stable extension consists of all formulas. Therefore, we consider only consistent stable extensions. For default theories with consistent W, we can trivially transform every formula in W into a default rule. Hence, in this paper we generally assume that

$W = \emptyset$ and write a default theory simply as set of default rules. Moreover, we refer by $\mathrm{SE}(D)$ to the set of all consistent stable extensions of D.

Example 1. *Let the default theories D_1 and D_2 be given as*

$$D_1 := \left\{ d_1 = \frac{\top : a}{a \vee b}, d_2 = \frac{\top : \neg a}{\neg b} \right\},$$

$$D_2 := \left\{ d_1 = \frac{c : a}{a \vee b}, d_2 = \frac{c : \neg a}{\neg b}, d_3 = \frac{\top : c}{c}, d_4 = \frac{\top : \neg c}{\neg c} \right\}.$$

D_1 has no stable extension, while D_2 has only one stable extension $E_1 = \{\neg c\}$.

In our paper, we use an alternative characterization of stable extension beyond fixed point semantics, which is inspired by Reiter's stage construction [19].

Definition 2. *Let D be a default theory and $S \subseteq D$. Further, we let $E(S) := \{\gamma(d) \mid d \in S\}$. We call a default $d \in D$ p-*satisfiable *in S, if $E(S) \cup \neg\alpha(d)$ is satisfiable; and j-*satisfiable *in S, if $E(S) \cup \beta(d)$ is unsatisfiable;* c-satisfiable *in S, if $d \in S$. The set S is a* satisfying default set*, if each default $d \in D$ is p-satisfiable in S, or j-satisfiable in S, or c-satisfiable in S.*

The set S is a stable default set*, if (i) S is a satisfying default set and (ii) there is no S' where $S' \subsetneq S$ such that for each default d it holds that d is p-satisfiable in S', or j-satisfiable in S, or c-satisfiable in S'. We refer by $SD(D)$ to the set of all stable default sets of D.*

The following lemma establishes that we can simply use stable default sets to obtain stable extensions of a default theory.

Lemma 1 ($\star$[1])**.** *Let D be a default theory. Then,*

$$\mathrm{SE}(D) = \bigcup_{S \in SD(D)} \mathrm{Th}(\{\gamma(d) \mid d \in S\}).$$

In particular, $S \in SD(D)$ is a generating default of extension $\mathrm{Th}(\{\gamma(d) \mid d \in S\})$.

Given a default theory D we are interested in the following problems:

The *extension existence problem* (called Ext) asks whether D has a consistent stable extension. Ext is Σ_2^{p}-complete [14]. The extension computation problem (called CompSE) asks to output a stable default set of D. The extension counting problem (called #SE) asks to output the number of stable default sets of D. The enumerating problem asks to enumerate all stable default sets of D (called EnumSE).

3 Dynamic Programming on TDs for Default Logic

In this section, we present the basic methodology and definitions to solve our problems more efficiently for default theories that have small treewidth. Our algorithms are inspired by earlier work for another non-monotonic framework [9]. However, due to much more evolved semantics of DL, we require extensions of the underlying concepts.

Before we provide details, we give an intuitive description. The property treewidth was originally introduced for graphs and is based on the concept of a tree decomposition (TD). Given a graph, a TD constructs a tree where each node consists of sets of vertices of the original graph (bags) such that additional conditions hold. Then, we define a dedicated graph representation of the default theory and our algorithms

[1]Statements or descriptions whose proofs or details are omitted due to space limitations are marked with "$\star$". These statements are sketched in the appendix.

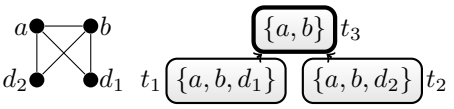

Figure 1 Graph G (left) and an TD $\mathcal{T}$ (right) of G.

work by dynamic programming (DP) along the tree decomposition (post-order) where at each node of the tree, information is gathered in tables. The size of these tables is triple exponential in the size of the bag. Intuitively, the TD fixes an order in which we evaluate the default theory. Moreover, when we evaluate the default theory for one node, we can restrict the theory to a sub-theory and parts of prerequisites, justifications, and conclusions that depends only on the content of the currently considered bag.

Tree Decompositions. Let $G = (V, E)$ be a graph, $T = (N, F, n)$ be a tree (N, F) with root n, and $\chi : N \to 2^V$ be a mapping. We call the sets $\chi(\cdot)$ *bags* and N the set of nodes. The pair $\mathcal{T} = (T, \chi)$ is a *tree decomposition (TD)* of G if the following conditions hold: (i) for every vertex $v \in V$ there is a node $t \in N$ with $v \in \chi(t)$; (ii) for every edge $e \in E$ there is a node $t \in N$ with $e \subseteq \chi(t)$; and (iii) for any three nodes $t_1, t_2, t_3 \in N$, if t_2 lies on the unique path from t_1 to t_3, then $\chi(t_1) \cap \chi(t_3) \subseteq \chi(t_2)$. The *width* of the TD is the size of the largest bag minus one. The *treewidth* $tw(G)$ is the minimum width over all possible TDs of G. For $k \in \mathbb{N}$ we can compute a TD of width k or output that no exists in time $2^{\mathcal{O}(k^3)} \cdot |V|$ [3].

Next, we restrict the TD $\mathcal{T}$ such that we have only nice case distinctions for our DP algorithm later. Therefore, we define a *nice TD* in the usual way as follows. Given a TD (T, χ) with $T = (N, \cdot, \cdot)$, for a node $t \in N$ we say that type(t) is *leaf* if t has no children; *join* if t has children t' and t'' with $t' \neq t''$ and $\chi(t) = \chi(t') = \chi(t'')$; *int* ("introduce") if t has a single child t', $\chi(t') \subseteq \chi(t)$ and $|\chi(t)| = |\chi(t')| + 1$; *rem* ("removal") if t has a single child t', $\chi(t) \subseteq \chi(t')$ and $|\chi(t')| = |\chi(t)| + 1$. If every node $t \in N$ has at most two children, type$(t) \in \{leaf, join, int, rem\}$, and bags of leaf nodes and the root are empty, then the TD is called *nice*. For every TD, we can compute a nice TD in linear time without increasing the width [3]. In our algorithms we will traverse a TD bottom up, therefore, let post-order(T, t) be the sequence of nodes in post-order of the induced subtree $T' = (N', \cdot, t)$ of T rooted at t.

Example 2. *Figure 1 (left) depicts a graph G together with a TD of width 2 of G. Further, the TD $\mathcal{T}$ in Figure 2 sketches main parts of a nice TD of G (obvious parts are left out).*

Graph Representations of Default Theories. For a default theory D, its *primal graph* $P(D)$ is the graph that has the variables of D as vertices and an edge $a\,b$ if there exists a default $d \in D$ and $a, b \in \text{Vars}(d)$. The incidence graph $I(G)$ is the bipartite graph, where the vertices are variables of D and defaults $d \in D$, and there is an edge $d\,a$ between a default $d \in D$ and a corresponding variable $a \in \text{Vars}(d)$. The *semi-primal graph* $S(D)$ of D is the graph, where the vertices are variables $\text{Vars}(D)$ and defaults of D. For each default $d \in D$, we have an edge $a\,d$ if variable $a \in \text{Vars}(d)$ occurs in d. Moreover, there is an edge $a\,b$ if either $a, b \in \text{Vars}(\alpha(d))$, or $a, b \in \text{Vars}(\beta(d))$, or $a, b \in \text{Vars}(\gamma(d))$[2]. Observe the following connection. For any default theory D, we have that $tw(I(D)) \leq tw(S(D))$. Note that earlier work [17] uses a special version of the incidence graph $I'(D)$. The graph $I'(D)$ is a supergraph of $I(D)$ and still a bipartite graph, which contains an additional vertex for each subformula of every occurring formula, and corresponding edges between subformulas and variables. Consequently, we obtain the bound $tw(I(D)) \leq tw(I'(D))$.

Example 3. *Recall default theory D_1 of Example 1. We observe that graph G in the left part of Figure 1 is the semi-primal graph of D_1.*

In our DP algorithms for default logic we need to remember when we can evaluate a formula (prerequisite, justification, or conclusion) for a default, i.e., we have a default and all the variables of the formula in a bag. To that end, we introduce labels of nodes. Since we work along the TD and want a unique point

[2]Note that these formulas may also be $\top$ or $\bot$, which we "simulate" by means of the same formula $v \vee \neg v$ or $v \wedge \neg v$, where variable v does not occur in the default theory.

Listing 1: Algorithm $\mathcal{DP}(\mathcal{T})$ for Dynamic Programming on TD $\mathcal{T}$ for DL, cf. [9].

In: Pretty LTD $\mathcal{T} = (T, \chi, \delta)$ with $T = (N, \cdot, n)$ of the semi-primal graph $S(D)$.
Out: A table for each node $t \in T$ stored in a mapping Tables[t].

```
for iterate t in post-order(T,n) do
    Child-Tabs := {Tables[t'] | t' is a child of t in T}
    Tables[t] ← SPRIM(t, χ(t), δ(t), D_t, Child-Tabs)
return Tables[·]
```

where to evaluate, we restrict a label to the first occurrence when working along the TD. A *labeled tree decomposition (LTD)* $\mathcal{T}$ of a default theory D is a tuple $\mathcal{T} = (T, \chi, \delta)$ where (T, χ) is a TD of $S(D)$ and $\delta : N \to 2^{(\{\alpha,\beta,\gamma\} \times D)}$ is a mapping where for any (f, d) in $\{\alpha, \beta, \gamma\} \times D$ it holds that (i) if $(f, d) \in \delta(t)$, then $\{d\} \cup f(d) \subseteq \chi(t)$; and (ii) if $\{d\} \cup f(d) \subseteq \chi(t)$ and there is there is no descendent t' of t such that $(f, d) \in \delta(t')$, then $(f, d) \in \delta(t)$.

We need special case distinctions for DL. Therefore, we restrict an LTD as follows. For a node $t \in N$ that has exactly one child t' where $\chi(t) = \chi(t')$ and $\delta(t) \neq \emptyset$, we say that type(t) is *label*. If every node $t \in N$ has at most two children, type$(t) \in \{$*leaf*, *join*, *int*, *label*, *rem*$\}$, bags of leaf nodes and the root are empty, $|\delta(t)| \leq 1$, and $\delta(t) = \emptyset$ for type$(t) \neq$ *label* then the LTD is called *pretty*. It is easy to see that we can construct in linear time a pretty LTD without increasing the width from a nice TD, simply by traversing the tree of the TD and constructing the labels and duplicating nodes t where $\delta(t) \neq \emptyset$. Assume in the following, that we use pretty LTDs, unless mentioned otherwise.

Next, we briefly present the methodology and underlying ideas of our DP algorithms on TDs. The basis for our Algorithm is given in Listing 1 ($\mathcal{DP}$), which traverses the underlying tree of the given LTD (T, χ, δ) in post-order and runs an algorithm SPRIM at each node $t \in T$. SPRIM computes a new table τ_t based on the tables of the children of t. It has only a "local view" on *bag-defaults*, which are simply the "visible" defaults, i.e., $D_t := D \cap \chi(t)$. Intuitively, we store in each table information such as partial assignments of D_t, that is necessary to locally decide the default theory without storing information beyond variables that belong to the bag $\chi(t)$. Further, the *default theory below* t is defined as $D_{\leq t} := \{d \mid d \in D_{t'}, t' \in \text{post-order}(T, t)\}$, and the *default theory strictly below* t is $D_{<t} := D_{\leq t} \setminus D_t$. For root n of T, it holds that $D_{\leq n} = D_{<n} = D$.

Example 4. *Intuitively, the LTD of Figure 1 enables us to evaluate D by analyzing sub-theories ($\{d_1\}$ and $\{d_2\}$) and combining results agreeing on a, b. Indeed, for the given LTD of Figure 1, $D_{\leq t_1} = \{d_1\}$, $D_{\leq t_2} = \{d_2\}$ and $D = D_{\leq t_3} = D_{<t_3} = D_{\leq t_1} \cup D_{\leq t_2}$.*

The next section deals with the details of SPRIM. Before, we need a notion to talk about the result of sequences of a computation. For a node t, the Algorithm SPRIM stores tuples in a table τ_t based on a computation that depends on tuples (*originating tuples*) that are stored in the table(s) of the child nodes. In order to talk in informal explanations about properties that tuples or parts of tuples have when looking at the entire computation from the relevant leaves up to the node t in the post-order, we need a notion similar to a default theory below t for parts of tuples. Assume for now that our tuples in tables are only tuples of sets. Then, we collect recursively in pre-order along the induced subtree T' of T rooted at t a sequence s of originating tuples $(u, \vec{u_1}, \ldots, \vec{u_m})$. If the set T occurs in position i of tuple u, our notion $T^{\leq t}(s)$ takes the union over all sets $T, T_1, \ldots, T_m$ at position i in the tuples $\vec{u_1}, \ldots, \vec{u_m}$. Since a node of type *rem* will typically result in multiple originating tuples, we have multiple sequences $s_1, \ldots, s_m$ of originating tuples in general. This results in a family $\mathcal{T}^{\leq t} := \{T^{\leq t}(s) \mid s \in \{s_1, \ldots, s_m\}\}$ of such sets. However, when stating properties, we are usually only interested in the fact that each $S \in \mathcal{T}^{\leq t}$ satisfies the property. To this end, we refer to $T^{\leq t}$ as any arbitrary $S \in \mathcal{T}^{\leq t}$. Further, we let $T^{<t} := T^{\leq t} \setminus T$. The definition vacuously extends to nested tuples and families of sets. A more formal compact definition provide so-called extension pointers [2].

Example 5. *Recall the given TD in Figure 1 (right). For illustrating notation, we remove node t_2, since*

we only care about nodes t_1 and t_3 and thereby obtain a simpler TD $\mathcal{T} = (T, \chi, \delta)$ (of some simpler graph). Assume that for both nodes t in T we store a table of tuples, say of the form $\langle X, Y \rangle$, where X is a subset of the bag $\chi(t)$ and Y is a set of subsets of $\chi(t)$. Further, let the tables τ_i for the two nodes in this example be as follows: $\tau_1 := \{\vec{u_{1.1}} = \langle \{d_1\}, \{\emptyset, \{d_1\}, \{d_1, b\}\}\rangle, \vec{u_{1.2}} = \langle \{a\}, \{\{b\}\}\rangle\}$, and $\tau_3 := \{\vec{u_{3.1}} = \langle \emptyset, \{\{a\}\}\rangle\}$. Then, we let tuple $\vec{u_{3.1}}$ originate from tuple $\vec{u_{1.1}}$ of child table τ_1 and not from $\vec{u_{1.2}}$. We discuss only the Y part of tuple $\vec{u_{3.1}}$ (referred to by $Y_{3.1}$). In order to talk about any "extension" $Y_{3.1.1}^{\leq t} \supseteq Y_{3.1.1}$ of $Y_{3.1.1} = \{a\}$ in $\mathcal{T}$, we write $Y_{3.1.1}^{\leq t}$, which can be one of $\{a\}$, $\{a, d_1\}$, or $\{a, d_1, b\}$.

4 Computing Stable Default Sets

In this section, we present our table algorithm SPRIM. Therefore, let D be a given default theory and $\mathcal{T} = (T, \chi, \delta)$ a pretty LTD of $S(D)$.

Our table algorithm follows Definition 2, which consists of two parts: (i) finding sets of satisfying default sets of the default theory and (ii) generating smaller sets of conclusions for these satisfying default sets in order to invalidate subset minimality. Since, SPRIM has only a "local view" on default theory D, we are only allowed to store parts of satisfying default sets. However, we guarantee that, if for the "visible" part Z of a set of satisfying defaults for any node t of T there is no smaller set of satisfying defaults, then Z can be extended to a stable default set of $D_{<t}$. However, in general Z alone is not sufficient, we require auxiliary information to decide the satisfiability of defaults. We need a way to prove that Z witnessed a satisfying default set $Z^{\leq t}$. In particular, even though each $d \in Z^{\leq t}$ is vacuously c-satisfiable, we have to verify that each default $d \in D \setminus Z^{\leq t}$ is indeed p-satisfiable or j-satisfiable. In turn, we require a set $\mathcal{M}$ of (partial) assignments of $Z^{\leq t}$. To this end, we store in table τ_t tuples that are of the form $\langle Z, \mathcal{M}, \mathcal{P}, \mathcal{C} \rangle$, where $Z \subseteq D_t$ and $\mathcal{M} \subseteq 2^X$ for $X = \chi(t) \cap \mathrm{Vars}(D)$. The first three tuple positions cover Part (i) and can be seen as the *witness* part. The last position consists of a set of tuples $\mathcal{C} = \langle \rho, \mathcal{AC}, \mathcal{BC} \rangle$ to handle Part (ii) and can be seen as the *counter-witness* part.

In the following, we describe more details of our tuples. We call Z the *witness set*, since Z *witnesses* the existence of a satisfying default set $Z^{\leq t}$ for a *sub-theory* S. Each element M in the set $\mathcal{M}$ of *witness models* witnesses the existence of a model of $F_{\leq t} := \bigwedge_{d \in Z^{\leq t}} \gamma(d)$. For our assumed witness set Z, we require a set $\mathcal{P}$ of *witness proofs*. The set $\mathcal{P}$ consists of tuples of the form $\langle \sigma, \mathcal{A}, \mathcal{B} \rangle$, where $\sigma : D_t \to \{p, j, c\}$ and $\mathcal{A}, \mathcal{B} \subseteq 2^X$ for $X = \chi(t) \cap \mathrm{Vars}(D)$. The function σ, which we call *states function*, maps each default $d \in D_t$ to a decision state $v \in \{p, j, c\}$ representing the case where d is v-satisfiable. The set $\mathcal{A}$, which we call the *required p-assignments*, contains an assignment $A \in 2^X$ for *each* default d that is claimed to be p-satisfiable. More formally, there is an assignment $A \in \mathcal{A}$ for each default $d \in \sigma^{-1}(p) \cup D_{<t}$ where $\sigma^{\leq t}(d) = p$ such that there is an assignment $A^{\leq t}$ that satisfies $F_{\leq t} \wedge \neg\alpha(d)$. The set $\mathcal{B}$, which we call the *refuting j-assignments*, contains an assignment $B \in 2^X$ for certain defaults. Intuitively, for each $B \in \mathcal{B}$ there is a default d in the current bag $\chi(t)$ or was in a bag below t such that there is an assignment $B^{\leq t}$ where the justification is not fulfilled. More formally, there is a $B \in \mathcal{B}$ if there is an assignment $B^{\leq t}$ that satisfies $F_{\leq t} \wedge \beta(d)$ for some default $d \in \sigma^{-1}(j) \cup D_{<t}$ where $\sigma^{\leq t}(d) = j$. In the end, if Z proves the existence of a satisfying default set $Z^{\leq t}$ of theory $D_{<t}$, then there is at least one tuple $\langle \cdot, \cdot, \mathcal{B} \rangle \in \mathcal{P}$ with $\mathcal{B} = \emptyset$. Hence, we require that $\mathcal{B} = \emptyset$ in order to guarantee that each default $d \in D_{<t}$ is j-satisfiable where $\sigma^{\leq t}(d) = j$. To conclude, if table τ_n for (empty) root n contains $u = \langle Z, \cdot, \mathcal{P}, \mathcal{C} \rangle$ where $\mathcal{P}$ contains $\langle \cdot, \cdot, \emptyset \rangle$, then $Z^{\leq t}$ is a satisfying default set of the default theory D. The main aim of $\mathcal{C}$ is to invalidate the subset-minimality of $Z^{\leq t}$, and will be covered later.

Next, we briefly discuss important cases of Listing 2 for Part (i), which consists only of the first three tuple positions (colored red and green) and ignores the remaining parts of the tuple. We call the resulting table algorithm SCONS, which only concerns about computing satisfying default sets. Let $t \in T$ and $u' = \langle Z, \mathcal{M}, \mathcal{P}, \cdot \rangle$ a tuple of table τ' for a child node of t and $\langle \sigma, \mathcal{A}, \mathcal{B} \rangle$ a tuple in $\mathcal{P}$. We describe informally how we transform u' tuples into one or more tuples for the table in node t.

If t is of type *int* and a default d is introduced in t, Line 3 guesses whether d is p-satisfiable, j-satisfiable,

Listing 2 ($\star$)**:** Table algorithm $\mathsf{SPRIM}(t, \chi_t, \delta_t, D_t, \text{Child-Tabs})$.

In: Bag χ_t, label mapping δ_t, bag-theory D_t, and child tables Child-Tabs of t.
Out: Table τ_t.

1 **if** type$(t) =$ *leaf* **then** $\tau_t \leftarrow \{\langle \emptyset, \{\emptyset\}, \{\langle \emptyset, \emptyset, \emptyset \rangle\}, \emptyset \rangle\}$ `/* Abbreviations below. */`
2 **else if** type$(t) =$ *int*, $d \in D_t$ *is the introduced default, and* $\tau' \in$ *Child-Tabs* **then**
3 $\quad \tau_t \leftarrow \{\langle Z_d^+, \mathcal{M}, \mathrm{SGuess}_{d,\{c\}}(\mathcal{P}),\ \mathrm{SGuess}_{d,\{p,j,c\}}(\mathcal{C}) \cup \mathrm{SGuess}_{d,\{p,j\}}(\mathcal{P}, \mathcal{M}) \rangle,$
$\qquad\quad \langle Z, \mathcal{M}, \mathrm{SGuess}_{d,\{p,j\}}(\mathcal{P}),\ \mathrm{SGuess}_{d,\{p,j\}}(\mathcal{C}) \rangle \mid \langle Z, \mathcal{M}, \mathcal{P}, \mathcal{C} \rangle \in \tau' \}$
4 **else if** type$(t) =$ *label*, $\{(\gamma, d)\} = \delta_t$ *is the label of* t, $d \in D_t$, *and* $\tau' \in$ *Child-Tabs* **then**
5 $\quad \tau_t \leftarrow \{\langle Z, \mathrm{Mod}_{\mathcal{M}}(\gamma(d)), \mathrm{PCon}_d(\mathcal{P}),\ \mathrm{CCon}_d(\mathcal{C}) \rangle \mid \langle Z, \mathcal{M}, \mathcal{P}, \mathcal{C} \rangle \in \tau', d \in Z \} \cup$
$\qquad\quad \{\langle Z, \mathcal{M}, \mathcal{P},\ \mathcal{C} \rangle \mid \langle Z, \mathcal{M}, \mathcal{P}, \mathcal{C} \rangle \in \tau', d \notin Z \}$
6 **else if** type$(t) =$ *label*, $\{(\alpha, d)\} = \delta_t$ *is the label of* t, $d \in D_t$, *and* $\tau' \in$ *Child-Tabs* **then**
7 $\quad \tau_t \leftarrow \{\langle Z,\ \mathcal{M}, \mathrm{PPre}_d(\mathcal{P}, \mathcal{M}),\ \mathrm{CPre}_d(\mathcal{C}) \rangle \mid \langle Z, \mathcal{M}, \mathcal{P}, \mathcal{C} \rangle \in \tau' \}$
8 **else if** type$(t) =$ *label*, $\{(\beta, d)\} = \delta_t$ *is the label of* t, $d \in D_t$, *and* $\tau' \in$ *Child-Tabs* **then**
9 $\quad \tau_t \leftarrow \{\langle Z,\ \mathcal{M}, \mathrm{PJust}_d(\mathcal{P}, \mathcal{M}),\ \mathrm{PJust}_d(\mathcal{C}, \mathcal{M}) \rangle \mid \langle Z, \mathcal{M}, \mathcal{P}, \mathcal{C} \rangle \in \tau' \}$
10 **else if** type$(t) =$ *int*, $a \in \chi_t$ *is the introduced variable, and* $\tau' \in$ *Child-Tabs* **then**
11 $\quad \tau_t \leftarrow \{\langle Z, \mathcal{M} \cup \mathcal{M}_a^{\text{u}}, \mathrm{PGuess}_a(\mathcal{P}),\ \mathrm{PGuess}_a(\mathcal{C}) \rangle \mid \langle Z, \mathcal{M}, \mathcal{P}, \mathcal{C} \rangle \in \tau' \}$
12 **else if** type$(t) =$ *rem*, $d \notin D_t$ *is the removed default, and* $\tau' \in$ *Child-Tabs* **then**
13 $\quad \tau_t \leftarrow \{\langle Z_d^-, \mathcal{M}, \mathrm{SProj}_d(\mathcal{P}),\ \mathrm{SProj}_d(\mathcal{C}) \rangle \mid \langle Z, \mathcal{M}, \mathcal{P}, \mathcal{C} \rangle \in \tau' \}$
14 **else if** type$(t) =$ *rem*, $a \notin \chi_t$ *is the removed variable, and* $\tau' \in$ *Child-Tabs* **then**
15 $\quad \tau_t \leftarrow \{\langle Z, \mathcal{M}_a^{\sim}, \mathrm{AProj}_a(\mathcal{P}),\ \mathrm{AProj}_a(\mathcal{C}) \rangle \mid \langle Z, \mathcal{M}, \mathcal{P}, \mathcal{C} \rangle \in \tau' \}$
16 **else if** type$(t) =$ *join and* $\tau', \tau'' \in$ *Child-Tabs with* $\tau' \neq \tau''$ **then**
17 $\quad \tau_t \leftarrow \{\langle Z, \mathcal{M}' \cap \mathcal{M}'', \mathcal{P}' \hat{\bowtie}_{\mathcal{M}', \mathcal{M}''} \mathcal{P}'', (\mathcal{C}' \hat{\bowtie}_{\mathcal{M}', \mathcal{M}''} \mathcal{C}'') \cup (\mathcal{P}' \hat{\bowtie}_{\mathcal{M}', \mathcal{M}''} \mathcal{C}'') \cup (\mathcal{C}' \hat{\bowtie}_{\mathcal{M}', \mathcal{M}''} \mathcal{P}'') \rangle$
$\qquad\quad \mid \langle Z, \mathcal{M}', \mathcal{P}', \mathcal{C}' \rangle \in \tau', \langle Z, \mathcal{M}'', \mathcal{P}'', \mathcal{C}'' \rangle \in \tau'' \}$
18 **return** τ_t

$S_e^- := S \setminus \{e\}$, $S_e^{\sim} := \{S_e^- \mid S \in \mathcal{S}\}$, $S_e^+ := S \cup \{e\}$, and $\mathcal{S}_e^{\text{u}} := \{S_e^+ \mid S \in \mathcal{S}\}$.

or c-satisfiable. To this end, $\mathrm{SGuess}_{d,\mathcal{S}}(\mathcal{P})$ adds potential proofs to $\mathcal{P}$ where the satisfiability state of d is within $\mathcal{S}$. Lines 5, 7 and 9 cover nodes of type *label* as follows: In Line 5, if (γ, d) is the label and $\sigma(d) = c$, we enforce that each $M \in \mathcal{M}$ is also a model of $\gamma(d)$. $\mathrm{PCon}_d(\mathcal{P})$ only keeps tuples in $\mathcal{P}$ where each $A \in \mathcal{A}$ is a model of $\gamma(d)$. In Line 7, if (α, d) is the label and $\sigma(d) = p$, $\mathrm{PPre}_d(\mathcal{P}, \mathcal{M})$ enforces that each $A \in \mathcal{A}$ within $\mathcal{P}$ is a model of $\neg\alpha(d)$. In Line 9, if (β, d) is the label and $\sigma(d) = j$, $\mathrm{PJust}_d(\mathcal{P}, \mathcal{M})$ adds assignments of $\mathcal{M}$ to $\mathcal{B}$ that are also models of $\beta(d)$.

Next, we cover the case, where a variable a is introduced. In Line 11, we increase the existing witness set $M \cup \{a\}$ for each $M \in \mathcal{M}$. $\mathrm{PGuess}_a(\mathcal{P})$ works analogously for $\mathcal{B}$ and computes all potential combinations of every $A \in \mathcal{A}$, where a is either set to true or to false.

In Line 13, we remove default d from Z and $\mathrm{SProj}_d(\mathcal{P})$ removes d from the domain of the mapping σ, since d is not considered anymore. In Line 15, we remove variable a from each $M \in \mathcal{M}$ and $\mathrm{AProj}_a(\mathcal{P})$ works analogously for each assignment of $\mathcal{A}$ and $\mathcal{B}$.

Finally, if the node is of type *join*, we have a second child and its table τ'' as well as a tuple $u'' \in \tau''$. Intuitively, tuples u' and u'' represent intermediate results of two different branches in T. To combine these results, we have to join the tuples on the witness extension, witness states, and the witness models. The join operation $\bowtie$ can be seen as a combination of inner and outer joins, used in database theory [1]. Note that for instance for an assignment $B \in \mathcal{B}$ to endure within $\mathcal{P}$ of τ_t, it suffices that B is a corresponding witness model in u''.

Example 6. *Consider default theory D from Example 1 and in Figure 2 (left) pretty LTD $\mathcal{T} = (\cdot, \chi, \delta)$ of the semi-primal graph $S(D)$ and the tables $\tau_1, \ldots, \tau_{18}$ illustrating computation results obtained during post-order traversal of $\mathcal{T}$ by $\mathcal{DP}$ using* SCONS *instead of* SPRIM *in Line 3. We omit the last position of the tuples, since those are only relevant for* SPRIM*. Note that we discuss only selected cases, and we assume for presentation that each tuple in a table τ_t is identified by a number, i.e., the i-th tuple corresponds to $u_{t,i}^{\rightarrow} = \langle Z_{t,i}, \mathcal{M}_{t,i}, \mathcal{P}_{t,i}, \mathcal{C}_{t,i} \rangle$. The numbering naturally extends to sets in witness proofs and counter-witnesses.*

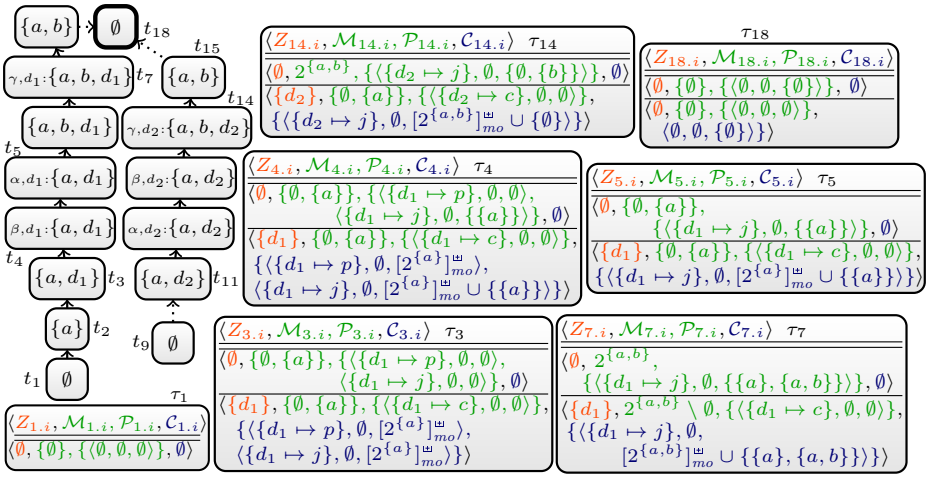

Figure 2 Selected DP tables of SPRIM for pretty LTD $\mathcal{T}$.

We obtain table $\tau_1 = \{\langle\emptyset,\{\emptyset\},\{\langle\emptyset,\emptyset,\emptyset\rangle\}\rangle\}$ *as* $\text{type}(t_1) = \textit{leaf}$ *(see Line 1). Since* $\text{type}(t_2) = \textit{int}$ *and* a *is the introduced variable, we construct table* τ_2 *from* τ_1 *by modifying* $\mathcal{M}_{2.1}$ *and* $\mathcal{P}_{2.1} = \{\langle\sigma_{1.1}, \mathcal{A}_{1.1}, \mathcal{M}_{2.1}\rangle\}$*, where* $\mathcal{M}_{2.1}$ *contains* $M_{1.1.k}$ *and* $M_{1.1.k} \cup \{a\}$ *for each* $M_{1.1.k}$ *(*$k \leq 1$*) in* τ_1*. This corresponds to a guess on* a*. Precisely,* $\mathcal{M}_{2.1} := \{\emptyset,\{a\}\}$ *(Line 11).*

Then, t_3 *introduces default* d_1*, which results in two tuples. In tuple* $u^{\rightarrow}_{3.1}$ *default* d_1 *is p-satisfiable or j-satisfiable due to* $\alpha(d_1)$ *or* $\beta(d_1)$ *(see* $\mathcal{P}_{3.1}$*, Line 3). In tuple* $u^{\rightarrow}_{3.2}$ *default* d_1 *is c-satisfiable and we have that* $Z_{3.2} = \{d_1\}$ *and* $\mathcal{P}_{3.2} = \{\{d_1 \mapsto c\},\emptyset,\emptyset\rangle\}$*.*

Node t_4 *introduces label* (β, d_1) *and modifies* $\mathcal{P}_{4.1.2}$*. In particular, it chooses among* $\mathcal{M}$ *candidates, which might contradict that* d_1 *is j-satisfiable (see Line 9). Obviously, we have that* $\mathcal{B}_{4.1.2} = \{\{a\}\}$*, since* $\beta(d_1) = a$*.*

In table τ_5*, we present the case where default* d_1 *should be p-satisfiable. In this case since* $\alpha(d_1) = \top$*, we do not find any model of* $\bot$*. In consequence, there is no corresponding successor of* $\mathcal{P}_{4.1.1}$ *in* τ_5*, i.e., in* τ_5 *it turns out that* d_1 *can not be p-satisfiable.*

Table τ_7 *concerns the conclusion* $\gamma(d_1)$ *of a default. It updates every assignment occurring in the table, such that the models satisfy* $\gamma(d_1)$ *if* d_1 *is c-satisfiable. The remaining cases work similarly.*

In the end, join node t_{16} *just combines witnesses agreeing on its content.*

Next, we briefly discuss the handling of counter-witnesses, which completes Algorithm SPRIM. The handling of counter-witnesses $\mathcal{C}$ is quite similar to the witness proofs $\mathcal{P}$. The tuples $\langle\rho, \mathcal{AC}, \mathcal{BC}\rangle \in \mathcal{C}$ consist of a states function $\rho : D_{\leq t} \mapsto \{p, j, c\}$, required p-assignments $\mathcal{AC} \subseteq 2^X$ and refuting j-assignments $\mathcal{BC} \subseteq 2^{(X \cup \{mo\})}$ for $X = \text{Vars}_{\leq t} \cap \chi(t)$. In contrast to the refuting j-assignments in $\mathcal{B}$, $\mathcal{BC}$ may in addition contain an assignment $B \in \mathcal{BC}$ with a marker mo. The marker indicates that $B^{\leq t}$ is actually not refuting, but only a model of $\gamma(d)$ for each default below t that is c-satisfiable, i.e., $\bigwedge_{d \in D_{\leq t}, \rho^{\leq t}(d) = c} \gamma(d)$. In other words, those assignments setting mo to true are the counter-witness assignments that do not refute c-assignments (comparable to witness assignments in $\mathcal{M}$ for Part (ii)).

The existence of a certain counter-witness tuple for a witness in a table τ_t establishes that the corresponding witness can *not* be extended to a stable default set of $D_{\leq t}$. In particular, there exists a stable extension for D if the table τ_n for root n contains a tuple of the form $\langle\emptyset,\{\emptyset\},\mathcal{P},\mathcal{C}\rangle$, where $\mathcal{P} \neq \emptyset$ and contains tuples of the form $\langle\cdot,\cdot,\emptyset\rangle$. Moreover, for *each* $\langle\rho, \mathcal{AC}, \mathcal{BC}\rangle \in \mathcal{C}$ there is $\emptyset \in \mathcal{BC}$ indicating a true refuting j-assignment for the empty root n. Intuitively, this establishes that there is no actual counter-witness, which contradicts that the corresponding satisfying default $Z^{\leq t}$ is subset-minimal and hence indeed a stable default set.

Due to space limitations, we omit a full description of both Parts (i) and (ii) together for our algorithm.

A major difference of Part (ii) is that we need a special function $\mathrm{CCon}_d(\mathcal{C})$ to establish that a default d is j-satisfiable, which is defined with respect to fixed set S, c.f., Case (ii) of Definition 2. Then, $\mathrm{CCon}_d(\mathcal{C})$ additionally adds potential proofs involving counter-witnesses and *mo* models, where $\rho(d) \neq c$, but $\sigma(d) = c$.

In the following, we state the correctness of the algorithm $\mathcal{DP}$.

Theorem 1 ($\star$)**.** *Given a default theory D, algorithm $\mathcal{DP}$ correctly solves* Ext.

Idea. The correctness proof of this algorithm needs to investigate each node type separately. We have to show that a tuple at a node t guarantees existence of a stable default set for a sub-theory of theory $D_{\leq t}$, which proves soundness. Conversely, one can show that each stable default set is indeed evaluated while traversing the pretty LTD, which establishes completeness. □

Next, we establish that we can extend $\mathcal{DP}$ to enumerate stable default sets. The algorithm on top of $\mathcal{DP}$ is relatively straight forward, which can be found in the appendix. The idea is to compute a first stable default set in linear time, followed by systematically enumerating subsequent solutions with linear delay. One can even further extend $\mathcal{DP}$ to solve #SE, similar to related work [9] in a slightly different context.

Theorem 2 ($\star$)**.** *Given a default theory D, algorithm* SPRIM *can be used as a preprocessing step to construct tables from which we can solve problem* EnumSE.

Idea. The correctness proof requires to extend the previous results to establish a one-to-one correspondence when traversing the tree of the TD and such that we can reconstruct each solution as well as we do not get duplicates. The proof proceeds similar to Theorem 1. □

The following theorem states that we obtain threefold exponential runtime in the treewidth.

Theorem 3 ($\star$)**.** *Algorithm $\mathcal{DP}$ runs in time $\mathcal{O}(2^{2^{2^{k+4}}} \cdot \|S(D)\|)$ for a given default theory D, where $k := tw(S(D))$ is the treewidth of semi-primal graph $S(D)$.*

5 Conclusion

In this paper, we established algorithms that operate on tree decompositions of the semi-primal graph of a given default theory. Our algorithms can be used to decide whether the default theory has a stable extension or to enumerate all stable default sets. The algorithms assume small treewidth and run in linear time and with linear delay, respectively. Even though already linear time results for checking the existence of a stable extension are known, we are able to establish runtime that is only triple exponential in the treewidth of the semi-primal graph.

In order to simplify the presentation, we mainly covered the semi-primal graph. However, we believe that our algorithms can be extended to tree decompositions of the incidence graph. Then we need additional states to handle the cases where prerequisite, justification, and conclusion do not occur together in one bag. Consequently, such an algorithm will likely be very complex. Further, we also believe that our algorithm can be extended to disjunctive defaults [13], where we have to guess which of the conclusion parts is to apply. An interesting research question is whether we can improve our runtime bounds. Still it might be worth implementing our algorithms to enumerate stable default sets for DL, as previous work showed that a relatively bad worst-case runtime may anyways lead to practical useful results [4].

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases: The Logical Level*. Addison-Wesley, Boston, MA, USA, 1st edition, 1995.

[2] B. Bliem, G. Charwat, M. Hecher, and S. Woltran. D-FLATˆ2: Subset minimization in dynamic programming on tree decompositions made easy. *FI*, 147:27–34, 2016.

[3] H. Bodlaender and A. M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008.

[4] G. Charwat and S. Woltran. Dynamic programming-based QBF solving. In *Proc. of the 4th Intl. Workshop on Quantified Boolean Formulas (QBF'16)*, pages 27–40, 2016.

[5] B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of theoretical computer science, Vol. B*, volume Formal Models and Semantics, pages 193–242. Elsevier Science Publishers, North-Holland, 1990.

[6] N. Creignou, A. Meier, J.-S. Müller, J. Schmidt, and H. Vollmer. Paradigms for parameterized enumeration. *Th. Comput. Syst.*, 60(4):737–758, 2017.

[7] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

[8] R. Downey and M. Fellows. *Fundamentals of Parameterized Complexity*. 2013.

[9] J. K. Fichte, M. Hecher, M. Morak, and S. Woltran. Answer set solving with bounded treewidth revisited. In *Proc. of the 14th Intl. Conference on LPNMR*, 2017.

[10] J. K. Fichte, M. Hecher, M. Morak, and S. Woltran. DynASP2.5: Dynamic programming on tree decompositions in action. In *Proc. of the 12th IPEC*, 2017.

[11] J. K. Fichte, A. Meier, and I. Schindler. Strong backdoors for default logic. In *Proc. of the 19th Intl. Conference on Theory and Applications of SAT (SAT'16)*, 2016.

[12] J. Flum and M. Grohe. *Parameterized Complexity Theory*, volume XIV of *Th. Comput. Sc.* Springer, Berlin, 2006.

[13] M. Gelfond, V. Lifschitz, H. Przymusinska, and M. Truszczynski. Disjunctive defaults. pages 230–237. Morgan Kaufmann, 1991.

[14] G. Gottlob. Complexity results for nonmonotonic logics. *JLC*, 2(3):397–425, 1992.

[15] J. Kneis and A. Langer. A practical approach to Courcelle's theorem. *Electronic Notes in Theoretical Computer Science*, 251:65–81, 2009.

[16] V. W. Marek and M. Truszczyński. *Nonmonotonic Logic: context-dependent reasoning*. Artificial Intelligence. Springer, Berlin, Germany, 1993.

[17] A. Meier, I. Schindler, J. Schmidt, M. Thomas, and H. Vollmer. On the parameterized complexity of non-monotonic logics. *Archive for Math. Logic*, 54(5-6):685–710, 2015.

[18] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[19] R. Reiter. A logic for default reasoning. *AIJ*, 13:81–132, Apr. 1980.

A Equivalence of stable default sets and stable extensions

We provide in the following insights on the correspondence between sets $\mathrm{SD}(D)$ and $\mathrm{SE}(D)$ for a given default theory D.

Lemma 1. *Let D be a default theory. Then,*

$$\mathrm{SE}(D) = \bigcup_{S \in SD(D)} \mathrm{Th}(\{\gamma(d) \mid d \in S\}).$$

In particular, $S \in SD(D)$ is a generating default of extension $\mathrm{Th}(\{\gamma(d) \mid d \in S\})$.

Sketch. Consider an arbitrary default theory D.
"$\Longrightarrow$": Take any extension $E \in \mathrm{SE}(D)$. Observe that E is closed unter $\mathrm{Th}(\cdot)$, i.e., $E = \mathrm{Th}(E)$. From E, we now construct set $S := \{d \in D \mid \gamma(d) \in E, \alpha(d) \in E, \neg\beta(d) \notin E\}$. Assume towards a contradiction that S is not a stable default set, so it either dissatisfies at least one default, which immediately leads to a contradiction since E is a stable extension, or S is not subset-minimal. If S is not subset-minimal, there is a smaller set $S' \subsetneq S$, which is a satisfying default set. Observe that there is at least one $d \in S \setminus S'$ where $\gamma(d) \notin \mathrm{Th}(\{\gamma(d') \mid d' \in S'\})$, since otherwise S' can not be a satisfying default set due to $\mathrm{Th}(\{\gamma(d) \mid d \in S\}) = \mathrm{Th}(\{\gamma(d') \mid d' \in S'\})$ and $S' \subsetneq S$, which results in at least one default in $S \setminus S'$ that is dissatisfied (by construction of S, c.f. Definition 2(i)). As a result, there is a smaller extension $E' \subsetneq E$, where $E' := \mathrm{Th}(\{\gamma(d') \mid d' \in S'\})$, which contradicts, once again, that E is a stable extension.
"$\Longleftarrow$": Assume any stable default set S. We define $E := \mathrm{Th}(\{\gamma(d) \mid d \in S\})$. Assume towards a contradiction that E is not stable. Obviously, by construction of E, $\Gamma(E) := E$ satisfies Definition 1(i)-(iii). It remains to show, that, indeed there is no smaller $\Gamma'(E) \subsetneq \Gamma(E)$ which also satisfies the three conditions. Assume towards a contradiction, that such a set $\Gamma'(E)$ with $\Gamma'(E) = \mathrm{Th}(\Gamma'(E))$ indeed exists. Then there is at least one default $d \in D$, such that $\gamma(d) \in \Gamma(E) \setminus \Gamma'(E)$. As a result, by construction of E, S can not be stable default set, which yields a contradiction.

□

B Auxiliary Definitions of Table algorithm SPRIM

We provide formal definitions for abbreviations that are used in algorithm SPRIM, which we explained only verbally in Section 4. We abbreviate by $\mathcal{S}_{MO}$ the set $\{S \mid S \in \mathcal{S}, mo \in S\}$ for set $\mathcal{S}$ of sets. Further, we define the abbreviation $\mathcal{S}_e^?$ for set $\mathcal{S}$ of sets as follows: $\emptyset_e^? := \{\emptyset\}$ and $\mathcal{S}_e^? := \bigcup_{S \in \mathcal{S}, S' \in (\mathcal{S} \setminus S)_e^?} \{S' \cup \{S_e^+\}, S' \cup \{S\}\}$.

$$\mathrm{cpy}_d(\mathcal{P},\pi) := \{\langle\sigma,\mathcal{A},\mathcal{B}\rangle \mid \langle\sigma,\mathcal{A},\mathcal{B}\rangle \in \mathcal{P}, \sigma(d)\neq\pi\} \tag{1}$$

$$\mathrm{SGuess}_{d,\mathcal{S}}(\mathcal{P},\mathcal{M}) := \{\langle\sigma^{+}_{d\mapsto\pi},\mathcal{A},\mathcal{M}^{+}_{mo}\cup\mathcal{B}\rangle \mid \langle\sigma,\mathcal{A},\mathcal{B}\rangle\in\mathcal{P},\pi\in\mathcal{S}\} \tag{2}$$

$$\mathrm{SGuess}_{d,\mathcal{S}}(\mathcal{P}) := \mathrm{SGuess}_{d,\mathcal{S}}(\mathcal{P},\emptyset) \tag{3}$$

$$\mathrm{PCon}_d(\mathcal{P}) := \{\langle\sigma,\mathcal{A},\mathrm{Mod}_{\mathcal{B}}(\gamma(d))\rangle \mid \langle\sigma,\mathcal{A},\mathcal{B}\rangle\in\mathcal{P},\sigma(d)=c, \mathcal{A}=\mathrm{Mod}_{\mathcal{A}}(\gamma(d))\} \tag{4}$$

$$\mathrm{CCon}_d(\mathcal{C}) := \mathrm{PCon}_d(\mathcal{C})\cup\{\langle\rho,\mathcal{AC},\mathcal{BC}_{MO}\cup\mathrm{Mod}_{\mathcal{BC}}(\gamma(d))\rangle \mid \langle\rho,\mathcal{AC},\mathcal{BC}\rangle\in\mathcal{C},\rho(d)\neq c\} \tag{5}$$

$$\mathrm{PPre}_d(\mathcal{P},\mathcal{M}) := \mathrm{cpy}_d(\mathcal{P},p)\cup\{\langle\sigma,\mathcal{A}\cup\mathcal{A}',\mathcal{B}\rangle \mid \langle\sigma,\mathcal{A},\mathcal{B}\rangle\in\mathcal{P}, \sigma(d)=p,\mathcal{A}'\in\mathrm{Mod}_{\mathcal{M}\cup\mathcal{B}^{\sim}_{mo}}(\neg\alpha(d))\} \tag{6}$$

$$\mathrm{CPre}_d(\mathcal{C}) := \mathrm{PPre}_d(\mathcal{C},\emptyset) \tag{7}$$

$$\mathrm{PJust}_d(\mathcal{P},\mathcal{M}) := \mathrm{cpy}_d(\mathcal{P},j)\cup\{\langle\sigma,\mathcal{A},\mathcal{B}\cup[\mathrm{Mod}_{\mathcal{M}}(\beta(d))]^{\sim}_{mo}\rangle \mid \langle\sigma,\mathcal{A},\mathcal{B}\rangle\in\mathcal{P},\sigma(d)=j\} \tag{8}$$

$$\mathrm{PGuess}_a(\mathcal{P}) := \{\langle\sigma,\mathcal{A}',\mathcal{B}\cup\mathcal{B}^{\sqcup}_a\rangle \mid \mathcal{A}'\in\mathcal{A}^{?}_a,\langle\sigma,\mathcal{A},\mathcal{B}\rangle\in\mathcal{P}\} \tag{9}$$

$$\mathrm{SProj}_d(\mathcal{P}) := \{\langle\sigma\setminus\{d\mapsto p,d\mapsto j,d\mapsto c\},\mathcal{A},\mathcal{B}\rangle \mid \langle\sigma,\mathcal{A},\mathcal{B}\rangle\in\mathcal{P}\} \tag{10}$$

$$\mathrm{AProj}_a(\mathcal{P}) := \{\langle\sigma,\mathcal{A}^{\sim}_a,\mathcal{B}^{\sim}_a\rangle \mid \langle\sigma,\mathcal{A},\mathcal{B}\rangle\in\mathcal{P}\} \tag{11}$$

$$\mathcal{M}'\bowtie\mathcal{M}'' := \{M'\cup M'' \mid M'\in\mathcal{M}',M''\in\mathcal{M}'',M'\cap[\chi_t]^{+}_{mo}= M''\cap[\chi_t]^{+}_{mo}\} \tag{12}$$

$$\mathcal{B}'\bowtie_{\mathcal{M}',\mathcal{M}''}\mathcal{B}'' := [\mathcal{B}'\bowtie(\mathcal{B}''\cup\mathcal{M}'')]\cup[(\mathcal{B}'\cup\mathcal{M}')\bowtie\mathcal{B}''] \tag{13}$$

$$\mathcal{P}'\hat{\bowtie}_{\mathcal{M}',\mathcal{M}''}\mathcal{P}'' := \{\langle\sigma,\mathcal{AR},\mathcal{B}'\bowtie_{\mathcal{M}',\mathcal{M}''}\mathcal{B}''\rangle \mid \langle\sigma,\mathcal{A}',\mathcal{B}'\rangle\in\mathcal{P}', \langle\sigma,\mathcal{A}'',\mathcal{B}''\rangle\in\mathcal{P}'',\mathcal{AR}=\mathcal{A}'\bowtie(\mathcal{A}''\cup\mathcal{M}''\cup[\mathcal{B}'']^{\sim}_{mo}), \mathcal{A}'\cup\mathcal{A}''\subseteq\mathcal{AR}\}\cup\{\langle\sigma,\mathcal{RA},\mathcal{B}'\bowtie_{\mathcal{M}',\mathcal{M}''}\mathcal{B}''\rangle \mid \langle\sigma,\mathcal{A}',\mathcal{B}'\rangle\in\mathcal{P}',\langle\sigma,\mathcal{A}'',\mathcal{B}''\rangle\in\mathcal{P}'', \mathcal{RA}=\mathcal{A}''\bowtie(\mathcal{A}'\cup\mathcal{M}'\cup[\mathcal{B}']^{\sim}_{mo}),\mathcal{A}'\cup\mathcal{A}''\subseteq\mathcal{AR}\} \tag{14, 15}$$

C Proof of Correctness

Before we provide more insights on the correctness of our algorithms, we require some missing auxiliary definitions.

Bag-default parts. Consider an LTD (T,χ,δ) of the graph $S(D)$ of a given default theory D. The set $\mathrm{Vars}_{\leq t} := \{v \mid v\in\mathrm{Vars}(D)\cap\chi(t'),t'\in\text{post-order}(T,t)\}$ is called *variables below* t. Further, the *bag-default parts* for prerequisite, justification, or conclusion $f\in\{\alpha,\beta,\gamma\}$ contain $f_t := \{f(d) \mid (f,d)\in\delta(t)\}$. We naturally extend the definition of the bag-default parts to the respective default parts below t (analogously to our definitions for default theory below t), i.e., we also use $\alpha_{\leq t},\beta_{\leq t}$, and $\gamma_{\leq t}$.

Further, we define mapping $\Gamma_t : 2^{\gamma(D_{\leq t})}\to 2^{\gamma_{\leq t}}$ by $\Gamma_t[E] := E\cap\gamma_{\leq t}$.

Example 7. *Recall the LTD of Figure 2. Observe that* $\mathrm{Vars}_{\leq t_6}=\mathrm{Vars}(D)$, $\alpha_{\leq t_6}=\{\alpha(d_1)\}$ *and* $\gamma_{\leq t_{16}}=\{\gamma(d_1),\gamma(d_2)\}$.

We employ the correctness argument using the notions of (i) *partial solutions* consisting of *partial extensions* and the notion of (ii) *local partial solutions*.

Definition 3. *Let D be a default theory, $\mathcal{T} = (T, \chi, \delta)$ be an LTD of the semi-primal graph $S(D)$ of D, where $T = (N, \cdot, \cdot)$, and $t \in N$ be a node. Further, let $\emptyset \subsetneq \mathcal{B} \subseteq 2^{\mathrm{Vars}_{\leq t} \cup \{mo\}}$, $\mathcal{A} \subseteq 2^{\mathcal{B}\widetilde{}_{mo}}$, $\sigma : D_{\leq t} \to \{p, j, c\}$, $E \supseteq \gamma(Z)$, where $Z := \sigma^{-1}(c)$. The tuple $(\sigma, \mathcal{A}, \mathcal{B})$ is a* partial extension under E for t *if the following conditions hold:*

1. *Z is a set of satisfying defaults of $D_{<t} \setminus [\{d \in D_{<t} \mid \sigma(d) = j, \exists B \in \mathcal{B} : B \vDash \Gamma_t[E] \wedge \beta(d)\}]$,*
2. *$\mathcal{A}$ is a set such that:*
   (a) *$|\mathcal{A}| \leq |\sigma^{-1}(p)| - 1$,*
   (b) *$\exists d \in D_{\leq t} : \sigma(d) = p, \alpha(d) \in \alpha_{\leq t}, A \vDash \Gamma_t[\gamma(Z)] \wedge \neg\alpha(d)$ for every $A \in \mathcal{A}$,*
   (c) *$\exists A \in \mathcal{A} : A \vDash \Gamma_t[\gamma(Z)] \wedge \neg\alpha(d) \Longleftarrow \sigma(d) = p$ for every $d \in D_{\leq t}$ such that $\alpha(d) \in \alpha_{\leq t}$; and*
3. *$\mathcal{B}$ is the largest set such that:*
   (a) *$B \vDash \Gamma_t[\gamma(Z)]$ for every $B \in \mathcal{B}$,*
   (b) *$\exists d \in D_{\leq t} : \sigma(d) = j, \beta(d) \in \beta_{\leq t}, B \vDash \Gamma_t[E] \wedge \beta(d)$ for every $B \in \mathcal{B}$ where $mo \notin B$.*

Definition 4. *Let D be a default theory, $\mathcal{T} = (T, \chi, \delta)$ where $T = (N, \cdot, n)$ be an LTD of $S(D)$, and $t \in N$ be a node. A* partial solution for t *is a tuple $(Z, \mathcal{M}, \mathcal{P}, \mathcal{C})$ where $Z \subseteq D_{\leq t}$, and $\mathcal{P}$ is the largest set of tuples such that each $(\sigma, \mathcal{A}, \mathcal{B}) \in \mathcal{P}$ is a partial extension under $\gamma(Z)$ with $\mathcal{B}_{MO} = \emptyset$ and $Z = \sigma^{-1}(c)$. Moreover, $\mathcal{C}$ is the largest set of tuples such that for each $(\rho, \mathcal{AC}, \mathcal{BC}) \in \mathcal{C}$, we have that $(\rho, \mathcal{AC}, \mathcal{BC})$ is a partial extension under $\gamma(Z)$ with $\rho^{-1}(c) \subsetneq \sigma^{-1}(c)$. Finally, $\mathcal{M} \subseteq 2^{\mathrm{Vars}_{\leq t}}$ is the largest set with $M \vDash \Gamma_t[\gamma(Z)]$ for each $M \in \mathcal{M}$.*

The following lemma establishes correspondence between stable default sets and partial solutions.

Lemma 2. *Let D be a default theory, $\mathcal{T} = (T, \chi, \delta)$ be an LTD of the semi-primal graph $S(D)$, where $T = (\cdot, \cdot, n)$, and $\chi(n) = \emptyset$. Then, there exists a stable default set Z' for D if and only if there exists a partial solution $u = (Z', \mathcal{M}, \mathcal{P}, \mathcal{C})$ for root n with at least one tuple $\langle \sigma, \mathcal{A}, \mathcal{B} \rangle \in \mathcal{P}$ where $\mathcal{B} = \emptyset$, where $\mathcal{C}$ is of the following form: For each $(\rho, \mathcal{AC}, \mathcal{BC}) \in \mathcal{C}$, $\mathcal{BC}_{MO} \neq \mathcal{BC}$.*

Sketch. Given a stable default set Z' of D we construct $u = (Z', \mathcal{M}, \mathcal{P}, \mathcal{C})$ where we generate every potential $\sigma : D \to \{p, j, c\}$ such that $\sigma(d) = c$ for $d \in Z'$ as follows. For $d \in D \setminus Z'$, we are allowed to set $\sigma(d) := p$ if $\gamma(Z') \wedge \neg\alpha(d)$ is satisfiable and $\sigma(d) := j$ if $\gamma(Z') \wedge \beta(d)$ is unsatisfiable.

For each of this functions σ, we require $\langle \sigma, \mathcal{A}, \emptyset \rangle \in \mathcal{P}$, where $\mathcal{A} \subseteq 2^{\mathrm{Vars}(D)}$ is the smallest set with $|\mathcal{A}| \leq |\sigma^{-1}(\alpha)| - 1$ such that for all $d \in \sigma^{-1}(\alpha)$ there is at least one $A \in \mathcal{A}$ with $A \vDash \gamma(Z') \wedge \neg\alpha(d)$.

Moreover, we define set $\mathcal{M} := \mathrm{Mod}_{2^{\mathrm{Vars}(D)}}(\bigwedge_{d \in Z'} \gamma(d))$, in order for u to be a partial solution for n (see Definition 4). We construct $\mathcal{C}$, consisting of partial solutions $(\rho, \mathcal{AC}, \mathcal{BC})$ where we use every potential state function ρ with $\rho^{-1}(c) \subsetneq \sigma^{-1}(c)$. For this, let $Z := \rho^{-1}(c)$. For the defaults d with $\rho(d) \neq c$, i.e., defaults d that are p-satisfiable or j-satisfiable, we also set their state $\rho(d)$ to α or β, respectively (analogous to above). Finally, we define set $\mathcal{BC} := [\mathrm{Mod}_{2^{\mathrm{Vars}(D)}}(\bigwedge_{d \in Z} \gamma(d))^{\boxplus}_{mo} \cup [\bigcup_{d:\rho(d)=j} \mathrm{Mod}_{2^{\mathrm{Vars}(D)}}([\bigwedge_{d \in Z'} \gamma(d)] \wedge \beta(d))]$, and $\mathcal{AC} \subseteq 2^{\mathrm{Vars}(D)}$ as the smallest set such that $|\mathcal{AC}| \leq |\rho^{-1}(p)| - 1$ and for all $d \in \rho^{-1}(p)$, there is at least one $AC \in \mathcal{AC}$ with $AC \vDash \gamma(Z) \wedge \neg\alpha(d)$ according to Definition 3.

For the other direction, Definitions 3 and 4 guarantee that Z' is a stable extension if there exists such a partial solution u. In consequence, the lemma holds. □

Next, we require the notion of local partial solutions corresponding to the tuples obtained in Algorithm 2.

Definition 5. *Let D be a default theory, $\mathcal{T} = (T, \chi, \delta)$ an LTD of the semi-primal graph $S(D)$, where $T = (N, \cdot, n)$, and $t \in N$ be a node. A tuple $(\sigma, \mathcal{A}, \mathcal{B})$ is a* local partial solution part *of partial solution $(\hat{\sigma}, \hat{\mathcal{A}}, \hat{\mathcal{B}})$ for t if*

1. $\sigma = \hat{\sigma} \cap (\chi(t) \times \{p, j, c\})$,

2. $\mathcal{A} = \hat{\mathcal{A}}_t$, *and*

3. $\mathcal{B} = \hat{\mathcal{B}}_t$, *where* $\mathcal{S}_t := \{S \cap (\chi(t) \cup \{mo\}) \mid S \in \mathcal{S}\}$.

Definition 6. *Let D be a default theory, $\mathcal{T} = (T, \chi, \delta)$ an LTD of the semi-primal graph $S(D)$, where $T = (N, \cdot, n)$, and $t \in N$ be a node. A tuple $u = \langle Z, \mathcal{M}, \mathcal{P}, \mathcal{C} \rangle$ is a* local partial solution *for t if there exists a partial solution $\hat{u} = (\hat{Z}, \hat{\mathcal{M}}, \hat{\mathcal{P}}, \hat{\mathcal{C}})$ for t such that the following conditions hold: (1) $Z = \hat{Z} \cap 2^{D_t}$, (2) $\mathcal{M} = \hat{\mathcal{M}}_t$, (3) $\mathcal{P}$ is the smallest set containing local partial solution part $(\sigma, \mathcal{A}, \mathcal{B})$ for each $(\hat{\sigma}, \hat{\mathcal{A}}, \hat{\mathcal{B}}) \in \hat{\mathcal{P}}$, and (4) $\mathcal{C}$ is the smallest set with local partial solution part $(\rho, \mathcal{AC}, \mathcal{BC}) \in \mathcal{C}$ for each $(\hat{\rho}, \hat{\mathcal{AC}}, \hat{\mathcal{BC}}) \in \hat{\mathcal{C}}$.*

We denote by $\hat{u}^t$ the local partial solution u for t given partial solution $\hat{u}$.

The following proposition provides justification that it suffices to store local partial solutions instead of partial solutions for a node $t \in N$.

Lemma 3. *Let D be a default theory, $\mathcal{T} = (T, \chi, \delta)$ an LTD of $S(D)$, where $T = (N, \cdot, n)$, and $\chi(n) = \emptyset$. Then, there exists a stable default set set for D if and only if there exists a local partial solution of the form $\langle \emptyset, \{\emptyset\}, \mathcal{P}, \mathcal{C} \rangle$ for the root $n \in N$ with at least one tuple of the form $\langle \sigma, \mathcal{A}, \emptyset \rangle \in \mathcal{P}$. Moreover, for each $\langle \rho, \mathcal{AC}, \mathcal{BC} \rangle$ in $\mathcal{C}$, $\mathcal{BC}_{MO} \neq \mathcal{BC}$.*

Proof. Since $\chi(n) = \emptyset$, every partial solution for the root n is an extension of the local partial solution u for the root $n \in N$ according to Definition 6. By Lemma 2, we obtain that the lemma is true. □

In the following, we abbreviate variables occurring in bag $\chi(t)$ by Vars_t, i.e., $\text{Vars}_t := \chi(t) \setminus D_t$.

Proposition 1 (Soundness)**.** *Let D be a default theory, $\mathcal{T} = (T, \chi, \delta)$ an LTD of the semi-primal graph $S(D)$, where $T = (N, \cdot, \cdot)$, and $t \in N$ a node. Given a local partial solution u' of child table τ' (or local partial solution u' of table τ' and local partial solution u'' of table τ''), each tuple u of table τ_t constructed using table algorithm* SPRIM *is also a local partial solution.*

Proof. Let u' be a local partial solution for $t' \in N$ and u a tuple for node $t \in N$ such that u was derived from u' using table algorithm SPRIM. Hence, node t' is the only child of t and t is either removal or introduce node.

Assume that t is a removal node and $d \in D_{t'} \setminus D_t$ for some default d. Observe that for $u = \langle Z, \mathcal{M}, \mathcal{P}, \mathcal{C} \rangle$ and $u' = \langle Z', \mathcal{M}, \mathcal{P}', \mathcal{C}' \rangle$, sets $\mathcal{A}$ and $\mathcal{B}$ are equal, i.e., $\langle \cdot, \mathcal{A}, \mathcal{B} \rangle \in \mathcal{P} \iff \langle \cdot, \mathcal{A}, \mathcal{B} \rangle \in \mathcal{P}'$ and $\langle \cdot, \mathcal{A}, \mathcal{B} \rangle \in \mathcal{C} \iff \langle \cdot, \mathcal{A}, \mathcal{B} \rangle \in \mathcal{C}'$. Since u' is a local partial solution, there exists a partial solution $\hat{u}'$ of t', satisfying the conditions of Definition 6. Then, $\hat{u}'$ is also a partial solution for node t, since it satisfies all conditions of Definitions 3 and 4. Finally, note that $u = (\hat{u}')^t$ since the projection of $\hat{u}'$ to the bag $\chi(t)$ is u itself. In consequence, the tuple u is a local partial solution.

For $a \in \text{Vars}_{t'} \setminus \text{Vars}_t$ as well as for introduce nodes, we can analogously check the proposition.

Next, assume that t is a join node. Therefore, let u' and u'' be local partial solutions for $t', t'' \in N$, respectively, and u be a tuple for node $t \in N$ such that u can be derived using both u' and u'' in accordance with the SPRIM algorithm. Since u' and u'' are local partial solutions, there exists partial solution $\hat{u}' = (\hat{Z}', \hat{\mathcal{M}}', \hat{\mathcal{P}}', \hat{\mathcal{C}}')$ for node t' and partial solution $\hat{u}'' = (\hat{Z}'', \hat{\mathcal{M}}'', \hat{\mathcal{P}}'', \hat{\mathcal{C}}'')$ for node t''. Using these two partial solutions, we can construct $\hat{u} = (\hat{Z}' \cup \hat{Z}'', \hat{\mathcal{M}}' \bowtie \hat{\mathcal{M}}'', \hat{\mathcal{P}}' \hat{\bowtie}_{\hat{\mathcal{M}}', \hat{\mathcal{M}}''} \hat{\mathcal{P}}'', (\hat{\mathcal{C}}' \hat{\bowtie}_{\hat{\mathcal{M}}', \hat{\mathcal{M}}''} \hat{\mathcal{C}}'') \cup (\hat{\mathcal{P}}' \hat{\bowtie}_{\hat{\mathcal{M}}', \hat{\mathcal{M}}''} \hat{\mathcal{C}}'') \cup (\hat{\mathcal{C}}' \hat{\bowtie}_{\hat{\mathcal{M}}', \hat{\mathcal{M}}''} \hat{\mathcal{P}}''))$ where for $\bowtie (\cdot, \cdot)$ and $\hat{\bowtie}(\cdot, \cdot)$ we refer to Listing 2. Then, we check all conditions of Definitions 3 and 4 in order to verify that $\hat{u}$ is a partial solution for t. Moreover, the projection $\hat{u}^t$ of $\hat{u}$ to the bag $\chi(t)$ is exactly u by construction and hence, $u = \hat{u}^t$ is a local partial solution.

Since one can provide similar arguments for each node type, we established soundness in terms of the statement of the proposition.

□

Proposition 2 (Completeness)**.** *Let D be a default theory, $\mathcal{T} = (T, \chi, \delta)$ where $T = (N, \cdot, \cdot)$ be an LTD of $S(D)$ and $t \in N$ be a node. Given a local partial solution u of table τ_t, either t is a leaf node, or there exists a local partial solution u' of child table τ' (or local partial solution u' of table τ' and local partial solution u'' of table τ'') such that u can be constructed by u' (or u' and u'', respectively) and using table algorithm* SPRIM*.*

Proof. Let $t \in N$ be a removal node and $d \in D_{t'} \setminus D_t$ with child node $t' \in N$. We show that there exists a tuple u' in table $\tau_{t'}$ for node t' such that u can be constructed using u' by SPRIM (Listing 2). Since u is a local partial solution, there exists a partial solution $\hat{u} = (\hat{Z}, \hat{\mathcal{M}}, \hat{\mathcal{P}}, \hat{\mathcal{C}})$ for node t, satisfying the conditions of Definition 6. It is easy to see that $\hat{u}$ is also a partial solution for t' and we define $u' := \hat{u}^{t'}$, which is the projection of $\hat{u}$ onto the bag of t'. Apparently, the tuple u' is a local partial solution for node t' according to Definition 6. Then, u can be derived using SPRIM algorithm and u'. By similar arguments, we establish the proposition for $a \in \mathrm{Vars}_{t'} \setminus \mathrm{Vars}_t$ and the remaining node types. Hence, the propositions sustains. □

Now, we are in the situation to prove Theorem 1, which states that we can decide the problem Ext by means of Algorithm $\mathcal{DP}$.

Theorem 1. *Given a default theory D, the algorithm $\mathcal{DP}$ correctly solves* Ext*.*

Proof. We first show soundness. Let $\mathcal{T} = (T, \chi, \delta)$ be the given LTD, where $T = (N, \cdot, n)$. By Lemma 3 we know that there is a stable default set if and only if there exists a local partial solution for the root n. Note that the tuple is by construction of the form $\langle \emptyset, \{\emptyset\}, \mathcal{P}, \mathcal{C} \rangle$, where $\mathcal{P} \neq \emptyset$ can contain a combination of the following tuples $\langle \emptyset, \emptyset, \emptyset \rangle$, $\langle \emptyset, \{\emptyset\}, \emptyset \rangle$. For each $\langle \rho, \mathcal{AC}, \mathcal{BC} \rangle \in \mathcal{C}$, we have $\mathcal{BC}_{MO} \neq \mathcal{BC}$. In total, this results in 16 possible tuples, since $\mathcal{C} \subseteq 2^C$ can contain any combination (4 many) of C, where $C = \{\langle \emptyset, \emptyset, \{\emptyset, \{mo\}\} \rangle, \langle \emptyset, \{\emptyset\}, \{\emptyset, \{mo\}\} \rangle\}$. Hence, we proceed by induction starting from the leaf nodes in order to end up with such a tuple at the root node n. In fact, the tuple $\langle \emptyset, \{\emptyset\}, \{\langle \emptyset, \emptyset, \emptyset \rangle\}, \emptyset \rangle$ is trivially a partial solution for (empty) leaf nodes by Definitions 3 and 4 and also a local partial solution of $\langle \emptyset, \{\emptyset\}, \{\langle \emptyset, \emptyset, \emptyset \rangle\}, \emptyset \rangle$ by Definition 6. We already established the induction step in Proposition 1. Hence, when we reach the root n, when traversing the TD in post-order by Algorithm $\mathcal{DP}$, we obtain only valid tuples inbetween and a tuple of the form discussed above in the table of the root n witnesses an answer set.

Next, we establish completeness by induction starting from the root n. Let therefore, $\hat{Z}$ be an arbitrary stable default set of D. By Lemma 3, we know that for the root n there exists a local partial solution of the discussed form $\langle \emptyset, \{\emptyset\}, \mathcal{P}, \mathcal{C} \rangle$ for some partial solution $\langle \hat{Z}, \hat{\mathcal{M}}, \hat{\mathcal{P}}, \hat{\mathcal{C}} \rangle$. We already established the induction step in Proposition 2. Hence, we obtain some (corresponding) tuples for every node t. Finally, stopping at the leaves n. In consequence, we have shown both soundness and completeness resulting in the fact that Theorem 1 is true. □

Proposition 3 (Completeness for Enumeration)**.** *Let D be a default theory, $\mathcal{T} = (T, \chi, \delta)$ where $T = (N, \cdot, \cdot)$ be an LTD of $S(D)$ and $t \in N$ be a node. Given a partial solution $\hat{u}$ and the corresponding local partial solution $u = \hat{u}^t$ for table τ_t, either t is a leaf node, or there exists a local partial solution u' of child table τ' (or local partial solution u' of table τ' and local partial solution u'' of table τ'') such that u can be constructed by u' (or u' and u'', respectively) and using table algorithm* SPRIM*.*

Idea. The correctness proof requires to extend the previous results to establish a one-to-one correspondence when traversing the tree of the TD and such that we can reconstruct each solution as well as we do not get duplicates. The result then follows from the proof for completeness (see Proposition 2). □

Theorem 2. *Given a default theory D, the algorithm $\mathcal{DP}$ can be used as a preprocessing step to construct tables from which we can correctly solve the problem* EnumSE*. More precisely, this is solved by first running Algorithm $\mathcal{DP}$, constructing the $\prec$-smallest solution $\mathcal{S}$, and then running Algorithm $\mathcal{NSD}_{\prec}(\mathcal{T}, \mathcal{S})$ on the resulting tables of Algorithm $\mathcal{DP}$ until $\mathcal{NSD}_{\prec}(\mathcal{T}, \mathcal{S})$ returns "undefined".*

For showing the theorem, we require the following three results.

Listing 3: Algorithm $\mathcal{NSD}_{\prec}(\mathcal{T}, \mathcal{S})$ for computing the next stable default set of $\mathcal{S}$.

In: TD $\mathcal{T} = (T, \cdot, \cdot)$ with $T = (N, \cdot, n)$, solution tuples $\mathcal{S}$, total ordering $\prec$ of $\text{orig}_{\cdot}(\cdot)$.
Out: The next solution tuples of $\mathcal{S}$ using $\prec$.

```
1  Tables[·] ← DP(T)
2  for iterate t in post-order(T,n) do
3      Child-Tabs := {Tables[t'] | t' is a child of t in T}
4      t̂ := parent of t
5      S[t] ← direct successor s' ≻ S[t] in orig_t̂(S[t̂])
6      if S[t] defined then
7          for iterate t' in Child-Tabs do
8              for iterate t'' in pre-order(T,t') do
9                  t̂'' := parent of t''
10                 S[t''] ← ≺-smallest element in orig_t̂''(S[t̂''])
11     return S
12 return undefined
```

Observation 1. *Let D be a default theory, $\mathcal{T} = (T, \chi, \delta)$ where $T = (N, \cdot, \cdot)$ be an LTD of $S(D)$ and $t \in N$ be a node. Then, for each partial solution $u = \langle Z, \mathcal{M}, \mathcal{P}, \mathcal{C} \rangle$ for t, $\mathcal{M}, \mathcal{P}$ and $\mathcal{C}$ are functional dependent from Z, i.e., for any partial solution $u' = \langle Z, \mathcal{M}', \mathcal{P}', \mathcal{C}' \rangle$ for t, we have $u = u'$.*

Proof. The claim immediately follows from Definition 4. □

Lemma 4. *Let D be a default theory, $\mathcal{T} = (T, \chi, \delta)$ with $T = (N, \cdot, \cdot)$ be an LTD of $S(D)$, and Z be a stable default set. Then, there is a unique set of tuples S, containing exactly one tuple per node $t \in N$ containing only local partial solutions of the unique partial solution for Z.*

Proof. By Observation 1, given Z, we can construct one unique partial solution $\hat{u} = \langle Z, \mathcal{M}, \mathcal{P}, \mathcal{C} \rangle$ for n. We then define the set S by $S := \bigcup_{t \in N} \{\hat{u}^t\}$. Assume that there is a different set $S' \neq S$ containing also exactly one tuple per node $t \in N$. Then there is at least one node $t \in N$, for which the corresponding tuples $u \in S, u' \in S'$ differ ($u \neq u'$), since $\hat{u}$ is unique and the computation $\hat{u}^t$ is defined in a deterministic, functional way (see Definition 6). Hence, either $\hat{u}^t \neq u$ or $\hat{u}^t \neq u'$, leading to the claim. □

Proposition 4. *Let D be a default theory, $\mathcal{T} = (T, \chi, \delta)$ with $T = (N, \cdot, \cdot)$ be an LTD of $S(D)$, and Z be a stable default set. Moreover, let S be the unique set of tuples, containing exactly one tuple per node $t \in N$ and containing only local partial solutions of the unique partial solution for Z. Given S, and tables of Algorithm* SPRIM*, one can compute in time $\mathcal{O}(\|D\|)$ a stable default set Z' with $Z' \neq Z$, assuming one can get for a specific tuple u for node t its corresponding $\prec$-ordered predecessor tuple set $\text{orig}_t(u)$ of tuples in the child node(s) of t in constant time.*

Proof. Note that with Z, it is easy to determine, which element of S belongs to which node t in T, hence, we can construct a mapping $\mathcal{S} : N \to S$. With $\mathcal{S}$, we can easily apply algorithm $\mathcal{NSD}$, which is given in Listing 3, in order to construct a different solution $\mathcal{S}'$ in a systematic way with linear time delay, since $\mathcal{T}$ is nice. □

of Theorem 2 (Sketch). First, we construct an LTD $\mathcal{T} = (T, \chi, \delta)$ with $T = (N, n)$ for graph $S(D)$. Then we run our algorithm $\mathcal{DP}$ and get tables for each TD node. In order to enumerate all the stable default sets, we investigate each of these tuple, which lead to a valid stable default set (see proof of Theorem 1). For each of these tuples (if exist), we construct a first solution S, if exist, (as done in Lines 7 to 10 of Listing 3, for the root n) using $\text{orig}_t(\cdot)$, and total order $\prec$. Thereby, we keep track of which tuple in S belongs to which node, resulting in the mapping $\mathcal{S}$ (see proof of Proposition 4). Note that $\text{orig}_t(\cdot)$ and $\prec$ can easily be provided by remembering for each tuple an ordered set of predecessor tuple sets during construction (using

table algorithm SPRIM). Now, we call algorithm $\mathcal{NSD}_{\prec}(\mathcal{T}, \mathcal{S})$ multiple times, by outputting and passing the result again as argument, until the return value is undefined, enumerating solutions in a systematic way. Using correctness results (by Theorem 1), and completeness result for enumeration by Proposition 3, we obtain only valid solution sets, which directly represent stable default sets and, in particular, we do not miss a single one. Observe, that we do not get duplicates (see Lemma 4). □

D Proof of Runtime Guarantees

Theorem 3. *Given a default theory D, the algorithm $\mathcal{DP}$ and runs in time $\mathcal{O}(2^{2^{2^{k+4}}} \cdot \|S(D)\|)$, where $k := tw(S(D))$ is the treewidth of the semi-primal graph $S(D)$.*

First, we give a proposition on worst-case space requirements in tables for the nodes of our algorithm.

Proposition 5. *Given a default theory D, an LTD $\mathcal{T} = (T, \chi, \delta)$ with $T = (N, \cdot, \cdot)$ of the semi-primal graph $S(D)$, and a node $t \in N$. Then, there are at most $2^{k+1} \cdot 2^{2^{k+1}} \cdot 2^{2\cdot(3^{k+1}\cdot 2^{2^{k+2}})}$ tuples in τ_t using algorithm $\mathcal{DP}$ for width k of $\mathcal{T}$.*

Sketch. Let D be the given default theory, $\mathcal{T} = (T, \chi, \delta)$ an LTD of the semi-primal graph $S(D)$, where $T = (N, \cdot, \cdot)$, and $t \in N$ a node of the TD. Then, by definition of a decomposition of the semi-primal graph for each node $t \in N$, we have $|\chi(t)| - 1 \leq k$. In consequence, we can have at most 2^{k+1} many witness defaults and $2^{2^{k+1}}$ many witnesses models. Each set $\mathcal{P}$ may contain a set of witness proof tuples of the form $\langle \sigma, \mathcal{A}, \mathcal{B} \rangle$, with at most 3^{k+1} many witness state σ mappings, $2^{2^{k+1}}$ many backfire witness models $\mathcal{B}$, and $2^{2^{k+1}}$ many required witnesses model sets. In the end, we need to distinguish $2^{k+1} \cdot 2^{2^{k+1}} \cdot 2^{(3^{k+1}\cdot 2^{2^{k+2}})}$ different witnesses of a tuple in the table τ_t for node t. For each witness, we can have at most $2^{(3^{k+1}\cdot 2^{2^{k+2}})}$ many counter-witnesses per witness default, witness models, and required witness model sets. Therefore, there are at most $2^{k+1} \cdot 2^{2^{k+1}} \cdot 2^{2\cdot(3^{k+1}\cdot 2^{2^{k+2}})}$ tuples in table τ_t for node t. In consequence, we established the proposition. □

of Theorem 3. Let D be a default theory, $S(D) = (V, \cdot)$ its semi-primal graph, and k be the treewidth of $S(D)$. Then, we can compute in time $2^{\mathcal{O}(k^3)} \cdot |V|$ an LTD of width at most k [3]. We take such a TD and compute in linear time a nice TD [3]. Let $\mathcal{T} = (T, \chi, \delta)$ be such a pretty LTD with $T = (N, \cdot, \cdot)$. Since the number of nodes in N is linear in the graph size and since for every node $t \in N$ the table τ_t is bounded by $2^{k+1} \cdot 2^{2^{k+1}} \cdot 2^{2\cdot(3^{k+1}\cdot 2^{2^{k+2}})}$ according to Proposition 5, we obtain a running time of $\mathcal{O}(2^{2^{2^{k+4}}} \cdot \|S(D)\|)$. Consequently, the theorem sustains. □