Prolog technology for default reasoning: proof theory and compilation techniques

Torsten Schaub a,*, Stefan Brüning b,1

a Institut für Informatik, Universität Potsdam, Postfach 60 15 53, D-14415 Potsdam, Germany
b TLC GmbH, Hahnstraße 43a, D-60528 Frankfurt, Germany

Received 24 February 1997; received in revised form 3 June 1998

Abstract

The aim of this work is to show how Prolog technology can be used for efficient implementation of query answering in default logics. The idea is to translate a default theory along with a query into a Prolog program and a Prolog query such that the original query is derivable from the default theory iff the Prolog query is derivable from the Prolog program. In order to comply with the goal-oriented proof search of this approach, we focus on default theories supporting local proof procedures, as exemplified by so-called semi-monotonic default theories. Although this does not capture general default theories under Reiter’s interpretation, it does so under alternative ones'.

For providing theoretical underpinnings, we found the resulting compilation techniques upon a top-down proof procedure based on model elimination. We show how the notion of a model elimination proof can be refined to capture default proofs and how standard techniques for implementing and improving model elimination theorem provers (regularity, lemmas) can be adapted and extended to default reasoning. This integrated approach allows us to push the concepts needed for handling defaults from the underlying calculus onto the resulting compilation techniques.

This method for default theorem proving is complemented by a model-based approach to incremental consistency checking. We show that the crucial task of consistency checking can benefit from keeping models in order to restrict the attention to ultimately necessary consistency checks. This is supported by the concept of default lemmas that allow for an additional avoidance of redundancy.

© 1998 Elsevier Science B.V. All rights reserved.

Keywords: Default reasoning; Automated reasoning; Default Logic; Model elimination; PTPP; Model-based consistency checking

* Corresponding author. Email: torsten@cs.uni-potsdam.de.
1 Email: Stefan.Bruening@tlc.de.
1. Introduction

In many AI applications default reasoning plays an important role since many subtasks involve reasoning from incomplete information. This is why there is a great need for systematic methods that allow us to integrate default reasoning capabilities. In fact, the two last decades have provided us with a profound understanding of the underlying problems and have resulted in well-understood formal approaches to default reasoning. Therefore, we are now ready to build advanced default reasoning systems. For this undertaking, we have chosen Reiter's default logic [71] as our point of departure.

Default logic augments classical logic by default rules that differ from standard inference rules in sanctioning inferences that rely upon given as well as absent information. Knowledge is represented in default logics by default theories \((D, W)\) consisting of a consistent set of formulas \(W\), also called facts, and a set of default rules \(D\). A default rule \(\alpha : \beta \gamma\) has two types of antecedents: a prerequisite \(\alpha\) which is established if \(\alpha\) is derivable and a justification \(\beta\) which is established if \(\beta\) is consistent. If both conditions hold, the consequent \(\gamma\) is concluded by default. A set of such conclusions (sanctioned by default rules and classical logic) is called an extension of an initial set of facts: given a set of formulas \(W\) and a set of default rules \(D\), any such extension \(E\) is a deductively closed set of formulas containing \(W\) such that, for any

\[
\begin{align*}
\alpha : \beta \\
\gamma \\
\end{align*}
\]

\(\in D,\)

if \(\alpha \in E\) and \(\neg \beta \notin E\) then \(\gamma \in E\). (A formal introduction to default logic is given in Section 2.)

In what follows, we are interested in implementing the basic approach to query answering in default logic that allows for determining whether a formula is in some extension of a given default theory. Unlike other approaches that address this problem by encapsulating the underlying theorem prover as a separate module, we are proposing a rather different approach that integrates default reasoning into existing automated theorem provers. In order to comply with the methodology underlying query-oriented classical theorem provers, it is more or less indispensable to center the overall approach around local proof procedures (i.e., proof procedures that allow for deciding whether a set of default rules forms a default proof by looking at the constituent rules only). This is because such procedures permit validating a default inference step during the goal-directed proof search in a locally determinable way (see Section 2 for details).

The methodology presented in this paper has its origins in an approach to default query answering proposed in [79]. This approach integrates the notion of a default proof into a calculus for classical logic, which renders it especially qualified for implementations by means of existing theorem provers. To be more precise, Schaub [79] furnishes a mating-based characterization of default proofs inside the framework provided by the connection method [15]. This results in a connection calculus for query answering in so-called semi-monotonic default logics. (The advantage of these default systems is that they allow for the

\(^2\) Membership in all extensions is actually computable by appeal to a procedure testing membership in some extension (see [89]).
aforementioned local proof procedures, as we detail in Section 2.) In fact, there are already numerous implementations of connection calculi. Most of them, like the high performance theorem prover SETHEO [47], are based on model elimination [53] which can be regarded as a member of the family of connection calculi.

We draw on this relationship in this paper and show how an implementation technique for model elimination, namely Prolog Technology Theorem Proving (PTTP) [87,88], can be used for default reasoning. Our overall contribution can thus be looked at from different perspectives: first, we provide implementation techniques for (semi-monotonic) default logics. Second, we extend an existing automated theorem prover by means for handling default information. In particular, we show how the notion of a model elimination proof can be refined to capture (semi-monotonic) default proofs and how standard techniques for implementing and improving model elimination theorem provers (regularity, lemmas) can be adapted and extended to default reasoning. And finally, one can view our contribution somehow as a logic programming system integrating disjunction, classical as well as default negation.

To give a more precise overview of our approach, we start by noting that default logic is among the consistency-based approaches to default reasoning. In these formalisms, a logical formalization of a consistency-driven procedure is added to a standard logic. As explained above, this is done in default logic by means of the justification of a default rule. In this way, default reasoning is mapped onto a deductive task and a consistency checking task. Of course, this carries over to the resulting proof procedures for query answering, too.

As anticipated above, we address the deductive task of default query answering by appeal to techniques borrowed from Prolog Technology Theorem Proving. The idea is to translate a default theory along with a query into a Prolog program and a Prolog query such that the original query belongs to an extension of the default theory iff the Prolog query is derivable from the Prolog program. For providing theoretical underpinnings, we found the resulting compilation techniques upon a top-down proof procedure based on model-elimination. This proof procedure has its roots in the mating-based characterization of default query answering given in [79], which also integrates the notion of a default proof into a calculus for classical logic. This integrated approach allows us to push the concepts needed for handling defaults from the underlying calculus, over the corresponding proof procedure, into the resulting compilation techniques.

As regards the task of consistency checking, the first interesting question is whether we can find a way of pruning "inconsistent subproofs" while reducing the computational efforts for consistency checking. For this, we observe that a formula is consistent (or satisfiable) iff it has a model. This leads us to the following incremental approach to consistency checking: we start with a model of the initial set of facts. Each time, we apply a default rule, we check whether the actual model satisfies the underlying default assumptions (which can be done in linear time in propositional logic). If this is the case, we continue proving. If not, we try to generate a new model of the initial set of facts satisfying the current as well as all default assumptions underlying the partial default proof at hand. If we succeed, we simply continue proving under the new model. Otherwise, we know that the considered default assumption cannot be assumed in a consistent way (since there is no model for the continuation of our current default proof with the default rule at
hand). In this way, we restrict the generation of new models to the ultimately necessary ones.

The second interesting question is then whether a simultaneous treatment of theorem proving and satisfiability checking allows for proof procedures benefiting from structure and information sharing. This is important since the two tasks encompass genuine and even orthogonal sources of (putative) exponential complexity, as reflected by the fact that query answering in default logics is $\Sigma^P_2$-complete [42]. For addressing this issue, the idea is to communicate information from the theorem prover to a model generator. This communication is accomplished by (default) lemma handling. That is, apart from the traditional usage of lemmas for improving inferential processes, we use them furthermore as a communication device between two such processes. We will see that this allows for a drastic reduction of the search space in case a new model has to be generated.

The paper is organized as follows. After inserting the current work into the literature, we provide in Section 2 an introduction to default logic along with its basic proof theory (for normal default theories, see below). Section 3 introduces then a model-elimination calculus that is proven to be sound and complete for query answering in (normal) default logic. This calculus furnishes the theoretical foundation of the compilation techniques introduced in Section 5. This section extends the work found in [75]. While our integral approach addresses full-fledged (semi-monotonic) default logics, which support local proof procedures, we restrict our exposition up to Section 6 to so-called normal default theories over a propositional language. This restriction is justified by the fact that except for consistency checking all techniques developed for normal default theories carry over to the general case without any modifications. Hence, we present in Section 4 our model-based approach to consistency checking with normal default theories. This section extends the work found in [20]. We ultimately lift the overall approach to general default theories in Section 6. We accomplish this by extending the model-based approach of Section 4 in order to encompass the variety of consistency checks found in existing default logics. This builds on the work in [76]. As a final outcome, we obtain a PTTP-based implementation platform for query answering in default logics that support local proof procedures.

We draw the reader's attention right from the start to the fact that this paper focuses on the fundamentals of our approach, such as proof theoretical issues and compilation techniques. A more detailed description of the resulting system along with more experimental analysis can be found in [59]; a companion paper on further implementation techniques is in preparation. For an impression, consult Appendix A containing an example along with the resulting "object-code".

Our system is freely available at [83]. Documentation and sets of test cases can be obtained from the same location. In order to make use of the system you will need a standard Prolog system, preferably Eclipse Prolog.

In what follows, we assume the reader to be familiar with the basic concepts of propositional and first-order logic [10,32] and we presume some acquaintance with automated theorem proving [15,53] and programming language Prolog [27]. Throughout the paper, we deal with the propositional case, even though our implementation treats variables over a finite universe (in the rudimentary sense that a formula or a rule, respectively, is regarded as the representative of all its ground instances; thus, skolemization is not considered). We give some details of this in Section 5.2.5.
1.1. Related work

The question of determining whether a formula is in some extension of a given default theory was first addressed in [71]. An alternative approach was recently proposed in [79]. In contrast to this, most of the work found in the literature focuses on the determination of entire extensions, like \([2,11,25,45,50,63,81,82]\), or related decision procedures, like \([62, 63]\). In the former approaches, queries are then answerable either by look-up operations or by restricting the construction of extension to those containing the query at hand. Junker and Konolige [45] and Lévy [50] propose truth maintenance based systems. Ben-Eliyahu and Dechter [11] reduce default reasoning to a constraint satisfaction problem. Among the approaches integrating default reasoning into inference systems for classical logic, we find in [82] a tableaux-based framework for the computation of extensions of normal default theories; it was extended to full-fledged classical default logic in [81] and to justified and constrained default logic in [72]. In this framework, clausal tableaux are used for capturing maximal sets of "consistent default rules". Unlike this, Amati et al. [2] use parallel tableaux for capturing at once default inferences and consistency checks. In both approaches, final extensions are then characterized by a resulting tableau. Another hybrid method is used in [17], where a cut-free sequent calculus is proposed. This system consists of three parts: a classical LK-calculus, a sequent calculus based on "anti-sequents" for consistency checking, and certain default inference rules. Along this line of research, Bonatti and Olivetti [16] propose a sequent calculus for skeptical reasoning in default logic. In contrast to these approaches, many others, like \([5,25,45,63,71]\), abstract from an underlying inference engine and presuppose an automated theorem prover module furnishing an oracle for classical inference relations. There is also a variety of approaches addressing certain fragments of default logic, like \([13,18,71,82]\), and notably Poole's Theorist framework \([64,65,67,69]\).

Apart from the context of default logic, we mention the work on automating autoepistemic logic, e.g., \([61,62]\), and circumscription, e.g., \([6,37,39,43,70]\). Among them, though conceptually different from our approach, Gelfond and Lifschitz [37] compile circumscription into logic programs, while Przymusinski [70] uses a form of linear resolution that is related to model elimination.

Along the broader theme of our work, that is, the utilization of standard automated theorem proving techniques for implementing default reasoning, there is clearly a broader range of background literature. Among them, we find the work of Hoppe [44] and Sattar [74] dealing with incremental (default) reasoning in different settings, such as the aforementioned Theorist framework in the latter case. Moreover, a lot of effort has already been devoted to specific topics, like consistency checking, lemma handling, etc., on which outcomes we rely without giving a detailed account of the literature. And last but not least we mention the large efforts taken in the logic programming community for implementing extended logic programming [36].

Among the aforementioned approaches to default reasoning, an exposed position is arguably held by those of the groups around Niemelä [63] and around Marek and Truszczyński [25], also due to the impressive performances exhibited by their default reasoning systems. These approaches are orthogonal to the one proposed in this paper for three reasons: (i) they aim primarily at computing entire extensions (comprising query answer-
ing in the aforementioned way), (ii) they deal with Reiter's original default logic only, and (iii) they encapsulate the underlying theorem prover. Moreover, Niemelä [63] puts strong emphasis on conflict resolution techniques, while Cholewiński et al. [25] discuss in depth the influence of stratification techniques (see below). In this paper, we shift the emphasis towards the utilization of classical automated theorem proving techniques, while adopting a query-oriented perspective on default reasoning that relies on local proof procedures. (As regards (iii), we acknowledge that our final implementation is also not fully homogeneous since we use an independent Davis–Putnam procedure [28] for model-finding.)

As mentioned in the introductory section, our decision to center the approach around local proof procedures is motivated by the desire of integrating it into an existing goal-oriented, top-down automated theorem prover. We thus aim at verifying the validity of each inference step when it is performed in order to improve upon proof search. A related aim is found in approaches splitting default theories into smaller parts in order to apply default reasoning in a local way. Among them, we find [26,33,90]. For example, Cholewinski [26] takes up the notion of stratification techniques known in logic programming [3], which provide a salient part of the DeReS system [25].

In all, our restriction to default theories supporting local proof procedures (exemplified by semi-monotonic default theories) should be seen as a compromise between (i) our ultimate goal to exploit the deductive power of advanced inference engines and (ii) the expressiveness of the default theories under consideration. The latter has to do with the fact that default logics denying semi-monotonicity, like Reiter's full-fledged default logic, necessitate the inspection of all default rules for answering no matter what query. This setting is incompatible with the idea behind local, goal-oriented proof procedures; it rather requires a more global approach relying arguably on bottom-up procedures. This also explains why many of the aforementioned approaches to Reiter's original default logic are primarily interested in the computation of entire extensions: their setting necessitates the inspection of the entire set of default rules anyway.

2. Default logic

This section introduces Reiter's classical default logic along with some important formal concepts needed for providing a proof theory adequate for our purposes.

As already sketched in the introductory section, default logic augments classical logic by default rules of the form $\text{Prereq}(\delta)$ is the set of prerequisites of all default rules in $D$; $\text{Conseq}(D)$ is defined analogously. A set of default rules $D$ and a set of consistent formulas $W$ form a default theory $(D, W)$ that may induce a single or multiple extensions in the following way [71].

3 Reiter [71] considers default rules having finite sets of justifications. Marek and Truszczynski [57] show that any such default rule can be transformed into a set of default rules having a single (or no) justification.

4 The restriction to consistent set of facts is not really necessary, but it simplifies matters.
Definition 2.1. Let \((D, W)\) be a default theory. For any set of formulas \(S\), let \(\Gamma(S)\) be the smallest set of formulas \(S'\) such that:

1. \(W \subseteq S'\);
2. \(\text{Th}(S') = S'\);
3. for any \(\alpha \vdash \beta \in D\), if \(\alpha \in S'\) and \(S \cup \{\beta\}\) is consistent then \(\gamma \in S'\).

A set of formulas \(E\) is a classical extension of \((D, W)\) iff \(\Gamma(E) = E\).

Observe that \(E\) must be a fixed point of \(\Gamma\). Any such set represents a possible set of beliefs about the world at hand.

As already put forward in [71], query answering in default logics is most feasible in the presence of the property of semi-monotonicity: if \(D' \subseteq D\) for two sets of default rules, then if \(E'\) is an extension of \((D', W)\), there is an extension \(E\) of \((D, W)\) such that \(E' \subseteq E\). Given this property, it is sufficient to consider a relevant subset of default rules while answering a query, since applying other default rules would only enlarge and thus preserve a partial extension at hand. In Reiter’s default logic, semi-monotonicity is enjoyed by normal default theories. Moreover, all major variants of default logic such as classical default logic [71], justified default logic [55], cumulative default logic [19], constrained default logic [30], and rational default logic [58] coincide on this particular fragment. This is why we have chosen normal default theories as an initial exemplar for our approach. We show in Section 6 how general default theories are treated in the aforementioned variants, provided they enjoy semi-monotonicity.

In the presence of semi-monotonicity, extensions are constructible in a truly iterative way by applying one applicable default rule after another by appeal to a rather local notion of consistency:

Theorem 2.1. Let \((D, W)\) be a normal default theory and let \(E\) be a set of formulas. Then, \(E\) is an extension of \((D, W)\) iff there is some maximal \(D' \subseteq D\) that has an enumeration \((\delta_i)_{i \in I}\) such that for \(i \in I\), we have:

\[
E = \text{Th}(W \cup \text{Conseq}(D')) ,
\]

(1)

\[
W \cup \text{Conseq}(\{\delta_0, \ldots, \delta_{i-1}\}) \vdash \text{Prereq}(\delta_i),
\]

(2)

\[
W \cup \text{Conseq}(\{\delta_0, \ldots, \delta_{i-1}\}) \nvdash \text{Conseq}(\delta_i).
\]

(3)

This type of characterization was first given in [82], except for condition (3) reflecting an incremental approach, as opposed to a rather global approach requiring

\[
W \cup \text{Conseq}(D') \nvdash \bot.
\]

Condition (2) spells out that \(D'\) has to be grounded in \(W\). In general, a set of default rules \(D\) is grounded in a set of facts \(W\) iff there exists an enumeration \((\delta_i)_{i \in I}\) of \(D\) that satisfies condition (2). Condition (3) expresses the notion of incremental consistency. Here, the “consistent” application of a default rule is checked at each step, whereas this must be done with respect to the final extension in a non-semi-monotonic default logic.

---

5 This is meant modulo the augmented language used by this system (see Section 6 for more details).
These notions lead us to the following notion of a default proof from normal default theories, on which we build our initial formal characterization of query answering:

**Definition 2.2.** Let \((D, W)\) be a normal default theory and \(\varphi\) a formula. A normal default proof for \(\varphi\) from \((D, W)\) is a finite sequence of default rules \((\delta_i)_{i \in I}\) with \(\delta_i \in D\) for all \(i \in I\) such that \(W \cup \{\text{Conseq}(\delta_i) | i \in I\} \vdash \varphi\) and conditions (2) and (3) are satisfied for all \(i \in I\).

The following immediate consequence of Theorem 2.1 assures that a query is in some extension of the (normal) default theory at hand iff it has a (normal) default proof:

**Theorem 2.2.** Let \((D, W)\) be a normal default theory and \(\varphi\) a formula. Then, \(\varphi \in E\) for some extension \(E\) of \((D, W)\) iff there is a normal default proof for \(\varphi\) from \((D, W)\).

That is, for verifying whether \(\varphi\) is in some extension of a default theory \((D, W)\), it is enough to determine a grounded and consistent set of default rules \(D_0 \subseteq D\) that allows for proving \(\varphi\) from the facts in \(W\) and the consequents of all default rules in \(D_0\).

Now, given the concept of a default proof, let us elucidate the computational advantage of local proof procedures provided by semi-monotonicity: for deciding whether a set of default rules forms a default proof, it is sufficient to investigate the constituent rules only. A local proof procedure must thus never consider a rule in \((D \setminus D_0)\) for deciding whether \(D_0\) is a default proof for some query \(\varphi\). Note that in the absence of a property like semi-monotonicity a proof procedure must necessarily consider all default rules in the given default theory. This is, for instance, the case for general default theories under Reiter’s interpretation. Finally, we emphasize that we are not interested in semi-monotonicity as such, it is rather the resulting localness of proof procedures that we draw upon, since this is an essential feature of the model elimination based theorem provers that we are aiming at.

As an example, consider the following set of statements about a child predisposed to an allergy against milk products: “children normally eat ice-cream”, “ice-cream usually contains milk”, “ice-cream usually contains sugar”, and “milk is an allergen in case of a predisposition”. The corresponding default theory along with facts \(\text{child} \land \text{predispo}\) (expressing that the considered child has the aforementioned predisposition) is the following one:

\[
\left( \{ \text{child} : \text{icecream}, \text{icecream} : \text{milk}, \text{icecream} : \text{sugar} \} \right). \\
\{ \text{child}, \text{predispo}, \text{milk} \land \text{predispo} \rightarrow \text{allergen} \}
\]

---

6 Since we deal with normal default theories up to Section 6, we omit the prefix normal up to this point.

7 Unless it has a particular syntactical structure allowing for “localizing” the proof procedure, for instance by stratification [3] or similar techniques.

8 This is actually an extremely rich and non-trivial domain for studying default reasoning.
For instance, we can explain the presence of an allergen in the above situation by proving allergen from child ∧ predispo by means of default proof:

\[
\begin{align*}
(\text{child} : \text{icecream} & , \text{icecream} : \text{milk}) \\
\text{icecream} & , \text{milk}
\end{align*}
\]

Importantly, this proof can be found by a top-down backward-chaining procedure that starts from the query by ignoring irrelevant default rule

\[
\text{icecream} : \text{sugar}
\]

This illustrates the great advantage of local proof procedures.

For the most part of the paper, we follow [79] in dealing with default theories in atomic format in the following sense: for a default theory \((D, W)\) in language \(\mathcal{L}_\Sigma\) over some alphabet \(\Sigma\), let \(\mathcal{L}'_\Sigma\) be the language over the alphabet \(\Sigma'\), obtained by adding three new propositions, named \(\alpha_\delta, \beta_\delta, \gamma_\delta\) for each \(\delta \in D\). Then, \((D, W)\) is mapped into default theory \((D', W')\) in \(\mathcal{L}'_\Sigma\), where

\[
D' = \left\{ \alpha_\delta : \beta_\delta \mid \gamma_\delta \mid \delta \in D \right\},
\]

\[
W' = W \cup \{\text{Prereq}(\delta) \rightarrow \alpha_\delta, \beta_\delta \rightarrow \text{Justif}(\delta), \gamma_\delta \rightarrow \text{Conseq}(\delta) \mid \delta \in D\}.
\]

The resulting default theory \((D', W')\) is called the atomic format of the original default theory, \((D, W)\). As shown in [73], this transformation does not affect the computation of queries to the original default theory. That is in terms of default proofs, given a query \(\varphi\) in \(\mathcal{L}_\Sigma\), then \(\varphi\) has a default proof from \((D, W)\) iff \(\varphi\) has a default proof from \((D', W')\). We can therefore restrict our attention to atomic default rules without losing generality. The advantages of atomic default rules over arbitrary ones are, first, that their constituents are not spread over several clauses while transforming them into clausal format (see Section 3) and, second, that these constituents are uniquely referable to. The motivations for this format are somehow similar to the ones for definitional clausal form in automated theorem proving [31]. For default reasoning, the naming of defaults was first done by Poole in [67].

For clarity, we refrain from turning default rules into their atomic counterparts whenever they are composed of atomic components. This is for instance the case with Default theory (4). For an example of the transformation the reader is referred to Section 4; Section 5.2 describes its benefits on the implementation level.

3. Query answering in default logics

The aim of this section is to provide formal underpinnings for the compilation techniques to be introduced in Section 5. As mentioned in the introductory section, our approach is rooted in the mating-based characterization of default proofs developed in [79]. Such matings are used in connection calculi as a structure-oriented means for characterizing the unsatisfiability of formulas. For brevity, we refer the reader for details on this approach to [79] and confine ourselves in what follows to an introduction to its underlying ideas, needed for a better understanding of the model elimination calculus, presented in the major part of this section.
First of all, we introduce the following conventions: we let $L^c$ denote the literal complementary to $L$. We mainly deal with formulas in conjunctive normal form (CNF), which are given by a conjunction of disjunctions of literals. To ease notation, we denote such formulas in clausal form, that is, a formula in CNF is given as a set of clauses, where a clause is a set of literals $\{L_1, \ldots, L_n\}$ representing a disjunction $L_1 \lor \cdots \lor L_n$. A clause is called negative if it contains only negative literals.

The mating-based characterization of default proofs relies on the idea that an atomic default rule $\frac{\alpha}{\gamma}$ can be decomposed into a classical implication $\alpha \rightarrow \gamma$ along with two proof-theoretic conditions on the usage of the resulting clause $\{\neg\alpha, \gamma\}$; these conditions are referred to as admissibility and compatibility. Intuitively, both of them rely on a sequence of clauses, stemming from default rules only, which is induced by the underlying mating (see [79]). Such a sequence amounts to an enumeration of default rules $\delta_i$, as given in Theorem 2.1 and Definition 2.2. In fact, while admissibility provides the proof-theoretic counterpart of condition (2), that is groundedness, compatibility enforces the notion of consistency described in condition (3).

Now, in order to find out whether a formula $\varphi$ is in some extension of a default theory $(D, W)$, we proceed as follows: first, we transform the default rules in $D$ into a set of indexed implications $W_D$. In our example, this encoding yields the set

$$W_D = \{\text{child}_{\delta_1} \rightarrow \text{icecream}_{\delta_1}, \text{icecream}_{\delta_2} \rightarrow \text{milk}_{\delta_2}, \text{icecream}_{\delta_3} \rightarrow \text{sugar}_{\delta_3}\}.$$  \hspace{1cm} (6)

The indexes denote the respective default rules in (4) from left to right. \(^9\)

Second, we transform both $W$ and $W_D$ into their clausal forms, $C_W$ and $C_D$. The clauses in $C_D$ are called $\delta$-clauses; they are of the form \(^{10}\) $\{\neg\alpha, \gamma\}$; all other clauses are referred to as $\omega$-clauses. In our example, we obtain the following clause set for $C_W \cup C_D$:

$$\{\{\text{predispo}, \{\text{child}\}, \{\neg\text{predispo}, \neg\text{milk, allergen}\}\} \cup$$

$$\{\{\neg\text{child}_{\delta_1}, \text{icecream}_{\delta_1}\}, \{\neg\text{icecream}_{\delta_2}, \text{milk}_{\delta_2}\}, \{\neg\text{icecream}_{\delta_3}, \text{sugar}_{\delta_3}\}\}. \hspace{1cm} (7)$$

Finally, a query $\varphi$ is derivable from $(D, W)$ iff the set of clauses $C_W \cup C_D \cup \{\{\neg\varphi\}\}$ is unsatisfiable and agrees with the (mating-based) concepts of admissibility and compatibility.

3.1. A default model elimination calculus

For providing the formal underpinnings for our compilation techniques, we develop in the sequel a proof procedure which finds a (default) refutation for $\neg \varphi$ from clause set $C_W \cup C_D$ iff there exists a default proof for $\varphi$ from $(D, W)$.

Unlike [79] though, we address this problem by means of a variant of model elimination (ME) [53]. This is because the mating-based approach provides a purely declarative characterization of default proofs, which is much too abstract to provide an adequate basis for the compilation techniques to be introduced in Section 5. In particular, it does not provide means for representing derivations and it cannot reflect the relation between goals

\(^9\) Of course, these indexes do not influence two literals' complementarity.

\(^{10}\) Recall that the atomic format allows us to deal with binary $\delta$-clauses only.
and their subgoals. In contrast to this, an adequate account of these notions is furnished by a ME-based approach.

The basic inference steps of ME-based calculi are called extension and reduction step. Intuitively, an extension step amounts to Prolog's use of input resolution: a subgoal $L$ is resolved with an input clause $\{L', K_1, \ldots, K_n\}$ resulting in the new subgoals $K_1, \ldots, K_n$. For illustration, let us consider the clauses in (7) along with query allergen: we can resolve initial goal $\neg$allergen with clause $\{\neg$predispo, $\neg$milk, allergen\}. The resulting subgoal $\neg$milk can then be resolved with clause $\{milk, \neg$icecream\} and so on. The reduction step renders the inference system complete for (full) propositional clause logic: a subgoal is solved if it is complementary to one of its ancestor subgoals. In our example, we thus obtain the set $\{\neg$allergen, $\neg$milk\} of ancestor goals after the two aforementioned extension steps; this allows for applying subsequently reduction steps to putative subgoals allergen and milk.

For incorporating default reasoning into such a calculus, both inference steps have to be adapted appropriately: first, one has to take care of groundedness (cf. condition (2)). To this end, we have to guarantee that whenever a $\delta$-clause $\{\neg\alpha_3, \gamma_3\}$ is used as input clause, (i) only $\gamma_3$ is resolved upon, and (ii) after such an “extension step” the ancestor goals of the resulting subgoal $\neg\alpha_3$ must not be used for subsequent reduction steps. Moreover, (iii) each such “extension step” must guarantee the consistency with the previous proof segment (cf. condition (3)).

Among the diverse mouldings of ME-based calculi, we consider a variant that relies on so-called ME-tableaux as basic proof objects (cf. [47,48]). Note that the original definition of ME given in [54] is based on so-called ME-chains which, roughly speaking, correspond to open tableau branches (see below). Our choice to use ME-tableaux rather than ME-chains as basic proof objects is motivated by the requirement to consider from time to time whole derivations during a deduction for checking consistency. Such a global view on derivations is furnished by entire ME-tableaux.

In what follows, we restrict the definitions to the propositional case (corresponding definitions for full clause logic can be found in [47,48]). The following list gives definitions of the concepts needed for the resulting ME-calculus:

**Literal tree.** A literal tree is a pair $(t, \lambda)$ consisting of an ordered tree $t$ and a labeling function $\lambda$ assigning literals to the non-root nodes in $t$.

We say node $o$ in $t$ is labeled with literal $L$, if $\lambda(o) = L$.

**Successor sequence.** The successor sequence of a node $o$ in an ordered tree $t$ is the sequence of nodes with immediate predecessor $o$, in the order given by $t$.

**Clausal tableau.** A (clausal) tableau $T$ of a set of clauses $M$ is a literal tree $(t, \lambda)$ in which $\{\lambda(o_1), \ldots, \lambda(o_n)\} \in M$ for every maximal successor sequence $(o_1, \ldots, o_n)$.

**Tableau clause and literals.** Such a clause $\{\lambda(o_1), \ldots, \lambda(o_n)\} \in M$ is called a tableau clause and the elements of a tableau clause are called tableau literals.

---

This is why in implementations these ancestor goals are normally accumulated by keeping the one from each previous input clause.
**Depth of a tableau clause.** The depth of a tableau clause \( \{ \lambda(o_1), \ldots, \lambda(o_n) \} \) in a clausal tableau \((t, \lambda)\) is defined as the depth of \(o_1\) in \(t\) (where the root node of \(t\) has depth 0, and the depth of a non-root node in \(t\) is the depth of its direct predecessor plus one).

**Model elimination tableau.** A tableau is called model elimination tableau (ME-tableau) if each inner node \(o\) labeled with a literal \(L\) has a leaf node \(o'\) among its immediate successor nodes which is labeled with literal \(L^c\).

Given a tableau \(T\) containing some node \(o\), we often denote the literal attached to \(o\) in \(T\) by \(\lambda^T(o)\) (or simply \(\lambda(o)\), if clear from the context).

**Branch.** A branch of a tableau \(T\) is a sequence \(\langle o_1, \ldots, o_n \rangle\) of nodes in \(T\) such that \(o_1\) is the root of \(T\), \(o_i\) is the immediate predecessor of \(o_{i+1}\) for \(1 \leq i < n\), and \(o_n\) is a leaf of \(T\).

We sometimes denote a branch \(\langle o_1, \ldots, o_n \rangle\) by a sequence containing the labels of its nodes, that is, we write \(\langle \lambda(o_2), \ldots, \lambda(o_n) \rangle\) (note that the root node of a tableau is never labeled with a literal).

**Complementary branch.** A branch is complementary if the labels of its nodes \(o_1, \ldots, o_n\) contain some literal \(L\) and its complement \(L^c\).

**Open and closed branch.** In order to distinguish the simple presence of a complementary branch and the detection of this fact, we allow to label branches as closed.

Each branch which is labeled as closed must be complementary. A branch which is not marked as closed is called an open branch.

**Open and closed tableau.** A tableau is closed if each of its branches is closed, otherwise it is open.

**Ancestor node and ancestor literal.** Given a branch \(\langle o_1, \ldots, o_n \rangle\) in an ME-tableau \(T\), we call \(o_i\) an ancestor (node) of \(o_j\) and \(\lambda^T(o_i)\) an ancestor (literal) of \(\lambda^T(o_j)\) iff \(i < j\).

We denote the immediate ancestor \(o_i\) of node \(o_{i+1}\) by \(\text{prev}(o_i)\).

**Successor node and successor literal.** Correspondingly, we call \(o_i\) a successor (node) of \(o_j\) and \(\lambda^T(o_i)\) a successor (literal) of \(\lambda^T(o_j)\) iff \(i > j\).

**Open goal.** Let \(b\) be an open branch of an ME-tableau \(T\). If literal \(L\) is the label of the leaf node of \(b\), then \(L\) is called an open goal of \(T\).

Consider the ME-tableau \(T\) given in Fig. 1; it contains three tableau clauses (namely, \{\neg\text{allergen}, \neg\text{milk}, \neg\text{predispo}\} and \{\text{milk}_{o_2}, \neg\text{icecream}_{o_2}\}) and four branches (namely, \{\neg\text{allergen}, \neg\text{allergen}, \neg\text{milk}, \neg\text{predispo}, \neg\text{milk}, \text{milk}_{o_2}\} and \{\neg\text{allergen}, \neg\text{milk}, \neg\text{icecream}_{o_2}\}). Two of these branches (the first and third one) are complementary since they contain complementary literals and can therefore be labeled as closed. Closed branches are marked by underlining their leaf nodes. Since \(T\) contains open branches it is not closed.

The following theorem, proven among others in [49], forms the basis of calculi using ME-tableaux as proof objects:
Fig. 1. An ME-tableau of clause set (7) with query allergen. Each node of the tableau is represented by its label except for the root node which is depicted by $\bullet$.

**Theorem 3.1.** Let $M$ be a clause set. Then, $M$ is unsatisfiable iff there exists a closed model elimination tableau of $M$.

Based on the above definitions, we are now apt to introduce the inference steps for the model-elimination-based calculus needed for furnishing a top-down technique generating ME-tableaux: the first inference step to be introduced is the so-called *initialization step*; it allows to build initial tableaux consisting of a root node and one tableau clause. By using so-called *$\omega$-extension steps* and *$\delta$-extension steps*, respectively, tableaux can then be extended by $\omega$- and $\delta$-clauses from a given clause set. Finally, *reduction steps* are used to check the complementarity of an open goal to some of its ancestors.

Throughout the following definitions, let $T = (t, \lambda)$ be an arbitrary yet fixed tableau of some set of input clauses $M = C_W \cup C_D$, comprising a set $C_W$ of $\omega$-clauses and a set $C_D$ of $\delta$-clauses.

**Definition 3.1** (*Initialization step*). Tableau $T'$ is obtained by an initialization step in the following way.

- Let $o$ be the root of a one-node tree.
- Select in $M$ a negative $\omega$-clause $\{L_1, \ldots, L_n\} \in M$.
- Then, attach $n$ new successor nodes to $o$, and label them in turn with $L_1, \ldots, L_n$.

The new branch with leaf node $o_i$ is marked as closed.

For extending tableaux, we define two different variants of extension steps. The first variant is restricted to use input clauses from $C_W$ (i.e., $\omega$-clauses) for tableau expansion; this is identical to the extension step in classical ME-calculi.

**Definition 3.2** (*$\omega$-extension step*). Tableau $T'$ is obtained from $T$ by an $\omega$-extension step in the following way.

- Select in $t$ a leaf node $o$ of an open branch labeled with literal $L$.
- Let $\{L_1, \ldots, L_n\}$ be a $\omega$-clause in $M$ such that $L^c = L_i$ for some $i \in \{1, \ldots, n\}$.
- Then, attach $n$ new successor nodes $o_1, \ldots, o_n$ to $o$, and label them in turn with $L_1, \ldots, L_n$, respectively.
- The new branch with leaf node $o_i$ is marked as closed.
For extending tableaux by $\delta$-clauses from $C_D$, we introduce $\delta$-extension steps. To begin with, their definition must reflect the fact that defaults are inference rules rather than formulas: a clause $\{-\alpha_\delta, \gamma_\delta\}$ from $C_D$ can only be applied to a branch with open goal $\neg \gamma_\delta$. Taking into account that ME is a top-down backward-chaining calculus, this amounts to the application of the underlying default rule. Note that taking $\{-\alpha_\delta, \gamma_\delta\}$ as an ordinary $\omega$-clause allows for applying a $\omega$-extension step to an open goal $\alpha_\delta$; this corresponds to reasoning by contraposition, which denies the inference rule character of default rules. Such inferences are disallowed by $\delta$-extension steps:

**Definition 3.3** ($\delta$-extension step). Tableau $T'$ is obtained from $T$ by a $\delta$-extension step in the following way.

- Select in each leaf node $o$ of an open branch labeled with literal $L$.
- Let $\{-\alpha_\delta, \gamma_\delta\}$ be a $\delta$-clause in $M$ such that $L^c = \gamma_\delta$.
- Then, attach the two new successor nodes $o_1$ and $o_2$ to $o$, and label them in turn with $\neg \alpha_\delta$ and $\gamma_\delta$, respectively.
- The new branch with leaf node $o_2$ is marked as closed.

In what follows, we need the following vocabulary: considering Definition 3.2 (Definition 3.3, respectively), we call $o_1$ ($o_2$) an $\alpha$-extension node ($\delta$-extension node), and each element of $\{o_1, \ldots, o_n\} \setminus \{o_i\} \setminus \{o_1\}$, $\omega$-extension resulting node ($\delta$-extension resulting node), or simply a non-extension node. A literal attached to an $\omega$-extension node ($\delta$-extension node) is called an $\omega$-extension literal ($\delta$-extension literal), otherwise a non-extension literal. We sometimes omit the prefixes $\omega$- and $\delta$- whenever it is clear from the context.

As argued above, the use of $\delta$-clauses for tableaux extension must reflect the properties of default rules. Apart from being an inference rule, default rules must be applied in a consistency-preserving way. Transposed to $\delta$-clauses, we must guarantee that a $\delta$-extension step with $\delta$-clause $\{-\alpha_\delta, \gamma_\delta\}$ does not violate consistency criterion (3) in Theorem 2.1. To this end, it is sufficient to check whether $\gamma_\delta$ is consistent with all other $\delta$-extension literals in the current derivation.\(^{12}\)

**Definition 3.4** (Compatible $\delta$-extension step). Let $T$ be a tableau and let $\{\gamma_\delta_1, \ldots, \gamma_\delta_j\}$ be the set of $\delta$-extension literals occurring in $T$.

A $\delta$-extension step with $\delta$-clause $\{-\alpha_\delta, \gamma_\delta\}$ applied to $T$ is called compatible if $C_W \cup \{\gamma_\delta_1, \ldots, \gamma_\delta_j\} \cup \{\gamma_\delta\}$ is consistent.

Observe that although this definition is in accord with consistency condition (2) regarding their common incremental flavor, it involves a different set of underlying default rules. This is because the latter condition is conceived in a bottom-up fashion, while Definition 3.4 relies on a top-down approach. For a default rule $\delta_k$ in an entire default proof $\langle \delta_i \rangle_{i \in I}$, the former criterion involves default rules in $\langle \delta_i \rangle_{i < k}$, while the latter considers default rules in $\langle \delta_i \rangle_{i > k}$.

\(^{12}\) Note that $\delta$-extension literals correspond (as long as normal default rules are considered) to the justifications of the respective default rules.
Even though all these conceptions involve multiple default rules or \( \delta \)-clauses, respectively, it is always sufficient to restrict our attention to those used in the actual derivation only. This is due to the notion of localness backed up by semi-monotonicity. Nonetheless compatibility has a special status because it necessitates a more global treatment as opposed to \( \omega \)- and \( \delta \)-extension steps. We come back to this issue at the end of this section.

Actually, Definition 3.3 furnishes only one-half of the machinery needed for ensuring the inference rule character of default rules. Intuitively, this is because we must also eliminate reasoning by cases for \( \delta \)-clauses. This is done by restricting the well-known reduction step in classical ME-calculi. The sole difference is that subgoals of a \( \delta \)-extension-resulting literal \( \neg \alpha_\delta \) must not be solved by reduction steps using ancestors of \( \neg \alpha_\delta \).\(^{13}\) There must thus exist an independent default proof of \( \neg \alpha_\delta \), which ignores all ancestors of \( \neg \alpha_\delta \), or in other words, there must exist a closed ME-tableau with top-clause \( \{ \neg \alpha_\delta \} \).

**Definition 3.5 (Reduction step).** Tableau \( T' \) is obtained from \( T \) in the following way.

- Select in \( T \) the leaf node \( o_k \) of an open branch \( b = (o_1, \ldots, o_k) \) where \( o_k \) is labeled with literal \( L \).
- If there is an ancestor node \( o_i \) on \( b \) labeled with literal \( L^c \) and all nodes \( o_{i+1}, \ldots, o_k \) are \( \omega \)-extension-resulting nodes, then mark \( b \) as closed.

Consider the three tableaux depicted in Fig. 2. The leftmost tableau \( T_1 \) is generated by an initialization step with top-clause \( \{ \neg \text{allergen} \} \). The second tableau \( T_2 \) is generated from \( T_1 \) by applying an \( \omega \)-extension step with \( \omega \)-clause \( \{ \text{allergen}, \neg \text{milk}, \neg \text{predispo} \} \) to the sole open goal \( \neg \text{allergen} \). The rightmost tableau emerges from \( T_2 \) by the application of a \( \delta \)-extension step with \( \delta \)-clause \( \{ \text{icecream}_{\delta_2}, \text{milk}_{\delta_1} \} \) to open goal \( \neg \text{milk} \).

The above inference steps provide us with a sound and complete calculus, as given in the next definition:

**Definition 3.6 (Default model elimination).** A sequence \( \langle T_1, \ldots, T_n \rangle \) of ME-tableaux is called a **DME-derivation** for a clause set \( M \) (called the set of input clauses) if:

- \( T_1 \) is obtained by an initialization step and
- for \( 1 < i \leq n \), \( T_i \) is obtained from \( T_{i-1} \) by applying to \( T_{i-1} \) either
  - a reduction step,

\(^{13}\) This restriction reflects the property of admissibility, introduced in \([79]\).
Fig. 3. Generating a closed tableau from clause set (7) and top-clause \{-allergen\}.

- a \(\omega\)-extension step, or
- a compatible \(\delta\)-extension step.

A DME-derivation is called a **DME-refutation** if it generates a closed tableau.

Observe that each clause used in an extension step is by definition an input clause.

For convenience, we sometimes identify the elements of a derivation (namely the tableaux) with their generating inference steps: we thus write \(\langle d_1, \ldots, d_n \rangle\) instead of \(\langle T_1, \ldots, T_n \rangle\), where each \(d_i\) denotes the instance of the respective inference rule used for obtaining \(T_i\).

Let \(L\) be an open goal attached to a node \(o\) in an ME-tableau \(T\). A **DME-subderivation** \(D\) for \(o\) (or \(L\)) is a sequence of derivation steps where the first element of \(D\) selects \(o\) and each further element selects a descendant of \(o\). \(D\) is called a **DME-subrefutation** if after applying \(D\) to \(T\), each branch containing \(o\) is closed. We sometimes omit the prefix DME whenever it is clear from the context.

We continue the example developed in Fig. 2. A DME-refutation for query allergen from clause set (7) can be constructed as follows: first, a \(\omega\)-extension step with \(\omega\)-clause \{-predispo\} is applied to open goal \{-predispo\} of the rightmost tableau in Fig. 2. Second, a \(\delta\)-extension step with \(\delta\)-clause \{-child\(_{d_2}\), icecream\(_{d_2}\}\) is applied to open goal \{-icecream\(_{d_2}\}\.

As a result, we get a tableau (the leftmost tableau in Fig. 3) containing a single open branch. This open branch is closed by the application of a \(\omega\)-extension step with unit clause \{child\}. Hence, the resulting tableau (the one in the right in Fig. 3) is closed and so we have found a DME-refutation of allergen from clause set (7).

The following theorem tells us that a mechanism generating DME-derivations in an exhaustive manner constitutes a sound and complete proof mechanism for query answering in default logic (here, restricted to normal default theories).

**Theorem 3.2.** Let \((D, W)\) be a normal default theory and \(\varphi\) be an atomic formula. Let \(M\) be the clausal representation of the atomic format of \((D, W)\).

Then, there is a default proof for \(\varphi\) from \((D, W)\) iff there is a DME-refutation for \(M\) with top-clause \{\(\varphi^c\}\).
The restriction to atomic queries is no limitation of the approach, since an arbitrary query formula $\phi$ can always be replaced by an atomic one, say $\varphi$, and an additional formula $\phi \rightarrow \varphi$ in $W$. (This technique is also used by PTTP [86].)

It is worth emphasizing that for pure propositional clause sets without $\delta$-clauses, Default Model Elimination behaves exactly like ordinary ME (as defined for, example, in [47]). This is because:

- the definitions of initialization steps and $\omega$-extension steps are exactly the same as for classical propositional ME-calculi,
- $\delta$-extension steps do not enter the derivation, and
- if no $\delta$-clauses are used during a derivation, DME-reduction steps correspond exactly to classical ME-reduction steps.

The material presented in this section constitutes the fundamental basis for our approach to compiling query answering in default logic. The resulting DME-calculus provides us with a homogeneous and systematic characterization of default proofs that leaves room for diverse design decisions as regards the ultimate implementation. In fact, a salient feature of our approach is that it relies on a local proof procedure employing an incremental consistency check. As a result, all inference steps of our DME-calculus are executable in a more or less local fashion. This comprises the reduction step, searching among ancestor goals, as well as the verification of compatibility for $\delta$-extension steps which involves inspecting all $\delta$-extension literals occurring in the tableau at hand. But even though the latter allows for ignoring all default clauses outside the current derivation, it has nonetheless a special status due to the involved consistency check. Of course, such a consistency check can be mapped, roughly speaking, onto "unsuccessful" ME-derivations that yield (ordinary) ME-tableaux comprising at least one open branch.  

This would amount to a generalization of Prolog's "negation-as-failure" mechanism to full clause logic. We argue however that such an approach is infeasible due to the combinatoric explosion of (repeated) saturations; this gets even worse in the presence of disjunctions. Another objection is that we aim at compiling DME-derivations which is impracticable for consistency checking since we deal with dynamically changing sets of formulas (cf. Definition 3.4). Moreover, we argue in Section 4 that it is not even necessary to continuously perform exhaustive consistency checks, once we can actually represent and then eventually reuse the result of a successful consistency check. This issue is addressed in Section 4 before we introduce the actual compilation techniques in Section 5.

3.2. Extensions and refinements

3.2.1. Lemma handling

Lemma handling is an important means for eliminating redundancy in automated theorem proving (cf. [4]). This task is however more difficult in our context, since proofs may depend on default rules and their induced consistency requirements.

In fact, in classical ME-based calculi, a lemma $l$ is simply a set of literals that allows for closing each branch containing all elements of $l$ as labels of its nodes. It is well known

---

14 A similar approach was pursued in [79] with a facile decision procedure.
that, given two clause sets $M$ and $M'$ with $M' \subseteq M$, and a lemma $l$ (generated during a ME-derivation for $M'$), then $l$ can be applied safely during derivations for $M$. That is, each branch $b$ generated during a derivation for $M$ containing $l$ as a subset of the labels of $b$ can be marked as closed.

Unlike this classical approach, it is impossible to simply use such lemmas, like $l$, during a derivation in default theorem proving. This is because the subrefutation from which $l$ was generated might depend on a set of $\delta$-clauses $C_{DS}$. The use of $l$ during a derivation employing defaults not consistent with $C_{DS}$ would lead to incorrect results. In the context of default theorem proving, we therefore have to extend the concept of lemmas:

**Definition 3.7 (DME-lemma).** Let $C_W$ be a set of $\omega$-clauses, $C_D$ be a set of $\delta$-clauses, and let $T$ be a tableau generated from some DME-derivation for $C_W \cup C_D$. Let $o$ be some node in $T$ such that each branch in $T$ containing $o$ is closed, i.e., there exists a subrefutation $D$ for $o$. Further, let $o_1, \ldots, o_n$ be the ancestor nodes of $o$ used for reduction steps in $D$ and let $M \subseteq C_W \cup C_D$ be the set of input clauses used in $D$.

Then, clause $\{\lambda(o), \lambda(o_1), \ldots, \lambda(o_n)\}$ is called a **DME-lemma** with respect to $M$ and the set of default rules $\{D \mid \{\neg \alpha, \gamma \} \in (C_D \cap M)\}$.

Formal underpinnings for this notion can be given by appeal to so-called lemma default rules [78]. Restricted to normal default theories, a lemma default rule $\delta_l$ for a formula $l$ from $(D, W)$, is constructed from a default proof $\langle \delta_i \rangle_{i \in I}$ for $l$ in the following way:

$$\delta_l = \frac{\bigwedge_{\delta_i \in I} \text{Conseq}(\delta)}{l}.$$  

Since this results in a non-normal default rule, the precise meaning has to be fixed with respect to a full-fledged default logic. For full-fledged Reiter's default logic, it is shown in [78] that $E$ is an extension of $(D, W)$ if and only if $E$ is an extension of $(D \cup \{\delta_l\}, W)$.

Our approach is justified by the following result.

**Theorem 3.3.** Let $C_W$ be a set of $\omega$-clauses and $C_D$ be a set of $\delta$-clauses such that $W \cup \text{Conseq}(D)$ is consistent. Let $l$ be a DME-lemma with respect to some subset of $C_W \cup C_D$ and some set of default rules $D' \subseteq D$. Further, let $T$ be a tableau generated by a DME-derivation from $C_W \cup C_D$, and let $b = \langle o_1, \ldots, o_v, \ldots, o_m \rangle$ be a branch of $T$.

If $o_{v+1}, \ldots, o_m$ are $\omega$-extension-resulting nodes and $l \subseteq \{\lambda(o_v), \ldots, \lambda(o_m)\}$, then $b$ can be marked as closed (without losing soundness).

As a corollary to this result, we obtain that soundness is preserved when extending DME-derivations with an appropriate **Lemma step**, which can be given shape as follows. For this, let $T = (t, \lambda)$ and $l = \{L_1, \ldots, L_n\}$ be in accord with Theorem 3.3:

**Definition 3.8 (Lemma step).** Tableau $T'$ is obtained from $T$ in the following way.

- Select in $t$ an open branch $b = \langle o_1, \ldots, o_v, \ldots, o_m \rangle$ labeled with $K_1, \ldots, K_v, \ldots, K_m$.
- If $o_{v+1}, \ldots, o_m$ are $\omega$-extension-resulting nodes on $b$ and $\{L_1, \ldots, L_n\} \subseteq \{K_v, \ldots, K_m\}$, then mark $b$ as closed.
The concept of DME-lemmas is further discussed in Section 4, including an illustrative example. In particular, we rely on DME-lemmas in Section 4 for communicating information from DME-derivations to an attached consistency checker.

3.2.2. Loop checking by blockwise regularity

Regularity provides a highly efficient means for restricting the search space in ME-based theorem proving. It forbids to generate tableaux containing two nodes, \( o_1 \) and \( o_2 \) say, on the same branch such that \( o_1 \) and \( o_2 \) are labeled with the same literal. Using this refinement, the number of possible tableaux to be build during a deduction decreases considerably in many cases (e.g., see \([47,48]\)).

Unfortunately, this important refinement for classical ME cannot be applied to Default Model Elimination without losing completeness. However, as we will show in the sequel, it is possible to adapt regularity to the needs of default reasoning. This leads us to what we call blockwise regularity, which requires (i) that no subgoal \( L_1 \) is equal to one of its ancestors \( L_2 \) unless there is a \( \delta \)-extension-resulting literal between \( L_1 \) and \( L_2 \), and (ii) that no two different \( \delta \)-extension resulting nodes on the same branch have ancestor nodes labeled with equal literals (in other words, there is no need to construct a branch using two \( \delta \)-clauses corresponding to defaults with the same consequent).

In fact, on the pure classical parts of a default proof, blockwise regularity behaves exactly like ordinary regularity, whereas in the parts of a derivation involving \( \delta \)-clauses, it guarantees that no \( \delta \)-clauses with the same consequent are used for constructing a branch. Besides pruning large parts of the search space, blockwise regularity also guarantees completeness since it is necessarily violated by infinite branches.

The following definition introduces the concept of blocks which is required for a formal definition of blockwise regularity. Roughly spoken, a block can be considered as a "classical" part of a branch; i.e., a block is a part of a branch to which the full (classical) regularity restriction can be applied without losing completeness.

**Definition 3.9 (Block).** Let \( T \) be an ME-tableau generated by a DME-derivation for a clause set \( M \) and let \( b = \langle o_1, \ldots, o_n \rangle \) be a branch of \( T \). The sequence \( o_i, \ldots, o_j \) \((i > 1, j < n)\) is called a block of \( b \) iff

1. \( o_k \) is the immediate ancestor of \( o_{k+1} \) for \( i < k < j \),
2. \( o_i \) is a \( \delta \)-extension resulting node or was generated by an initialization step, and
3. each \( o_k \) with \( i < k < j \) is a \( \omega \)-extension node or a \( \omega \)-extension resulting node.

Condition (3) asserts simply that the literal attached to \( o_k \) stems from a \( \omega \)-clause in \( M \).

For illustration, consider Fig. 4. There, each node of a branch is represented by its label except the root node which is indicated by \( \bullet \). The start of a block is marked by typesetting the respective literal in boldface. (The purpose of underlining is explained below.)

Then, blockwise regularity is defined as follows.

**Definition 3.10 (Blockwise regularity).** Let \( T \) be an ME-tableau generated by a DME-derivation for a clause set \( M \). \( T \) is called blockwisely regular iff the following two conditions hold.

1. For each block \( o_1, \ldots, o_n \) of a branch of \( T \), \( \lambda(o_i) \neq \lambda(o_j) \) for all \( 1 \leq i < j \leq n \).
(2) For each branch \((o_1, \ldots, o_n)\) of \(T\), \(\lambda(prev(o_i)) \neq \lambda(prev(o_j))\) if \(o_i\) and \(o_j\) are \(\delta\)-extension resulting nodes for \(1 \leq i < j \leq n\).

We call a DME-derivation \(D\) blockwisely regular if the tableau generated by \(D\) is blockwisely regular.

Condition (2) asserts simply that there are no two literals on one branch which are proved by using \(\delta\)-clauses stemming from defaults having the same consequent.

The effect of blockwise regularity can be illustrated by the branches shown in Fig. 4, where the nodes characterizing the failure of blockwise regularity are underlined. An ME-tableau containing one of the first two branches need not be considered during a deduction, since these two branches violate condition (1) of Definition 3.10. The same holds for any ME-tableau containing the third branch since it does not meet condition (2) of Definition 3.10. This is because two \(\delta\)-clauses having the same "consequent" \(B_{\delta_1}\) and \(B_{\delta_2}\), respectively, have been used, namely \(\{\neg A_{\delta_1}, B_{\delta_1}\}\) and \(\{\neg D_{\delta_2}, B_{\delta_2}\}\). Observe that the two occurrences of \(\neg B\) on the third branch belong to different blocks. Only the rightmost branch does not violate any criteria for blockwise regularity.

The following theorem guarantees that every derivation generating a non-regular tableau can be pruned away without losing completeness.

**Theorem 3.4.** Let \(R\) be a DME-refutation of a clause set \(M\) such that the tableau \(T\) generated by \(R\) is not blockwisely regular. Then, there exists a DME-refutation \(R'\) for \(M\) such that the tableau \(T'\) generated by \(R'\) is blockwisely regular.

## 4. Model-based consistency checking

This section is devoted to the implementation of incremental consistency checking. According to the last section, this amounts to developing a mechanism for verifying compatibility of \(\delta\)-extension steps. For this, we start by taking an abstract point of view: we presuppose a procedure enumerating DME-derivations, along with a mechanism for
finding models of formulas in CNF. The usage of formulas in CNF as opposed to arbitrary ones is motivated by the need for continuous modifications (like additions and subsequent reductions) to the formulas handed over to the consistency check. These operations can be implemented more effectively for formulas in CNF.

Now, when checking consistency incrementally, and thus, repetitively, we should clearly avoid exhaustive general purpose mechanisms for reducing computational efforts. Our goal is therefore to furnish an approach that allows for pruning "inconsistent subproofs" while restricting exhaustive consistency checks to the ultimately necessary ones. We address this problem by means of a model-based approach: we use a model as a compact representation of the consistency of a (partial) default proof. In this way, a model bears witness to the compatibility of the involved $\delta$-extension steps. The aim is then to reuse such a model for as many subsequent compatibility checks as possible. Of course, this reusability depends on the chosen model. Hence, we sometimes encounter situations in which we have to look for a "better" model. We support this search by a potentially synergistic treatment of theorem proving and model handling. This approach relies on repeatedly reduced clause sets containing the initial facts and the justifications of the applied default rules (or to be more precise, the $\delta$-extension literals occurring in the current derivation). We refer to such clause sets as model-clause-sets because we exploit them as compact representations for their underlying set of models. Mutual benefits are then obtainable in the following way. First, each time a default rule applies, its justification is added to the model-clause-set. That is, for normal default rules, the $\delta$-extension literal of each $\delta$-extension step is added to this set. In addition, certain lemmas provided by the theorem prover are added. Both sorts of added formulas are then used to reduce the model-clause-set at hand. Conversely, a theorem prover may also benefit from the semantic account provided by a model for the current derivation for governing its proof search. This avenue has already proven to be of great value in classical automated theorem proving (cf. [85]). In what follows, however, we concentrate on realizing the two former issues, supporting the search for models by transferring information from the theorem prover to the "model finder".

First of all, let us make precise how we treat consistency checks via model handling: for a set of formulas $W$ and a sequence of default rules $(S_j)_{j<i}$ let $m$ be a model for $W \cup \text{Conseq}([\delta_0, \ldots, \delta_{i-1}])$. Function $\nabla$ checks whether $W \cup \text{Conseq}([\delta_0, \ldots, \delta_{i-1}]) \not\vdash \neg\text{Conseq}(\delta_i)$, as stipulated in condition (3) (and Definition 3.4):

$$\nabla(\delta_i, (m, W, (\delta_j)_{j<i})) = \begin{cases} \langle m, W, (\delta_j)_{j<i} \rangle & \text{if } m \models \text{Conseq}(\delta_i), \\ \langle m', W, (\delta_j)_{j<i} \rangle & \text{if } m \not\models \text{Conseq}(\delta_i) \text{ and for some } m' \neq m, \\ & m' \models W \cup \{\text{Conseq}(\delta_j) \mid j < i\}, \\ \bot & \text{if there is no } m'' \text{ such that } m'' \models W \cup \{\text{Conseq}(\delta_j) \mid j < i\}. \end{cases}$$

Function $\nabla$ gives a general description of our approach while making precise the intuition given above. We refine this specification in the sequel.

The following result shows that $\nabla$ provides a sound and complete specification of the consistency condition expressed in condition (3) in Definition 2.1 (and Definition 3.4):
Theorem 4.1. Let $W$ be a set of formulas and $\langle \delta_i \rangle_{i \in I}$ a sequence of normal default rules. Then, we have for all $i \in I$ that if there is a model $m$ of $W \cup \text{Conseq}(\langle \delta_0, \ldots, \delta_{i-1} \rangle)$, then there is either a model $m'$ of $W \cup \text{Conseq}(\langle \delta_0, \ldots, \delta_i \rangle)$ such that:

$$
\forall (\delta_i, (m, W, (\delta_j)_{j < i})) = (m', W, (\delta_j)_{j < i}) \iff W \cup \text{Conseq}(\langle \delta_0, \ldots, \delta_{i-1} \rangle) \not\models \lnot \text{Conseq}(\delta_i)
$$

or

$$
\forall (\delta_i, (m, W, (\delta_j)_{j < i})) = \bot \iff W \cup \text{Conseq}(\langle \delta_0, \ldots, \delta_{i-1} \rangle) \models \lnot \text{Conseq}(\delta_i).
$$

Observe that $m$ and $m'$ need not be distinct; thus covering the first two cases of $\forall$. At the start of a derivation, $m$ is set to an arbitrary model of the set of premises $C_W$.¹⁵

From a conceptual point of view, it is actually quite easy to read off such a model from the clause set by taking one literal from each clause, while never taking both a literal and its negation. Consider the $\omega$-clause set $C_W = \{\{\text{predispo}, \text{child}\}, \{-\text{predispo}, \neg\text{milk}, \text{allergen}\}\}$, given in (7). We obtain two models for $C_W$: $\{\text{predispo}, \text{child}, \neg\text{milk}\}$ and $\{\text{predispo}, \text{child}, \neg\text{allergen}\}$. Note that such models are actually partial models that only fix the truth-values of certain literals; hence, they are refineable along their degrees of freedom. We rely on this feature in the sequel.

Whenever a $\delta$-clause $\{-\gamma_3, \gamma_3\}$ is selected for a $\delta$-extension step in the course of a DME-derivation, we check whether $\gamma_3$ is satisfied by the current model $m$. In our setting, this can actually be done by simply checking whether $\gamma_3 \not\in m$, due to the nature of $m$ and $\gamma_3$. If this is the case, $\gamma_3$ is added to partial model $m$. In this way, we enforce that $m \cup \{\gamma_3\}$ is a model for

$$
C_W \cup \{\{\gamma_{i_1}\}, \ldots, \{\gamma_{i_t}\}\} \cup \{\{\gamma_3\}\},
$$

where $\gamma_{i_1}, \ldots, \gamma_{i_t}$ stand for the consequents of the previously used defaults. In DME-derivations, these are given by the $\delta$-extension literals occurring in the tableau at hand. This treatment amounts thus to the compatibility criterion imposed on $\delta$-extension steps in Definition 3.4. Otherwise, that is if $\gamma_3 \in m$, a new model for (8) has to be found for carrying on with the derivation. If no such model can be provided, $\delta$-clause $\{-\gamma_3, \gamma_3\}$ cannot be used in the current situation.

For reducing computational efforts of searching new models, we consider the aforementioned model-clause-sets, denoted by $M$, which are simplified yet equivalent variants of clause set (8). Coexisting models and model-clause-sets are therefore invariantly connected by the satisfiability relation, that is, a considered model always satisfies the current model-clause-set. At the start of a DME-derivation the model-clause-set equals $C_W$. During the DME-derivation it is then extended by the justifications of the used defaults and by certain lemmas provided by the theorem prover (see below). The principal idea is then to simplify $M$ after each such addition. Hence, in case a new model has to be generated, one does not have to start with the full clause set in (8) but rather a set which is already cut down as much as possible by appeal of previously gathered information.¹⁶,¹⁷ Each such

¹⁵ Such a model exists since we assume $W$ to be consistent (cf. Footnote 4).

¹⁶ Notably, such simplifications are doable in an anytime manner.

¹⁷ Note, however, that in case derivation steps have to be withdrawn, the corresponding modifications of the respective model-clause-sets have to be withdrawn, too.
simplification has to be model-preserving, i.e., a simplified clause set has to have the same partial models as the original one. In all, simplifications reduce the search space for easing possible further model generations by eliminating invalid or superfluous ones. In this paper, we restrict ourselves to unit-reductions and subsumption-deletion, both of which are well-known reduction techniques in automated theorem proving (cf. [23,53]) that can be carried out in polynomial time. While unit-reduction allows us to replace a clause \( \{L_1, \ldots, L_n\} \) by \( \{L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n\} \) in the presence of some unit clause \( \{L_i^e\} \), subsumption-deletion allows us to remove a clause \( \{L_1, \ldots, L_n\} \) in the presence of one of its proper subsets. In fact, both reductions are essential for Davis-Putnam- (and Resolution-) based proof procedures [28], one of which is also a part of our implementation (cf. Section 5). 18

For illustration, consider another set of statements dealing with nutrition allergies of children: "the considered child is usually lively", "normally, she is stressed if she is lively and must stay at home", and "normally, she becomes apathetic if she is stressed and scratches". Also, we know that she was at home and that she had sugar or milk. In her case, milk causes an allergic reaction, as does sugar under stress. Her allergy makes her scratch. This can be represented by the following default theory:

\[
D = \{ \text{lively}, \text{lively} \land \text{home} : \text{stress}, \text{stress} \land \text{scratch} : \text{apathetic} \},
\]

\[W = \{ \text{home, sugar} \lor \text{milk, milk} \rightarrow \text{allergy, stress} \land \text{sugar} \rightarrow \text{allergy, allergy} \rightarrow \text{scratch} \}.\]

For instance, we can explain why the considered child became apathetic via

\[
\{ \text{lively}, \text{lively} \land \text{home} : \text{stress}, \text{stress} \land \text{scratch} : \text{apathetic} \},
\]

The atomic format of our exemplary default theory \((D, W)\) is \((D', W')\), where

\[
D' = \{ \text{lively}, \text{R : stress}, \text{Q : apathetic} \}, \quad \text{and}
\]

\[
W' = W \cup \{ \text{lively} \land \text{home} \rightarrow R, \text{stress} \land \text{scratch} \rightarrow Q \}. \]

Let us illustrate our approach by explaining the child's apathy, apathetic, from theory \((D', W')\) by means of a DME-refutation for

\[
C_{W'} \cup C_{D'} \cup \{ \neg \text{apathetic} \}.
\]  

Figs. 5–9 give five snapshots of the interplay between DME-derivations and corresponding models and model-clause-sets. For brevity, we abbreviate in what follows the propositions in (10/11) by their two first letters in capitalized form.

As initial model of \(C_{W'}\), being the first model-clause-set \(M_0\), we take \(m_0 = \{ \text{HO, SU, AL, SC, LI, Q} \} \). \(M_0\) and \(m_0\) are given on the right of Fig. 5.

Intuitively, \(m_0\) is obtained from \(M_0\) by taking HO from the first clause in \(M_0\), SU from the second, AL from the third, etc.

Now, we start a DME-derivation by an initialization step with top-clause \(\neg \text{AP}\). The resulting tableau with open goal \(\neg \text{AP}\) is given on the left of Fig. 5.

18 An elaborated account on effective methods for simplifying clause sets used in the implementations of the Davis-Putnam procedure can be found in [34].
The second tableau in Fig. 6 emerges by applying four extension steps (one δ- and three ω-extension steps). Since the first δ-extension step uses δ-clause \{-Q_{δ_3}, AP_{δ_3}\}, model-clause-set \(M_1\) (the clause set in Fig. 6) contains clause \{AP_{δ_3}\}. (Observe that clause \{R, ¬LI, ¬HO\} has been reduced to \{R, ¬LI\} by unit-reduction.) Since \(m_0\) satisfies \(AP_{δ_3}\), or in other terms, \(V(δ_3, (m_0, W, \{\}))\) equals \(m_0, W, \{δ_3\}\), we may extend \(m_0\) to \(m_0' = m_0 \cup \{AP_{δ_3}\}\) and continue with \(m_0'\). Recall that while \(V\) deals with total models, we actually account for partial ones by extending them along their degrees of freedom; thus, the transition from \(m_0\) to \(m_0'\).

The next δ-extension step is applied to subgoal \(¬ST\) and it uses δ-clause \{ST_{δ_2}, ¬R_{δ_2}\} (see the tableau in Fig. 7). Since \(m_0'\) satisfies \(ST_{δ_2}\), we may proceed by simply extending \(m_0'\) to \(m_0'' = m_0' \cup \{ST_{δ_2}\}\). Furthermore, we add clause \{ST_{δ_2}\} to \(M_1\) and apply unit-reduction which gives us model-clause-set \(M_2\), the clause set in Fig. 7.

The following three extension steps are applied to subgoals \(¬R_{δ_2}, ¬LI, \) and \(¬HO\) (see the tableau in Fig. 8). The second one (a δ-extension step) uses δ-clause \{LI_{δ_1}\}. Notably, \(m_0''\) does not satisfy \(LI_{δ_1}\). Therefore, we have to search for a new model \(m_1\) of model-clause-set \(M_3\), which emerges from \(M_2 \cup \{LI_{δ_1}\}\) after the application of unit-reduction. Note that due to the rigorous application of unit-reductions, \(M_3\) comprises merely \(2^3 = 32\) potential models, as opposed to \(2^3 \times 3^3 = 216\) in the case of \(C_{W'}\). Formally, we get

\[
V(δ_1, (m_0', W, \{δ_3, δ_2\})) = (m_1, W, \{δ_3, δ_2, δ_1\}),
\]

with \(m_1 = \{HO, SU, AL, SC, R, Q, AP_{δ_3}, ST_{δ_2}, LI_{δ_1}\}\).
Now, the DME-derivation has reached a point where four subgoals, namely \(-\text{LI}, -\text{HO}, -\text{R}_2\), and \(-\text{ST}\), are proven. According to Definition 3.7, we have that:

- \(-\text{HO}\) is a DME-lemma with respect to \{\{\text{HO}\}\} and the empty set of defaults,
- \(-\text{LI}\) is a DME-lemma with respect to \{\{\text{LI}_1\}\} and \{\delta_1\},
- \(-\text{R}_2\) is a DME-lemma with respect to \{\{\text{HO}, \text{LI}_1\}, \{\text{R}, -\text{LI}, -\text{HO}\}\} and \{\delta_1\}, and
- \(-\text{ST}\) is a DME-lemma with respect to \{\{\text{HO}, \text{LI}_1\}, \{\text{R}, -\text{LI}, -\text{HO}\}, \{\text{ST}_{\delta_2}, -\text{R}_{\delta_2}\}\} and \{\delta_1, \delta_2\}.

In other words, we have shown that each open branch of a tableau generated by a DME-derivation from

\[ C_{\mathcal{W}} \cup \{\{\text{LI}_1\}, \{-\text{R}_{\delta_2}, \text{ST}_{\delta_2}\}, \{-\text{Q}_{\delta_3}, \text{AP}_{\delta_3}\}\} \]

(13)
can be marked as closed, if it contains one of \(-\text{Li}, -\text{HO}, -R_{\delta}, \) or \(-\text{ST}.\)

It is important to realize that this information can be exploited for reducing even more
the number of potential models in a model-clause-set. This is because each partial model
found in, for instance, \(M_3\) is also one in (13),\(^{19}\) or, in the other way round, each set of literals being no model of (13) cannot be a model of \(M_3\), either. Formally, we can capture this idea by the concept of DME-lemmas, as introduced in Section 3.2.1: for a DME-lemma \(l\) generated from some DME-derivation for a clause set \(M\), no partial model in \(M\) containing \(l\) as a subset has to be considered while searching for a model of \(M\). This is because each DME-derivation can be considered as a classical ME-derivation (see the considerations after Theorem 3.2). Hence, the generation of a DME-lemma \(l\) from some clause set \(M\) implies that \(-l\) is entailed by \(M\). As a consequence, each potential model containing \(l\) as a subset must be invalid and can be ignored.

For simplification issues, so-called unit lemmas consisting of singleton sets of literals
are of particular interest. In fact, after generating such a DME-lemma \(L\), the clause \(\neg L\) can be added to the respective model-clause-set and used for simplification. Other (non-unit) default lemmas can be used as a kind of constraints: while searching for a model in a model-clause-set, one can skip every potential model containing a DME-lemma as a subset. Moreover, there is no need for a consistency check, since all of these lemmas are drawn relative to default rules belonging to the current default proof. We discuss such so-called dynamic lemmas in Section 5.2.1.

Turning back to our running example, we could now take advantage of the generated
DME-lemmas \(\{\neg\text{Li}, \neg\text{HO}, \neg R_{\delta}, \}\) and \(\{\neg\text{ST}\}\. However, \(M_3\) already contains the corresponding (negated) unit clauses and therefore these particular DME-lemmas cannot be

\(^{19}\) This is because (i) each model-clause-set contains \(C_{\text{MC/}}\) (in a simplified form), and (ii) whenever a \(\delta\)-clause is used in a derivation, a subset of it is added to the respective model-clause-set.
used for further simplifications of $M_3$. Nevertheless, there are situations where the exploitation of DME-lemmas can be useful. Such a situation is given after the following three inference steps illustrated by the tableau in Fig. 9 (note that these steps do not use $\delta$-clauses). The reader may verify that now subgoals $\neg AL$ and $\neg SC$ are proven; hence we can add the corresponding negated unit clauses to $M_3$ (note that we do not have to extend $m_1$ since it already contains $AL$ and $SC$). In particular, $\neg AL$ turns out to provide a useful DME-lemma since adding the resulting unit clause \{AL\} to $M_3$ allows for two subsequent unit- and one subsumption-reductions. The resulting model-clause-set $M_4$ is the one in Fig. 9. Obviously, a new model (which is not needed in our example) could now be found with almost no efforts.

Finally, the sole remaining subgoal which has not been treated so far, viz. $\neg ST$, can be proven via the previously generated DME-lemma \{ST\}.

In all, we have thus shown that there exists a DME-refutation for clause set (12) with top clause \{\neg AP\}. The compatibility of the three involved $\delta$-extension steps was warranted by models $m_0$ ($m_0'/m_0''$) and $m_1$. We have thus shown that AP can be explained by means of default proof (9) from $(D', W')$.

Apart from the reuse of models as compact representations of previous consistency checks, the last example puts emphasis on simplifications of the model-clause-set. In fact, such simplifications are doable in an anytime manner. And only the addition of lemmas depending on default assumptions are subject to backtracking (since classical lemmata can be used independently from the set of $\delta$-clauses used in a derivation). This leaves room for different implementation strategies. One extreme is to trigger simplifications on each item added to the model-clause-set (as shown above to put forward the approach in principle). Another alternative is to do this by need, that is, only if enforced by a search for a new model. The first moulding is arguably very appealing in a parallel setting, since then the anytime property allows for continued simplifications without putting brakes on the actual inference engine. In a purely sequential setting, as ours, however, the second variant seems more appropriate, since the simplification efforts, though polynomial, are restricted to the ultimately necessary ones. This is why we have chosen the latter option for implementing our model-based approach to consistency checking, as shown in the next section.

5. Implementing query answering from default theories

The key observation that led to the results reported in this section is that the approach for query answering presented in the previous sections allows us to apply Prolog technology in a rather straightforward manner. This is due to the fact that model elimination is closely related to Prolog's linear input resolution [53], which renders it especially qualified for implementation by means of so-called Prolog Technology Theorem Provers (PTTPs) [86–88]. As regards classical theorem proving, this approach has already resulted in quite impressive high-performance proof systems, like [9,41,86]. The main reason for their convincing performance is their ability to take advantage of highly efficient underlying Prolog systems (or similar systems based on the Warren Abstract Machine [1]).

In what follows, we describe how Prolog technology can be used for implementing default reasoning systems. As regards classical theorem proving, we follow the approach
taken by Stickel’s PTTP [87,88]. PTTP can be seen as an extension of Prolog that provides a proof system for full first-order predicate calculus. In order to attain this, one has to enhance Prolog via measures guaranteeing (i) sound unification, (ii) complete search, and (iii) complete inference. In what follows, we concentrate on the last item, since we have restricted our exposition to the propositional case. In fact, there is no need for modifying the classical methods addressing the first two issues when dealing with default reasoning, so that we refer the reader for details to [87].

5.1. Implementation by an extended Prolog compiler

The standard approach is based on the idea to transform a theory \( W \) along with a query \( \varphi \) into a Prolog program \( P_W \) and a Prolog query \( Q_\varphi \) such that \( Q_\varphi \) is derivable from \( P_W \) iff \( W \land \neg \varphi \) is unsatisfiable. In this way, the Prolog inference mechanism remains unchanged, while the transformation has to guarantee the implementation of items (i)–(iii). Although this approach is conceptually very simple it has proven to be very successful.

The first step of Stickel’s transformation is a direct translation of a given set of input clauses into a set of Prolog rules. Since a Prolog rule \( a \leftarrow b \) can be read as an implication \( a \land b \), a first idea would be to transform a clause like \( \{a, \neg b\} \) into such a Prolog rule. More generally, a clause \( \{L_1, \ldots, L_n\} \) can be transformed into an implication \( L_i \leftarrow L_i^c \land \cdots \land L_{i-1}^c \land L_{i+1}^c \land \cdots \land L_n^c \), which is referred to as a contrapositive of the original clause. \(^{20}\) However, in order to guarantee completeness of reasoning by contraposition, one has to generate a Prolog rule for each contrapositive of a clause at hand. Our exemplary clause \( \{a, \neg b\} \) results thus in two contrapositives \( a \leftarrow b \) and \( \neg b \leftarrow \neg a \).

For further illustration, consider the classical formulas in (4). While child and predispo are turned into a single contrapositive \( \text{child} \leftarrow \text{predispo} \), respectively, we obtain three contrapositives from \( \{\neg \text{milk}, \neg \text{predispo}, \text{allergen}\} \), namely \( \text{allergen} \leftarrow \text{milk} \land \text{predispo} \), \( \neg \text{milk} \leftarrow \neg \text{allergen} \land \text{predispo} \), and \( \neg \text{predispo} \leftarrow \neg \text{allergen} \land \text{milk} \). Following PTTP’s translation schema [87], this amounts to the following Prolog rules: \(^{21}\)

\[
\begin{align*}
\text{child}. \\
\text{predispo}. \\
\text{allergen} & : - \text{milk}, \text{predispo}. \\
\text{not}_\text{milk} & : - \text{not}_\text{allergen}, \text{predispo}. \\
\text{not}_\text{predispo} & : - \text{not}_\text{allergen}, \text{milk}.
\end{align*}
\]

We refer to such rules as \( \omega \)-rules. Prolog’s treatment of each \( \omega \)-rule corresponds to an \( \omega \)-extension step using the original clause as input clause, while closing the branch labeled with the head literal. For a set containing \( n \) clauses whose maximal cardinality is \( m \), we thus obtain at most \( n \times m \) \( \omega \)-rules.

Taking into account that Prolog’s inference rule corresponds to the classical ME-extension step, one obviously has to provide additional means for complete inference (i.e., means for reasoning by cases). This is actually accomplished by extending the translation in order to integrate reduction steps (see Section 3.1). Following PTTP’s translation schema

\(^{20}\) Taking commutativity of \( \land \) into account, a clause consisting of \( n \) literals has thus \( n \) contrapositives.

\(^{21}\) Following PTTP’s syntax, allergen and not_allergen stand for allergen and \( \neg \text{allergen} \) etc.
[87], this is achieved in the following way. First, one has to memorize ancestors goals by adding an additional argument (a list \( \text{Anc} \)) to each literal (see left column below). We refer to this list as the ancestor list. Second, for each literal that is added in some rule to the ancestor list (like allergen or not_milk below) a further rule must be added in order to allow for testing whether a subsequent subgoal is complementary to this (possibly memorized) literal. These additional rules are given in the right column below. In our example, we thus get the following rules.

\[
\begin{align*}
\text{child}(\text{Anc}). \\
\text{predispo}(\text{Anc}). \\
\text{allergen}(\text{Anc}) :- \\
\quad \text{not_allergen}(\text{Anc}) :- \\
\quad \quad \text{NewAnc} = [\text{allergen}|\text{Anc}], \\
\quad \quad \quad \text{member}(\text{allergen}, \text{Anc}). \\
\quad \quad \text{milk}(\text{NewAnc}), \\
\quad \quad \text{predispo}(\text{NewAnc}). \\
\text{not_milk}(\text{Anc}) :- \\
\quad \text{milk}(\text{Anc}) :- \\
\quad \quad \text{NewAnc} = [\text{not_milk}|\text{Anc}], \\
\quad \quad \quad \text{member}(\text{not_milk}, \text{Anc}). \\
\quad \quad \text{not_allergen}(\text{NewAnc}), \\
\quad \quad \text{predispo}(\text{NewAnc}). \\
\text{not_predispo}(\text{Anc}) :- \\
\quad \text{predispo}(\text{Anc}) :- \\
\quad \quad \text{NewAnc} = [\text{not_predispo}|\text{Anc}], \\
\quad \quad \quad \text{member}(\text{not_predispo}, \text{Anc}). \\
\quad \quad \text{not_allergen}(\text{NewAnc}), \\
\quad \quad \text{milk}(\text{NewAnc}).
\end{align*}
\]

For a clause set over an alphabet counting \( l \) letters, we thus obtain at most \( 2 \times l \) additional Prolog rules. As shown in [88], the aforementioned compile time transformations provide a proof system for propositional clause logic. (For brevity, we refrain from adding Prolog rules implementing regularity checks and lemma handling, as discussed in Section 3.2. We address these issues separately in Section 5.2.)

Let us now turn to the transformations needed for integrating default rules. We have seen in Section 3.1, how a classical ME-based calculus can be adapted for default reasoning, namely (i) by adding a restricted extension step for \( \delta \)-clauses and (ii) by restricting the classical definition of reduction steps. In what follows, we advance PTTP’s translation schema further in order to incorporate these two enhancements.

As pointed out in Section 3.1, the definition of \( \delta \)-extension steps reflects the fact that defaults are inference rules by demanding that only literals corresponding to consequents of defaults are used as \( \delta \)-extension literals. This is actually what distinguishes them from conventional \( \omega \)-extension steps. By taking this difference into account, we may thus treat \( \delta \)-clauses and \( \omega \)-clauses in an analogous way. In fact, the translation of a \( \delta \)-clause \( \{ \neg \alpha_5, \gamma_5 \} \) equals that of a \( \omega \)-clause, except that only one of its contrapositives is considered for transformation, namely \( \gamma_5 \leftarrow \alpha_5 \). This yields a single Prolog rule \( \gamma_5 : - \alpha_5 \). The

\[
\text{For further examples, take any second Prolog rule of any procedure in Fig. A.2.}
\]

\[
\text{Note that we are not yet talking of compatible \( \delta \)-extension steps at this stage.}
\]
restriction to a single contrapositive renders reason by contraposition ineffective because it refuses inferences with $\delta$-clauses like $\{-\alpha_5, \gamma_3\}$ to solve an open goal like $\alpha_3$ (thus preserving $\delta$’s inference rule character). We refer to the resulting Prolog rules as $\delta$-rules and sometimes abbreviate a $\delta$-rule stemming from a default $\zeta$ by $r_\zeta$. In our example, we obtain for the default rules in (4) the following Prolog rules:

\begin{verbatim}
icecream :- child.
milk :- icecream.
sugar :- icecream.
\end{verbatim}

In analogy to $\omega$-rules, each such $\delta$-rule describes a possible transition between two tableaux by means of a $\delta$-extension step using the underlying default rule. For a clause set containing $d$ $\delta$-clauses, we thus obtain at most $d$ $\delta$-rules.

As with ordinary clauses, we have to provide means for applying reduction steps. As above, we add an additional argument memorizing ancestor subgoals. But instead of extending the resulting list by the head literal $\gamma_3$ of the considered $\delta$-rule, the ancestor list is set to the empty list for avoiding reduction steps using ancestor subgoals of $\gamma_3$. In this way, the resulting Prolog program implements the restriction of classical reduction steps given in Definition 3.5. For the $\delta$-clauses in our example, we thus get:

\begin{verbatim}
icecream(Anc) :- child([]).
milk(Anc) :- icecream([]).
sugar(Anc) :- icecream([]).
\end{verbatim}

The next step in our transformation provides means for checking consistency by guaranteeing compatibility for $\delta$-extension steps. We have seen in Section 4 that a single (partial) model $m$ can be used as a compact representation of the consistency of a default proof. For implementing this model-based approach to consistency checking, we add a further argument to each generated Prolog rule, containing among others a (partial) model $m$ for $C_W \cup \{\gamma_j \mid j < i\}$ provided that $\delta$-rules $r_{\delta_1}, \ldots, r_{\delta_{i-1}}$ have been successfully applied in the derivation up to this point. In this way, $m$ guarantees that all $\delta$-extension steps effectuated via $\delta$-rules $r_{\delta_1}, \ldots, r_{\delta_{i-1}}$ have preserved compatibility.

When testing compatibility at $\delta$-rule $r_{\delta_i}$, we check whether $\gamma_{\delta_i}$ is satisfied by $m$. (Note that such a satisfiability test is linear in the size of $\gamma_{\delta_i}$.) If this easy test succeeds, we continue by extending partial model $m$ in order to account for $\gamma_{\delta_i}$. If not, we try to generate a new model $m'$ satisfying

\begin{equation}
C_W \cup \{\gamma_{\delta_i} \mid j < i\} \cup \{\gamma_{\delta_i}\}.
\end{equation}

If this succeeds, we continue proving with $m'$. Otherwise, backtracking is engaged. This proceeding guards the compatible application of subsequent $\delta$-extension steps according to the specification of function $\nabla$ given in Section 4.
However, in order to be able to generate new models for dynamically increasing and decreasing clause sets of form (14), we need an extremely flexible representation for the corresponding model-clause-sets. This is why we have chosen to propagate model-clause-sets along with the current model throughout the ongoing derivation instead of compiling them into rigid Prolog code.

As a consequence, function $\triangledown$ is implemented by the run-time predicate compatible/3, which—as its name suggests—verifies whether a $\delta$-extension step with $\delta$-rule $r_{\delta_i}$ is compatible or not. The first argument comprises the justification of $\delta_i$ given by $\gamma_{\delta_i}$. While the second argument encapsulates the current model $m$ along with the current model-clause-set, the third argument contains the same information, yet enriched by the information gathered after a successful application of $r_{\delta_i}$. In case of failure, that is if $m \models \neg \gamma_{\delta_i}$ and no new model $m'$ of (14) can be found from the model-clause-set, predicate compatible/3 fails and backtracking is engaged.

In all, we thus need two physical variables ($M$ and $NewM$) for propagating models and two physical variables ($MM$ and $NewMM$) for propagating model-clause-sets throughout a derivation. Because of the tight relationship between models and model-clause-sets, we regroup them by means of binary functor $m/2$. This yields the following $\delta$-rules in our example:

\[
\text{icecream}(\text{Anc},m(M,MM),m(NewM,NewMM)) :- \\
\quad \text{child}([],m(M,MM),m(M1,MM1)), \\
\quad \text{compatible}(\text{icecream},m(M1,MM1),m(NewM,NewMM)) .
\]

\[
\text{milk}(\text{Anc},m(M,MM),m(NewM,NewMM)) :- \\
\quad \text{icecream}([],m(M,MM),m(M1,MM1)), \\
\quad \text{compatible}(\text{milk},m(M1,MM1),m(NewM,NewMM)) .
\]

\[
\text{sugar}(\text{Anc},m(M,MM),m(NewM,NewMM)) :- \\
\quad \text{icecream}([],m(M,MM),m(M1,MM1)), \\
\quad \text{compatible}(\text{sugar},m(M1,MM1),m(NewM,NewMM)) .
\]

Observe that pairs like $m(M,MM)$ correspond to those used in Figs. 5–9, like for instance $M_1$ and $M_3$.

Notably, our implementation allows us to decide at compile time whether we check compatibility before or after finding a proof of the original prerequisite. From a $\delta$-rule $\gamma_{\delta} \leftarrow \alpha_{\delta}$ we may thus generate one of the following Prolog rules depending on whether we choose, say, compiler option $\alpha$-ccm or $\alpha$-com:

\[
(\alpha\text{-ccm}) \quad \gamma_{\delta} : - \alpha_{\delta}, \text{compatible}(\gamma_{\delta})
\]

\[
(\alpha\text{-com}) \quad \gamma_{\delta} : - \text{compatible}(\gamma_{\delta}), \alpha_{\delta}.
\]

Under option $\alpha$-ccm, Prolog tries to find a proof of the prerequisite $\alpha_{\delta}$ before checking compatibility of $\gamma_{\delta}$, whereas under $\alpha$-com compatibility is checked first.

\[24\text{ Actually, our implemented system deals with default components in negation normal form. Consequently, the first argument contains the CNF of the justification. See also Section 5.2.4.}\]
Turning back to our example, the first of the above Prolog rules is replaced under com-α by

\[
\text{icecream}(\text{Anc}, \text{m}(M, MM), \text{m}(\text{NewM}, \text{NewMM})) :- \\
\text{compatible}(\text{icecream}, \text{m}(M, MM), \text{m}(\text{M1}, \text{MM1})), \\
\text{child}([], \text{m}(\text{M1}, \text{MM1}), \text{m}(\text{NewM}, \text{NewMM})).
\]  

A detailed empirical analysis of both options is given in Section 5.3. We just report here that a preceding verification of compatibility (com-α) turns out to be extremely valuable for pruning the search space in knowledge bases comprising a high number of potential conflicts. For instance, for deciding Hamiltonian cycle problems, this compiler option leads to much better results than a belated compatibility check (α-com). The inverse can be observed, for example, on terminological knowledge bases, where the search is more or less guided by classical inferencing.²⁵

From a schematic perspective, predicate \text{compatible}/3 can be defined in our simplistic setting ²⁶ by the following two Prolog clauses:

\[
\text{compatible}(K, \text{m}(M, MM), \text{m}(\text{NewM}, \text{NewMM})) :- \\
\text{negated_literal}(K, \text{NOT}_K), \\
\text{not member}(\text{NOT}_K, M), \\
\text{reduced_clause_set}([[\text{M}]], \text{NewMM}).
\]

\[
\text{compatible}(K, \text{m}(M, MM), \text{m}(\text{NewM}, \text{NewMM})) :- \\
\text{negated_literal}(K, \text{NOT}_K), \\
\text{member}(\text{NOT}_K, M), \\
\text{reduced_clause_set}([[\text{K}]], \text{NewMM}), \\
\text{model}(\text{NewMM}, \text{NewM}).
\]

(Run-time predicate \text{negated_literal}(X, Y) is true if Y is bound to the negation of X. Run-time predicate \text{member}/2 implements the common membership-relation. Clause sets are represented by lists of lists of literals.)

To begin with, we observe that the combination of variable K and term \text{m}(M, MM) amounts to the information passed to function \text{V} in Section 4 via \text{V}(\delta, \langle m, W, \langle \text{Conseq}(\delta_j) \rangle_{j \in J} \rangle). While K stands for the justification γₖ of δ (i.e., \text{Conseq}(\delta) = γₖ), M and MM represent the current model m along with the current model-clause-set M providing a compact representation of \textit{W} U \{\text{Conseq}(\delta_j) \mid j \in J\}. This is sufficient because \text{V} deals only with the justifications (or consequents, respectively) of the considered (normal) default rules.

The first clause handles the case where K is satisfied by model M. In our simplistic setting, this can be tested by \text{not member}(\text{NOT}_K, M). In addition, model-clause-set MM is extended by unit clause \text{[K]} and afterwards reduced by run-time predicate \text{reduced_clause_set}/2 (involving unit-reductions and subsumption-

²⁵ In fact, the gain in the latter case is not that drastic as in the former case; on the other hand, the inherent complexity of the problem set is also decreasing.

²⁶ This involves (i) one literal justifications (due to atomic format) as opposed to general ones in conjunctive normal form, (ii) closed formulas as justifications; these restrictions are unleashed in our system. See [59] for details.
deletions, as described in Section 4). Finally, \( \gamma_8 \) is added to \( m \) (neglecting multiple occurrences) through Prolog's head matching. In case \( \neg \gamma_8 \in m \), the second Prolog clause steps in: a new model NewM is computed via run-time predicate model/2 from model-clause-set NewMM, which results from \([K]\) and \(MM\) by applying the same reductions as in the first clause.

This proceeding is in accord with our principled approach allowing for continuous simplifications of the model-clause-set. As anticipated at the end of Section 4, we actually pursue a more pragmatic approach in our (current) implementation that simplifies only by need. Thus, no simplification is done whenever a model is reusable, as in the first case. This yields the following alternative pattern for the first clause:

\[
\text{compatible}(K,m(M,MM),m([K]|M),[[K]|MM])) :-
\text{negated-literal}(K,NOT_K),
\text{not member}(NOT_K,M).
\]

This pragmatic solution is justified by the fact that we observe on many examples rather few model switches. Investigations with a profiler showed that in such cases considerable CPU-time is consumed by simplification procedures.

For finding new models (via run-time predicate model/2), we use an adapted variant of the Davis–Putnam procedure [28], which is currently one of the fastest complete methods for finding propositional models. Importantly, this task is supported by repeated reductions of the model-clause-set by using information gathered during the proof search. This may lead to a drastic reduction of the search space for finding new models, as illustrated in Section 4. Further implementation-based improvements are detailed in [21] (see also [83]).

The propagation of models and model-clause-sets affects also the Prolog rules stemming from “classical” clauses, which completes the resulting Prolog program:

\[
\text{child}(\text{Anc},MMM,MMM).
\text{predispo}(\text{Anc},MMM,MMM).
\text{allergen}(\text{Anc},MMMI,MMMO) :-
\text{NewAnc} = [\text{allergen}|\text{Anc}],
\text{milk}(\text{NewAnc},MMMI,MMMI),
\text{predispo}(\text{NewAnc},MMMI,MMMO).
\text{not_milk}(\text{Anc},MMMI,MMMO) :-
\text{NewAnc} = [\text{not_milk}|\text{Anc}],
\text{not_allergen}(\text{NewAnc},MMMI,MMMI),
\text{predispo}(\text{NewAnc},MMMI,MMMO).
\text{not_predispo}(\text{Anc},MMMI,MMMO) :-
\text{NewAnc} = [\text{not_predispo}|\text{Anc}],
\text{not_allergen}(\text{NewAnc},MMMI,MMMI),
\text{milk}(\text{NewAnc},MMMI,MMMO).
\]

The above material provides a (simplified) recipe for transforming a default theory \((D,W)\) into a Prolog program \(PD,W\). For query answering, however, we have to provide a further transformation for queries, like \(\varphi\). Following PTTP, we use for this purpose a
special predicate query along with a Prolog rule of the form query :- \varphi. Basically, the application of this rule is similar to an initialization step in DME-derivations. For our treatment of consistency, however, it has to be enriched by run-time predicate \texttt{model/2} which generates an initial model of \( C_W \). Furthermore, the rule has to initialize the ancestor list (which is set to the empty list) and to propagate the model generated by \texttt{model/2}.

In our example (with query allergen), we thus get schematically the following Prolog rule, where \( C_W \) represents a reduced variant of \( C_W \):

\[
\text{query} :- \\
  \text{model}(C_W, M), \\
  \text{allergen}([], m(M, C_W), m(_, _)).
\] (16)

This rule allows us to pose our initial query allergen via Prolog query ?-query.

In all, we thus compile a default theory \((D, W)\) along with a query \( \varphi \) into a Prolog program \( P_{D,W,\varphi} = P_{D,W} \cup \{(16)\} \) along with a Prolog query \texttt{query}. The resulting program is compilable using a standard Prolog compiler which leads to its impressive performance. Subsequent queries are easily posed by replacing and recompiling the single query rule only.

Finally, let us give an estimate on the number of resulting Prolog rules obtained from the compilation of an atomic default theory \((D, W)\) along with query \( \varphi \) over alphabet \( \Sigma \). Let \( C_W \) contain \( n \) clauses whose maximal cardinality is \( m \), let \( \Sigma \) count \( l \) propositional symbols and let \( D \) be of cardinality \( d \). The estimate of the resulting code is summarized in Fig. 10. We refine this first estimate in Section 5.2.6, where we incorporate further rules stemming from refinements of our approach.

For an impression, consult Appendix A containing some examples treated in this paper along with the resulting "object-code".

5.2. Extensions, refinements and implementation

The previous section has presented our basic approach to implementing default reasoning by taking advantage of the power provided by PTPP. Our current implementation refines this basic approach in several ways in order to improve its flexibility and efficiency. This section summarizes the most important improvements.
5.2.1. Lemma handling

We have already stressed in Section 3.2.1, the importance of lemma handling as a means for eliminating redundancy in automated theorem proving. We have also seen that this task is more difficult in our context, since proofs may depend on default rules.

We can actually distinguish between two independent subtasks for lemma handling: generation and usage of lemmas:

For generating lemmas, we must have knowledge on the subderivations used for deriving them. This is because we must know (i) which subgoals have been solved by reduction steps using ancestor goals outside the considered subderivation and (ii) which consistency assumptions have been made during δ-extension steps. This requires recording the subderivations for each proposition to be lemmatized. This is done by means of PTTP's proof recording facilities, used in standard PTTP for recording the (overall) proof of the query. For this purpose, further arguments must be added to each Prolog predicate in order to account for subderivations; this is complemented by further runtime Prolog code for extracting the relevant information from these subderivations.

Lemma generation is then accomplished by passing a proven subgoal along with information gathered from its subderivation to a predicate lemmatize, which encapsulates the actual lemmatization proceeding. Concretely, we add lemma generation to Prolog rules by simply attaching predicate lemmatize to the end of the rules' bodies. At this location of a rule's body, we are in possession of the entire subderivation, since all subgoals have been proven.

For example, in the case of the Prolog rule headed by allergen, we obtain without detailing the extant construction the following rule:

\[
\text{allergen} :- \\
\quad \text{milk}, \\
\quad \text{predispo}, \\
\quad \text{lemmatize}(\text{allergen}).
\]

For propagating subderivations, allergen, milk, and predispo are then extended by two additional variables whose outcome is then passed to lemmatize(allergen) by extending it appropriately. (The entire rule is then of course subject to further compilations.) Note that lemma generation does thus not add entire rules to the resulting Prolog code, but rather a single subgoal for each proposition to be lemmatized.

This last subgoal, being composed in its final form of predicate lemmatize, the actual lemma, and its proof, does the aforementioned subproof analysis and then dumps the generated lemma along with the information extracted from its proof into the Prolog database (by means of Prolog's standard predicate assert/1), so that it becomes available for later usage. The dumped items have the following format:

\[
\text{lemma}(\text{Goal}, \text{Ancestors}, \text{Assumptions}).
\] (17)

This signifies that Goal has been proven by means of reductions with ancestor goals Ancestors under consistency assumptions Assumptions. The lemma itself is then given by the disjunction of Goal and all literals in Ancestors. That is in terms of Section 3.2, a lemma item such as \text{lemma}(L, \{L_1, \ldots, L_n\}, \{\beta_{\delta_1}, \ldots, \beta_{\delta_k}\}) represents a DME-lemma \{L, L_1, \ldots, L_n\} with respect to the set of default rules \{\delta_1, \ldots, \delta_k\}.
For using lemmas, we simply add terminating Prolog rules in front of each Prolog procedure. For resolving `allergen` via lemma usage, for example, we thus add schematically a Prolog rule

```
allergen :-
    lemma(allergen).
```

at the top of procedure `allergen`.

For the most part, predicate `lemma` looks up the Prolog database for corresponding lemmas; it actually matches in its final form with the lemma items given in (17). Once such an item is retrieved, we must (usually\(^{27}\)) take into account the consistency requirements attached to the stored lemma; this is done via model handling, as described in Sections 4 and 5. According to the compilation steps of Section 5, we thus obtain the following rule: \(^{28}\)

```
allergen(Anc,m(M,MM),m(NewM,NewMM)) :-
    lemma(allergen,Ancestors,Assumptions),
    compatible(Assumptions,m(M,MM),m(NewM,NewMM)).
```

As above, the entire rule is then subject to further compilations. Among them, those dealing with disjunctive lemmas, addressed by checking whether the ancestor goals in `Ancestors` form a subset of those in `Anc`. \(^{29}\)

In all, this feature adds at most \(2 \times l\) additional Prolog rules, where \(l\) is the size of the underlying alphabet.

In our current implementation, we particularly take care of the fact that using lemma mechanisms in an unrestricted fashion leads to the generation of a flood of useless lemmas swamping the storage. To this end, we employ techniques which have their roots in successful approaches to restrict lemma usage in classical theorem proving.

First, we actually distinguish between static and dynamic lemmas (cf. [51]).

Dynamic lemmas are temporary lemmas that disappear through backtracking: whenever a lemma is derived during a deduction (via some subderivation), it can be used in subsequent derivation steps but expires as soon as its own subderivation is tracked back. Hence, the number of available dynamic lemmas is limited by the number of proved subgoals in the tableau under consideration. Notably, the consistent use of such lemmas is warranted by the presence of its (compatible) subderivation. Thus, valid dynamic lemmas are usable without any consistency checks. \(^{30}\) For dynamic lemma handling the last Prolog rule can thus be replaced by the following one:

```
allergen(Anc,MMM,MMM) :-
    lemma(allergen,Ancestors,_).
```

\(^{27}\) This applies to static lemmas only; see below for further details.

\(^{28}\) Anticipating the treatment of general default rules in Section 5.2.4, `Assumptions` represents actually a clause set.

\(^{29}\) While our actual implementation allows for generating disjunctive lemmas, we do actually never use them. We rather concentrate on unit lemmas, whose usage is much more effective. See below.

\(^{30}\) Hence the application of a dynamic lemma has never to be tracked back (see Section 5.2.3).
Experiments have shown that the use of dynamic lemmas may result in a significant speed-up, while they practically never harm the proof search due to their restricted viability. This is illustrated in Section 5.3. The use of dynamic lemmas is actually related to a technique used for classical ME-based theorem proving, namely the (highly effective) folding-up technique [48] (or C-reduction [84]).

As opposed to this, static lemmas are kept along with their underlying consistency assumptions throughout a whole deduction; this requires verifying compatibility each time such a lemma is used. These lemmas have thus to be dealt with carefully, since they may lead to the aforementioned flood of useless lemmas swamping the storage.

Second, our implementation allows to restrict the generation of lemmas to so-called unit lemmas, which consist of one literal only. Due to the fact that the application of unit lemmas does not depend on the existence of suitable ancestor goals, such lemmas are clearly more effective than ordinary ones. Furthermore, their application (in particular if dynamic lemmas are considered) can be checked very efficiently (with a simple lookup in the database). For dynamic unit lemma handling the last Prolog rule can thus again be simplified:

\[
\text{allergen}(\text{Anc}, \text{MMM}, \text{MMM}) :\neg \\
\text{lemma}(\text{allergen}, [], \_).
\]

In the field of classical theorem proving, unit lemmas have shown to be inevitable for strengthening the deductive power of ME-based proof systems (e.g., see [14]).

5.2.2. Loop checking by blockwise regularity

As described in Section 3.2.2, regularity provides a highly efficient means for discarding subgoals identical to one of their ancestor subgoals in proof systems based on model elimination.

As with lemmas, we had to adapt this tool for default reasoning. This led us in Section 3.2.2 to what we called blockwise regularity requiring (i) that each block of a branch must not contain two identical literals and (ii) that every branch must not contain two identical literals to which a S-extension step has been applied.

Both conditions can be easily implemented via ancestor lists (as already put forward in [87] for classical regularity). For verifying condition (i) we make use of the fact that the literals of a block are exactly the literals which can be used for reduction steps. Hence, we simply put in front of each Prolog procedure a further terminating Prolog rule clause that checks whether the ancestor list memorizing potential candidates for reduction steps contain two identical literals. For instance,

\[
\text{allergen}(\text{Anc}) :\neg \\
\text{member}(\text{allergen}, \text{Anc}), \\
\_, \text{fail}.
\]

is put in front of the procedure allergen. For further examples, take any first Prolog rule of any procedure in Fig. A.2.

For verifying condition (ii), we have to check that after using a δ-rule \( \gamma_\delta :\neg a_\delta \), no δ-rule with head \( \gamma_\delta \) is used to prove \( a_\delta \). To this end, we must actually add another list memorizing
the ancestors of \( \delta \)-extension resulting literals, and provide further clauses (similar to the ones for checking condition (i)) which prevent derivations violating condition (ii).

As with implementing reduction or lemma steps, this feature adds at most \( 2 \times l \) additional Prolog rules, where \( l \) is the size of the alphabet.

5.2.3. Avoiding useless backtracking

For propositional clauses it is well known that reduction steps and extension steps with unit clauses need not to be tracked back. That is, if it is possible to apply such a derivation step to some open goal \( G \) in a tableau \( T \), one does not have to check derivations that apply another derivation step to \( G \). The same holds for the application of dynamic lemmas: if a branch with open goal \( G \) in a tableau \( T \) can be marked as closed due to the application of a dynamic lemma, there is no need to check any other derivation that applies a reduction or extension step to \( G \) in \( T \). This is because a dynamic lemma \( l \) can only be applied to a tableau \( T \) whose corresponding derivation contains for each \( \delta \)-rule \( r \), used in the derivation of \( l \), at least one \( \delta \)-extension step with \( r \). Hence, the application of \( l \) to \( T \) cannot restrict the application of further \( \delta \)-extension steps.

Fortunately, both enhancements are easily implementable using Prolog's cut: one only has to guarantee that (i) the Prolog-clauses implementing reduction steps, extension steps with unit clauses, and lemma usage are put in front of the Prolog clauses implementing extension steps, and (ii) that each of these clauses ends with "! . ".

5.2.4. General default rules

For simplicity, our presentation was so far dominated by default rules having atomic components only. As PTTP, however, our implementation deals with formulas in negation normal form. Let us illustrate this by detailing the transformation of default rule

\[
A \land (\neg B \lor C) : -D \lor E
\]

To begin with, note that translating this rule into its atomic format yields

\[
\delta_{42} = \begin{array}{c}
A \land (\neg B \lor C) \rightarrow \neg D \lor E
\end{array}
\]

along with \( \neg D \lor E \). This is motivated by the fact that implications of form \( \beta_{42} \rightarrow \text{Justif}(\delta_{42}) \) concern the consistency check only. These rules are therefore not transformed themselves but rather woven into the \( \delta \)-rule stemming from the atomic default rule.

In general, we thus generate for any default rule \( \delta_{i} \), exactly one \( \delta \)-rule of the following format:

\[
\gamma_{i} := \begin{array}{c}
\alpha_{i}, \text{justification(Justif}(\delta_{i}))
\end{array}
\]

(18)
Among the further compilation steps, the term formed by means of justification/1 is turned into a call to procedure compatible/3 (see below).

Although the two remaining implications, viz. \((A \land (\neg B \lor C)) \rightarrow \alpha_{\delta_2}\) and \(\gamma_{\delta_2} \rightarrow \neg D \lor E\), can now be treated as ordinary \(\omega\)-clauses, we make use of the fact that \(\alpha_{\delta_2}\) and \(\gamma_{\delta_2}\) are new atoms occurring at particular places only. For instance, normally, \(\omega\)-clause \(\{\neg \gamma_{\delta_2}, \neg D, E\}\) would give rise to three contrapositives \((\neg \gamma_{\delta_2} \leftarrow D \land \neg E), (\neg D \leftarrow \gamma_{\delta_2} \land \neg E), (E \leftarrow \gamma_{\delta_2} \land D)\). The first contrapositive is however redundant, since there is never any rule having \(\neg \gamma_{\delta_2}\) among its body literals. Generally, we may thus leave out all contrapositives with head \(\neg \gamma_{\delta_2}\) that are obtained from \(\gamma_{\delta_i} \rightarrow \text{Conseq}(\delta_i)\).

A similar argument shows that we may eliminate all contrapositives, having \(\neg \alpha_{\delta_i}\) among their body literals, because there are no rules with head \(\neg \alpha_{\delta_i}\). Since PTTP supports furthermore rule-bodies in negation normal form, we obtain in our example directly the contrapositive \(\alpha_{\delta_2} \leftarrow A \land (\neg B \lor C)\). In general, we thus get for any default \(\delta_i\) a single \(\omega\)-rule having \(\alpha(\delta_i)\) as head and (the negation normal form of) \(\text{Prereq}(\delta_i)\) as body:

\[
\alpha(\delta_i) : - \\
\text{Prereq}(\delta_i).
\]

See below for the \(\omega\)-rule obtained in our example.

In all, this proceeding results in \(\delta\)-rules sharing the syntactical format given in (18), while the original constituents of the default rules are pushed into \(\omega\)-rules that are then treated in the classical way by means of standard PTTP-techniques.

In concrete terms, our system generates for the above default rule the following (intermediate) Prolog code:

\[
\begin{align*}
\text{not}\_d : - \\
\quad \text{not}\_e, \\
\quad \gamma(42). \\
\text{e} : - \\
\quad d, \\
\quad \gamma(42). \\
\gamma(42) : - \\
\quad \alpha(42), \\
\quad \text{justification([not\_d, e])}. \\
\alpha(42) : - \\
\quad a, \\
\quad (\text{not}\_b ; c).
\end{align*}
\]

From the perspective of default rule

\[
\begin{align*}
A \land (\neg B \lor C) : - & \ D \lor E \\
\neg D \lor E
\end{align*}
\]

the first two rules tell us intuitively that we can use its consequent \(\neg D \lor E\) if we can "apply" the default rule, indicated by \(\gamma\). That is, we can prove \(\neg D\) if we can prove \(\neg E\) and \(\gamma\); and we can prove \(E\) if we can prove \(D\) and \(\gamma\). Analogously, the last
rule accounts for proving the prerequisite. Only the third Prolog rule must subsequently receive special treatment as a δ-rule; all others are treated by standard P'TP-techniques.

The body of the δ-rule comprises the term structure justification([not_d, e]) containing the clausal representation [{¬D, E}] of the default rule's justification. As mentioned above, this term is later on turned into a subgoal using run-time predicate compatible/3. The actual implementation of compatible/3 does thus not rely on a membership test, as put forward in the simplistic setting in Section 5, but rather a satisfiability test taking a partial model along with a clause set as arguments. Recall that such a test is linear in the size of the clause set. We refer the interested reader to [59] (or even [83]) for further implementation details.

In view of these details, we can estimate the number of Prolog rules stemming from default rules in the non-atomic case: assuming that all prerequisites and consequents have at most \( m \) literals, we thus obtain for a default theory including \( d \) default rules, \( (m \times d) \) \( \alpha \)-rules (for each default, at most \( (m - 1) \) with \( \gamma_5 \) in the body and 1 with \( \alpha_5 \) as head) and \( d \) δ-rules (for each default, one δ-rule).

5.2.5. Variables over a finite universe

Finally, a word on our treatment of variables. As mentioned in the introductory section, our current implementation treats variables over a finite Herbrand universe in the rudimentary sense that a formula or a rule, respectively, is regarded as the representative of all its ground instances; thus, skolemization is not considered. Although this boils down to propositional logic, too, it allows for expressing things more concisely. This is implemented by a technique known from automated theorem proving and deductive databases that makes Prolog rules range-restricted by inserting unary predicates enumerating the Herbrand universe for unbound variables violating range-restrictedness [22]. In this way, during an inference, every unbound variable of an open goal is bound to a ground term (furnished by the unary predicate). This allows us to handle variables occurring in default rules by Prolog variables and so to avoid the generation of ground instances. 31 The only particular restriction we impose concerns default rules: variables in justifications must occur either in the corresponding prerequisite or consequent.

5.2.6. Intermediate Prolog code

We have already given an estimate on the number of resulting Prolog rules obtained in the basic setting. Let us now make this more precise in the light of the refinements and further details discussed above. Consider a default theory \((D, W)\) along with query \( \varphi \) over an alphabet \( \Sigma \) counting \( l \) propositional symbols. Let \( W \) contain \( n \) formulas in negation normal form and let \( D \) contain \( d \) default rules, all of whose constituents are in negation normal form. Let \( m \) be the maximal number of literals occurring in a formula of \( W \) or as constituent of a default rule. The resulting estimate is given in Fig. 11.

As an example, consider our initial default theory (4). With \( l = 6, n = 3, m = 3, d = 3 \), we get 60 Prolog clauses, as a worst case estimate. Without lemma handling, this reduces to 48 Prolog clauses. This has to be contrasted with the actual number of Prolog rules

31 A sort-oriented insertion of ground terms is currently implemented.
obtained, which is (without lemma handling) actually 34 only, as testified by the resulting Prolog code given in Fig. A.2.

5.3. Experimental results: a case study

This section gives a series of experimental results obtained with our implementation, the XRay system [77]. A more detailed report on experiments, test series generators, and further implementation details is given in [21,59] (or even [83]), so that we concentrate here on a case study illustrating the main features of the resulting system.

So, for being able to concentrate on the utility of the different features, we have decided to focus on a parameterizable series of test cases. This is provided by an encoding of the Hamiltonian cycle problem through default theories, as advocated in [24].\footnote{Actually, the encoding maps the Hamiltonian cycle problem onto a query answering problem in constrained default logic (see Section 6.1); this is arguably the variant of default logic closest to normal default theories, as discussed after Theorem 6.1.} This encoding is detailed in [59].

To begin with, we give in Table 1, taken from [59], a test series that provides experimental results on the impact of certain features on the runtime of our system. This is complemented by Table 2 that focuses on the compile time behavior and the number of inferences obtained by varying these features.

Table 1 is filled with items containing a time measure in seconds, comprising system and user time\footnote{The sum of both system and user time is necessary due to the underlying bi-processor.} (excluding compile time), along with the length of the resulting proof in parentheses. An entry like > 1000 means that no proof was obtained in 1000 seconds. The test series vary in two respects, leading to four different columns: the first two columns contain results obtained when checking first for the existence of a proof for the prerequisite

\[
((n + d) \times m) + d + (6 \times l) + 3
\]

\[W + D + \varphi\]
Table 1
Runtime experiments (on Linux Bi-PentiumPro, 200 MHz, 256 MB)

<table>
<thead>
<tr>
<th>lemma handling</th>
<th>(\alpha)-com</th>
<th>(\alpha)-com</th>
<th>(\ell)</th>
<th>(\ell)</th>
</tr>
</thead>
<tbody>
<tr>
<td>people</td>
<td>0.08 (64)</td>
<td>0.08 (50)</td>
<td>0.09 (64)</td>
<td>0.08 (50)</td>
</tr>
<tr>
<td>ham_4_min</td>
<td>0.01 (36)</td>
<td>0.0 (21)</td>
<td>0.01 (36)</td>
<td>0.0 (21)</td>
</tr>
<tr>
<td>ham_4_max</td>
<td>424.03 (36)</td>
<td>16.99 (21)</td>
<td>1.34 (36)</td>
<td>0.5 (21)</td>
</tr>
<tr>
<td>ham_5_min</td>
<td>&gt; 1000</td>
<td>&gt; 1000</td>
<td>0.12 (52)</td>
<td>0.06 (26)</td>
</tr>
<tr>
<td>ham_5_max</td>
<td>&gt; 1000</td>
<td>&gt; 1000</td>
<td>7.51 (52)</td>
<td>1.66 (26)</td>
</tr>
<tr>
<td>ham_6_min</td>
<td>&gt; 1000</td>
<td>&gt; 1000</td>
<td>0.37 (71)</td>
<td>0.15 (31)</td>
</tr>
<tr>
<td>ham_6_max</td>
<td>&gt; 1000</td>
<td>&gt; 1000</td>
<td>330.53 (71)</td>
<td>69.72 (31)</td>
</tr>
<tr>
<td>ham_7_min</td>
<td>&gt; 1000</td>
<td>&gt; 1000</td>
<td>1.03 (93)</td>
<td>0.25 (36)</td>
</tr>
<tr>
<td>ham_7_max</td>
<td>&gt; 1000</td>
<td>&gt; 1000</td>
<td>&gt; 1000</td>
<td>&gt; 1000</td>
</tr>
<tr>
<td>ham_8_min</td>
<td>&gt; 2000</td>
<td>&gt; 2000</td>
<td>11.83 (118)</td>
<td>1.76 (41)</td>
</tr>
<tr>
<td>ham_10_min</td>
<td>&gt; 2000</td>
<td>&gt; 2000</td>
<td>27.91 (177)</td>
<td>5.38 (51)</td>
</tr>
<tr>
<td>ham_20_min</td>
<td>&gt; 5000</td>
<td>&gt; 5000</td>
<td>&gt; 5000</td>
<td>309.4 (101)</td>
</tr>
<tr>
<td>ham_20_max</td>
<td>&gt; 5000</td>
<td>&gt; 5000</td>
<td>&gt; 5000</td>
<td>&gt; 5000</td>
</tr>
</tbody>
</table>

of a default rule and subsequently for its compatibility (during a \(\delta\)-extension step); this is indicated by \(\alpha\)-com. The order of tasks is switched in the columns headed by \(\alpha\)-com. This is done via the compiler option described at (15). Furthermore, the columns differ as concerns the usage of dynamic lemmas, indicated by \(\ell\). The configuration for dynamic lemmas was set to dynamic unit lemmas stemming from default consequents. All tests are done with blockwise regularity checks and without static lemmas.

For a contrast to our scalable test series, we place in front a more meaningful example, people, given by a taxonomic knowledge base comprising 62 formulas, including disjunctive integrity constraints, implications, and default rules, over 40 propositional symbols. The test vector, obtained by querying a ternary disjunction, is representative for our tests on natural and well-structured examples: the choices of the underlying theorem prover are only rarely corrected by subsequent default-specific checks. Hence, on such examples, efficient query answering is more or less obtained by means of the power of the underlying inference engine.

This changes when considering Hamiltonian cycle problems: we denote by \(\text{ham}_{n}\) the default theory corresponding to a graph with \(n\) vertices; it is build over 3 predicate symbols and \(n\) identifiers for the nodes. Let us note that \(\text{ham}_{n}\) contains only one classical formula and \(n^2 - 1\) defaults (see [59] for details on the encoding).\(^{34}\) For instance, this results for

\(^{34}\)To be precise, \(n \times (n - 1) + (n - 1)\) due to the special treatment of the starting node.
ham_15, comprising originally 224 default rules, in an intermediate Prolog code (provided by XRay) containing 904 rules and around 400 kbytes; its compilation takes 3.9 seconds, including printing, under Eclipse Prolog (cf. Table 2).

For the first test series, we have actually constructed for each problem ham_n ten different randomly selected permutations of the set of defaults. From these test series, we give for each column the minimal (ham_n_min) and maximal (ham_n_max) times obtained over these ten permutations.

This witnesses the important influence of “programming” the knowledge base, since the same problem, for instance ham_8, can be solved in a time varying from 2 seconds (in ham_8_min) to more than 2000 seconds (in ham_8_max) in the last configuration. We have pushed this a bit further by doing 200 permutations over the default set in ham_8. The minimum test vector \(^{35}\) obtained was 0.05 (118), 0.02 (41), 0.04 (118), 0.02 (41). This phenomenon is due to the fixed search strategy imposed by the underlying Prolog system. In order to diminish this influence, we may stick with PTTP’s iterative deepening proof search (although blockwise regularity guarantees a finite search space).

Let us now take a closer look at Table 1. First of all, we observe that the use of dynamic lemmas always reduces the proof length and even more importantly the time spent for finding this proof. The latter is testified by the fact that no matter which order of tasks is employed, we always improve (sometimes even by an order of magnitude) on the elapsed time. The impact of lemma handling on the proof (and in particular its length) is illustrated by Figs. 12 and 13, where the proof in ham_4 is given first without and then with the use of lemmas. In these figures, ext, red, unit, default and dyn-lemma indicate the application of an extension, reduction, unit, default, or dynamic lemma inference, respectively. \(^{36}\) In fact, we observe that the (repeated) subproofs of \(\gamma(4, \text{move}(1, 3), \text{vstd}(3))\), \(\gamma(7, \text{move}(3, 2), \text{vstd}(2))\) and \(\gamma(3, \text{move}(2, 4), \text{vstd}(4))\) in Fig. 12 are entirely replaced in Fig. 13 by using the corresponding default lemmas, since they have already been solved when proving ext(\(\text{vstd}(4))\). \(^{37}\)

Now, let us turn to Table 2 for observing the impact of the previous features on the compile time behavior and the total number of inferences. In contrast to the first series of tests, we refrain from permuting the Prolog rules and consider simply the theories directly issued by our generator. This has the advantage that all samples have an analogous structure. The first two columns of Table 2 indicate the considered sample and the configuration under which it was compiled. Symbol \(\ell\) stands for the previous lemma setting. Let us explain the table by looking at the first line of ham_3. The compile time of 0.07/0.11 tells us that it took XRay 7 milliseconds to compile the source into the target (including printing); it took 4 additional milliseconds to dump the Prolog file to disk and have it compiled by Eclipse Prolog. The size of source and target code is given by the number of default or Prolog rules, respectively, along with the number of resulting bytes. For instance, the source code for ham_3 consists of 8 default rules; the corresponding source file has 863 bytes (including further text). In the first setting, this is compiled into

---

\(^{35}\) Such a vector corresponds to a line in Table 1.

\(^{36}\) A unit inference is an extension inference with a unit clause.

\(^{37}\) Since XRay may use PTTP’s iterative deepening search, we can even assure shortest proofs.
ext (query)
| ext (vstd(4))
| | default (gamma(3, (move(2, 4), vstd(4))))
| | ext (alpha(3, vstd(2)))
| | | ext (vstd(2))
| | | default (gamma(7, (move(3, 2), vstd(2))))
| | | | ext (alpha(7, vstd(3)))
| | | | | ext (vstd(3))
| | | | | default (gamma(4, (move(1, 3), vstd(3))))
| | | | | | ext (alpha(4, vstd(1)))
| | | | | | | unit (vstd(1))
| | ext (vstd(3))
| | default (gamma(4, (move(1, 3), vstd(3))))
| | | ext (alpha(4, vstd(1)))
| | | | unit (vstd(1))
| | ext (vstd(2))
| | default (gamma(7, (move(3, 2), vstd(2))))
| | | ext (alpha(7, vstd(3)))
| | | | ext (vstd(3))
| | | | default (gamma(4, (move(1, 3), vstd(3))))
| | | | | ext (alpha(4, vstd(1)))
| | | | | | unit (vstd(1))
| | unit (vstd(1))
| ext (vstdtwice(1))
| default (gamma(15, (move(4, 1), vstdtwice(1))))
| | ext (alpha(15, vstd(4)))
| | | ext (vstd(4))
| | | default (gamma(3, (move(2, 4), vstd(4))))
| | | | ext (alpha(3, vstd(2)))
| | | | | ext (vstd(2))
| | | | | default (gamma(7, (move(3, 2), vstd(2))))
| | | | | | ext (alpha(7, vstd(3)))
| | | | | | | ext (vstd(3))
| | | | | | | default (gamma(4, (move(1, 3), vstd(3))))
| | | | | | | | ext (alpha(4, vstd(1)))
| | | | | | | | | unit (vstd(1))
| ext (vstd(3))
| default (gamma(4, (move(1, 3), vstd(3))))
| | ext (alpha(4, vstd(2)))
| | | ext (vstd(3))
| | | default (gamma(7, (move(3, 2), vstd(2))))
| | | | ext (alpha(7, vstd(3)))
| | | | | ext (vstd(3))
| | | | | default (gamma(4, (move(1, 3), vstd(3))))
| | | | | | ext (alpha(4, vstd(1)))
| | | | | | | | unit (vstd(1))

Fig. 12. Proof tree obtained from ham_4 without lemma handling and setting com-α.

40 Prolog rules, having 11246 bytes, along with one query rule, having 442 bytes. The slightly different target sizes, viz. 11246 and 11207, obtained by switching α-com and com-α are due to variable namings done by the Prolog compiler. In fact, our pretty printer maps both to identical files containing 8186 bytes (by replacing variables like _g9894 by single letters). A larger difference is observed when lemma handling is done; this is even reflected by the size of the query code, viz. 442 versus 512, due to two additional variables for tracing lemma proofs. The size of the query rule is also interesting because its body contains the initial model-clause-set. By and large, we observe on these examples (comprising many default rules and few classical ones) a factor of 1:4 between the number of default and Prolog rules. 38 In all, the efforts taken for the overall compilation are rather

38 The corresponding number of bytes is rather insignificant, since the resulting code is full of lengthy strings.
Fig. 13. Proof tree obtained from ham_4 with lemma handling and setting com-α.

small, despite the fact that the measured times include printing and that we employ a multiple pass compiling technique for easing implementation.\textsuperscript{39}

The last two columns reflect the resulting run-time behavior. The format of the last one is identical to the ones in Table 1 and should help to relate the samples with its two extreme mouldings given there. The last but one column provides the total number of inferences performed for finding the proof.\textsuperscript{40} Actually, the given figures substantiate the importance of our refinements for enhancing the basic approach: first, we see that a preceding compatibility check allows for finding the same proofs with a significantly smaller number of inferences. The usage of dynamic lemmas reduces this even further. This is impressively witnessed by ham_4, where both features lead to a speed-up by several orders of magnitude, namely, 80 as opposed to 323136 inferences. Notably, both features are obtainable at low compilation costs: while task switching is even free, lemma handling adds (under the given configuration) only a single Prolog rule (for predicate gamma) along with one further subgoal for each δ-rule.

A significant speed up on Hamiltonian cycle problems is achieved by verifying compatibility before doing the actual δ-extension steps. In our particular case, this is due to the rather complex justifications. So, when it comes to guaranteeing compatibility, the subsequent proof search becomes constrained by the justifications’ consistency assumptions; in this way, we may discard a large number of putatively applicable yet

\textsuperscript{39} In the current implementation, we actually go over the source code 17 times, if all features are enabled.

\textsuperscript{40} We refrained from doing so in the first series of tests, since counting inferences slows down the theorem prover considerably.
Table 2
Compile time experiments (on Solaris Ultra2, 512 MB)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Configuration</th>
<th>Compile time (secs/secs)</th>
<th>Source (rules:bytes)</th>
<th>Target (rules:bytes)</th>
<th>Inferences (rules:bytes)</th>
<th>Runtime (secs (infers))</th>
</tr>
</thead>
<tbody>
<tr>
<td>ham-3</td>
<td>α-com</td>
<td>0.07/0.11</td>
<td>8:863</td>
<td>40:11246</td>
<td>1:442</td>
<td>302</td>
</tr>
<tr>
<td>ham-3</td>
<td>com-α</td>
<td>0.06/0.09</td>
<td>8:863</td>
<td>40:11207</td>
<td>1:442</td>
<td>28</td>
</tr>
<tr>
<td>ham-3</td>
<td>α-com, ϵ</td>
<td>0.08/0.11</td>
<td>8:863</td>
<td>41:13495</td>
<td>1:512</td>
<td>83</td>
</tr>
<tr>
<td>ham-3</td>
<td>com-α, ϵ</td>
<td>0.09/0.11</td>
<td>8:863</td>
<td>41:13495</td>
<td>1:512</td>
<td>19</td>
</tr>
<tr>
<td>ham-4</td>
<td>α-com</td>
<td>0.12/0.16</td>
<td>15:1505</td>
<td>68:20999</td>
<td>1:516</td>
<td>323136</td>
</tr>
<tr>
<td>ham-4</td>
<td>com-α</td>
<td>0.12/0.15</td>
<td>15:1505</td>
<td>68:21079</td>
<td>1:516</td>
<td>162</td>
</tr>
<tr>
<td>ham-4</td>
<td>α-com, ϵ</td>
<td>0.14/0.17</td>
<td>15:1505</td>
<td>69:25208</td>
<td>1:600</td>
<td>8079</td>
</tr>
<tr>
<td>ham-4</td>
<td>com-α, ϵ</td>
<td>0.14/0.19</td>
<td>15:1505</td>
<td>69:25354</td>
<td>1:600</td>
<td>80</td>
</tr>
<tr>
<td>ham-5</td>
<td>com-α</td>
<td>0.21/0.27</td>
<td>24:2525</td>
<td>104:33730</td>
<td>1:590</td>
<td>369</td>
</tr>
<tr>
<td>ham-5</td>
<td>α-com, ϵ</td>
<td>0.24/0.33</td>
<td>24:2525</td>
<td>105:4131</td>
<td>1:706</td>
<td>14241619</td>
</tr>
<tr>
<td>ham-5</td>
<td>com-α, ϵ</td>
<td>0.24/0.31</td>
<td>24:2525</td>
<td>105:41307</td>
<td>1:706</td>
<td>122</td>
</tr>
<tr>
<td>ham-6</td>
<td>com-α</td>
<td>0.33/0.4</td>
<td>35:4035</td>
<td>148:51763</td>
<td>1:700</td>
<td>712</td>
</tr>
<tr>
<td>ham-6</td>
<td>com-α, ϵ</td>
<td>0.38/0.49</td>
<td>35:4035</td>
<td>149:61855</td>
<td>1:796</td>
<td>176</td>
</tr>
<tr>
<td>ham-7</td>
<td>com-α</td>
<td>0.48/0.6</td>
<td>48:6131</td>
<td>200:72796</td>
<td>1:778</td>
<td>1225</td>
</tr>
<tr>
<td>ham-7</td>
<td>com-α, ϵ</td>
<td>0.57/0.71</td>
<td>48:6131</td>
<td>201:87306</td>
<td>1:904</td>
<td>242</td>
</tr>
<tr>
<td>ham-8</td>
<td>com-α</td>
<td>0.69/0.84</td>
<td>63:8909</td>
<td>260:99091</td>
<td>1:875</td>
<td>1942</td>
</tr>
<tr>
<td>ham-8</td>
<td>com-α, ϵ</td>
<td>0.73/0.92</td>
<td>63:8909</td>
<td>261:120270</td>
<td>1:1035</td>
<td>320</td>
</tr>
<tr>
<td>ham-9</td>
<td>com-α</td>
<td>0.95/1.16</td>
<td>80:12465</td>
<td>328:128785</td>
<td>1:955</td>
<td>2897</td>
</tr>
<tr>
<td>ham-9</td>
<td>com-α, ϵ</td>
<td>1.05/1.3</td>
<td>80:12465</td>
<td>329:158569</td>
<td>1:1131</td>
<td>410</td>
</tr>
<tr>
<td>ham-15</td>
<td>com-α</td>
<td>3.93/4.52</td>
<td>224:58133</td>
<td>904:416574</td>
<td>1:1447</td>
<td>15529</td>
</tr>
<tr>
<td>ham-15</td>
<td>com-α, ϵ</td>
<td>4.45/6.256</td>
<td>224:58133</td>
<td>905:531435</td>
<td>1:1757</td>
<td>1202</td>
</tr>
<tr>
<td>ham-20</td>
<td>com-α</td>
<td>11.18/12.84</td>
<td>399:138978</td>
<td>1605:1081161</td>
<td>1:2388</td>
<td>2192</td>
</tr>
</tbody>
</table>

Incompatible defaults in the course of the rest of the proof search. This is a great advantage of an incremental approach to consistency checking. The inverse phenomenon (albeit with a largely different significance) is sometimes observed on our taxonomic knowledge base, where the choices of the inference engine are only rarely corrected by the subsequent compatibility check.

Another significant influence on resolving Hamiltonian cycle problems is given by using dynamic lemmas. In addition, dynamic lemmas practically never harm the proof search due to their restricted viability. This can be testified by a failing query $41$ to ham-4 yielding a test vector 3964.03 (−), 191.64 (−), 17.69 (−), 4.64 (−) (in the format of Table 1). We see that despite the usage of dynamic lemmas, failure is detected much faster than without them.

$41$ A failing query was obtained by adding fail to the end of the successful query.
Technically, this is due to the fact that (propositional unit) lemmas are treatable as unit clauses; hence the application of a lemma during a derivation never needs to be replaced by another derivation step (cf. Section 5.2.3).

We have discussed the salient features of XRay by means of a test series on the Hamiltonian cycle problem. This problem has already been used in similar settings, like the computation of extensions in classical default logic [25] or the computation of stable models of logic programs [60], as a basis for a parameterizable problem set. Actually, current work includes benchmark generators tailored to query answering in semi-monotonic default logics, since the existing ones, aiming at the two aforementioned problems, do not apply. A selection of our generators and the resulting test sets can be found at [83], among them, test sets on coloring problems and random graphs, all of them initially drawn from the Stanford GraphBase [46]. For a complement, we also give at [83] a couple of example files stemming from an application on model-based diagnosis. A discussion of these examples is however beyond the scope of this paper.

6. Treating general default theories

Up to now, we have concentrated on normal default theories as a somewhat greatest common fragment of default logics. For generalizing the approach, we may actually take over most of the techniques developed in the previous sections. This is because full-fledged default logics, like classical [71], justified [55], constrained [30], or rational default logic [58], differ only in the way they deal with the consistency check. Due to our modular treatment of consistency, we may thus concentrate on this issue, while keeping the techniques developed for deduction and groundedness.

For implementing the (semi-monotonic fragments) of the aforementioned variants, we have actually two alternatives: either we address each variant in turn, or we provide a technique general enough to cover all of them. The latter option is clearly the more general one. Apart from the fact that it allows for realizing the first option anyway, it has moreover the advantage that we may mix multiple conceptions of default logics in the same setting. This is why we have chosen to pursue the more general approach.

This undertaking benefits from the fact that its theoretical underpinnings have already been established in [12], where a context-based framework for default logics was proposed. In this approach each variant of default logic corresponds to a fragment of a more general and uniform default reasoning system, called contextual default logic.

6.1. The general setting: contextual default logic

As mentioned above, full-fledged default logics differ basically in the way they address consistency. While classical and justified default logic employ a rather local notion of consistency by separately verifying the consistency of each justification, constrained and rational default logic take a global approach by stipulating that all justifications have to be jointly consistent with an extension at hand.

Let us illustrate this briefly by taking a situation frequently encountered when inviting allergic children. Assume the kid stayed overnight, and we do not remember whether she
must not eat eggs or whether her diet denies milk. If we can consistently assume that she may eat eggs, we will serve omelette, and if we can consistently assume that she may have milk, we will serve porridge. This can be represented by the following default theory:

\[
\left\{ \begin{array}{l}
\text{eggs} \\
\text{porridge}
\end{array} \right\} \leadsto \text{omelette}, \left\{ \begin{array}{l}
\text{milk}
\end{array} \right\} \leadsto \text{porridge}, \neg \text{eggs} \lor \neg \text{milk}
\]

While classical and justified default logic yield a single extension containing omelette \& porridge, constrained and rational default logic provide two alternative extensions, one containing omelette and another one with porridge. Because all of these extensions contain moreover \( \neg \text{eggs} \lor \neg \text{milk} \), a global approach does therefore not permit to assume both eggs and milk, while this is the case when treating justifications separately. For brevity, we refrain from commenting these results any further and refer the reader to the literature for a detailed discussion on the technical and intuitive consequences arising from these different approaches to consistency [19,30,35,38,68,80]. In summary, this literature shows that there is not only a formal need for distinct notions of consistency in order to obtain different formal properties, but moreover a need stemming from knowledge engineering due to numerous commonsense examples that demand one or the other conception to be handled in the intuitively more appealing way. 42

So, in order to combine variants of default logic, one has to compromise different notions of consistency. [12] capture this by means of the notion of pointwise closure \( \text{Th}_S(T) \):

**Definition 6.1.** Let \( T \) and \( S \) be sets of formulas. If \( T \) is non-empty, the pointwise closure of \( T \) under \( S \) is defined as

\[
\text{Th}_S(T) = \bigcup_{\phi \in T} \text{Th}(S \cup \{\phi\}).
\]

In addition, \( \text{Th}_S(\emptyset) = \text{Th}(S) \).

Given two sets of formulas \( T \) and \( S \), we say that \( T \) is pointwisely closed under \( S \) iff \( T = \text{Th}_S(T) \).

For illustration, consider how we can express the “context” witnessing the derivation of omelette \& porridge from Theory (19) in classical default logic:

\[
\text{Th}(\neg \text{eggs} \lor \neg \text{milk}, \text{omelette}, \text{porridge}) (\{\text{eggs}, \text{milk}\}).
\]

This set comprises two consistent, deductively closed sets of formulas, one containing eggs and another one containing milk; taken together, these two sets are inconsistent and would thus yield any formula by applying deductive closure. Such contexts are made explicit in the framework provided by contextual default logic.

In this approach, one considers three sets of formulas: a set of facts \( W \), an extension \( E \), and a certain context \( C \) such that \( W \subseteq E \subseteq C \). The set of formulas \( C \) is somehow established from the facts, default consequents, and underlying consistency assumptions, given by the justifications of the applying default rules. For those familiar with the

---

42 Interestingly, Reiter already anticipated in [71, p. 83] that “providing an appropriate formal definition of this consistency requirement is perhaps the thorniest issue in defining a logic for default reasoning”.

aforementioned default logics, this approach trivially captures the application conditions found in existing default logics: for \( \frac{\alpha \land \beta}{\gamma} \), e.g., \( \alpha \in E \) and \( \neg \beta \notin F \) in the case of classical default logic, and \( \alpha \in E \) and \( \neg(\beta \land \gamma) \notin C \) in constrained default logic.

This variety of application conditions motivates an extended notion of a default rule [12]: a contextual default rule \( \delta \) is an expression of the form

\[
\frac{\alpha_W \mid \alpha_E \mid \alpha_C : \beta_C \mid \beta_E \mid \beta_W}{\gamma}
\]

where \( \alpha_W, \alpha_E, \alpha_C, \beta_C, \beta_E, \beta_W \) and \( \gamma \) are formulas. \( \alpha_W, \alpha_E, \alpha_C \) are called the \( W- \), \( E- \) and \( C- \) prerequisites, also noted \( \text{Prereq}_W(\delta), \text{Prereq}_E(\delta), \text{Prereq}_C(\delta) \), \( \beta_C, \beta_E, \beta_W \) are called the \( C- \), \( E- \) and \( W- \) justifications, also noted \( \text{Justif}_C(\delta), \text{Justif}_E(\delta), \text{Justif}_W(\delta) \) and \( \gamma \) is called the consequent, also noted \( \text{Conseq}(\delta) \). These projections extend to sets of contextual default rules in the obvious way (e.g., \( \text{Justif}_E(D) = \bigcup_{\delta \in D} \text{Justif}_E(\delta) \)). For convenience, we omit tautological components; a non-existing component must thus be identified with \( T \).

A contextual default theory is a pair \( (D, W) \), where \( D \) is a set of contextual default rules and \( W \) is a consistent set of formulas. \(^{43}\) A contextual extension is a pair \( (E, C) \), where \( E \) is a deductively closed set of formulas and \( C \) is a pointwisely closed set of formulas. We give below the quasi-iterative characterization of contextual extensions:

**Definition 6.2.** Let \( (D, W) \) be a contextual default theory and let \( E \) and \( C \) be sets of formulas. Define

\[
E_0 = \text{Th}(W), \quad C_0 = \text{Th}(W)
\]

and for \( i \geq 0 \)

\[
\Delta_i = \left\{ \frac{\alpha_W \mid \alpha_E \mid \alpha_C : \beta_C \mid \beta_E \mid \beta_W}{\gamma} \in D \mid W \vdash \alpha_W, \ E_i \vdash \alpha_E, \ C_i \vdash \alpha_C, \ C \nvdash \beta_C, \ E \nvdash \beta_E, \ W \nvdash \beta_W \right\},
\]

\[
E_{i+1} = \text{Th}(W \cup \text{Conseq}(\Delta_i)),
\]

\[
C_{i+1} = \text{Th}_{W \cup \text{Conseq}(\Delta_i) \cup \text{Justif}_C(\Delta_i)}(\text{Justif}_E(\Delta_i)).
\]

Then, \( (E, C) \) is a contextual extension of \( (D, W) \) if

\[
(E, C) = \left( \bigcup_{i=0}^{\infty} E_i, \bigcup_{i=0}^{\infty} C_i \right).
\]

The extension \( E \) is built by successively introducing the consequents of all applying contextual default rules. For each partial context \( C_{i+1} \), the partial extension \( E_{i+1} \) is united with the \( C- \) justifications of all applying contextual default rules. \(^{44}\) This set is united in turn with each \( E- \) justification of all applying contextual default rules.

\(^{43}\) Originally, Besnard and Schaub [12] deal with a deductively closed set \( W \) in accord with the closure of \( E \) and \( C \). By compactness, we must actually never consider deductively closed premises for automated theorem proving purposes.

\(^{44}\) To see this, observe that \( \text{Th}_{W \cup \text{Conseq}(\Delta_i) \cup \text{Justif}_C(\Delta_i)}(\text{Justif}_E(\Delta_i)) = \text{Th}_{E_{i+1} \cup \text{Justif}_C(\Delta_i)}(\text{Justif}_E(\Delta_i)) \).
In [12], it was shown that classical, justified, and constrained default logic are embedded in contextual default logic. Linke and Schaub [52] extend these embeddings to rational default logic. For brevity, we exemplarily give the resulting mappings and refer the reader to [12,52] for the corresponding equivalence results:

**Definition 6.3.** Let \((D, W)\) be a default theory. We define

\[
\Phi_{\text{DL}}(D, W) = \left\{ \frac{\alpha \vdash \beta}{\gamma} \mid \alpha, \beta \in D \right\}, W \quad \text{(Classical default logic)}
\]

\[
\Phi_{\text{JDL}}(D, W) = \left\{ \frac{\alpha \vdash \beta \land \gamma}{\gamma} \mid \alpha, \beta \in D \right\}, W \quad \text{(Justified default logic)}
\]

\[
\Phi_{\text{CDL}}(D, W) = \left\{ \frac{\alpha \vdash \beta \land \gamma}{\gamma} \mid \alpha, \beta \in D \right\}, W \quad \text{(Constrained default logic)}
\]

\[
\Phi_{\text{RDL}}(D, W) = \left\{ \frac{\alpha \vdash \beta}{\gamma} \mid \alpha, \beta \in D \right\}, W \quad \text{(Rational default logic)}
\]

These embeddings extend to variants of default logic relying on labeled formulas (or assertions [19]): Delgrande et al. [30] show that constrained and cumulative default logic [19] are equivalent modulo representation. The same is shown by Giordano and Martinelli [40] for classical and Q-default logic [40], and by Linke and Schaub [52] for rational and CA-default logic [40], respectively.

For example, we obtain for theory \((D, W)\), as given in (19), the following theories:

\[
\Phi_{\text{JDL}}(D, W) = \left\{ \frac{\alpha \vdash \beta, \gamma}{\gamma} \mid \alpha, \beta \in D \right\}, (21)
\]

\[
\Phi_{\text{CDL}}(D, W) = \left\{ \frac{\alpha \vdash \beta, \gamma}{\gamma} \mid \alpha, \beta \in D \right\}, (22)
\]

It is instructive to verify that \(\Phi_{\text{JDL}}(D, W)\) has a single contextual extension containing omelette \& porridge, while \(\Phi_{\text{CDL}}(D, W)\) yields two distinct extensions, one containing omelette and another with porridge. This is due to the different consistency requirements used in each theory. While the C-justification eggs \& omelette of

\[
\frac{\alpha \vdash \beta, \gamma}{\gamma}
\]

must not only be consistent with the extension at hand but moreover with all justifications of other applying default rules, the E-justification eggs \& omelette of

\[
\frac{\alpha \vdash \beta, \gamma}{\gamma}
\]

requires consistency with the extension only. This is why both default rules of \(\Phi_{\text{JDL}}(D, W)\) contribute to its single extension, whereas each rule of \(\Phi_{\text{CDL}}(D, W)\) engenders a distinct extension.
In order to capture the family of default logics described in Definition 6.3, we may restrict ourselves to contextual default rules of the following form:

$$D^* = \left\{ \frac{\alpha | : \beta_C | \beta_E | \gamma}{\alpha, \beta_C, \beta_E, \gamma \in \mathcal{L}_E} \right\}. \quad (23)$$

As motivated in the introductory sections, our approach relies on the ability of forming default proofs in a local fashion. This is why we have applied our approach up to now to normal default theories only, since these theories enjoy the property of semi-monotonicity. Analogously, our generalization relies on locally determinable default proofs: we call a contextual default theory $$(D, W)$$ semi-monotonic if we have for any two subsets $D'$ and $D''$ of $D$ with $D'' \subseteq D' \subseteq D$ that if $$(E'', C'')$$ is a contextual extension of $$(D'', W)$$, then there is a contextual extension $$(E', C')$$ of $$(D', W)$$ such that $E'' \subseteq E'$ and $C'' \subseteq C'$. For example, theories (21) and (22) are semi-monotonic.

Actually, justified and constrained default logic enjoy semi-monotonicity in full generality. Clearly, this carries over to the corresponding fragments of contextual default logic, so that our approach extends immediately to these variants of default logic. Notably, this extends to the union of the respective default theories; thus allowing for treating some default rules according to justified default logic and others according to constrained default logic. For classical and rational default logic, on the other hand, we must restrict ourselves to semi-monotonic fragments. (Such fragments should be determinable by appropriate stratification techniques; a concrete adaptation of such techniques remains however future work.)

For furnishing an appropriate proof theory, we provide next a more proof-theoretic characterization of contextual extensions in the presence of semi-monotonicity:

**Theorem 6.1.** Let $$(D, W)$$ be a semi-monotonic contextual default theory such that $D \subseteq D^*$ and let $E$ and $C$ be sets of formulas. Then, $$(E, C)$$ is a contextual extension of $$(D, W)$$ iff there is some maximal $D' \subseteq D$ that has an enumeration $$(\delta_i)_{i \in I}$$ such that for $i \in I$, we have:

$$\begin{align*}
E &= \text{Th}(W \cup \text{Conseq}(D')) \\
C &= \text{Th}_{E \cup \text{Justif}_C(D')} (\text{Justif}_E(D'))
\end{align*} \quad (24)$$

$$W \cup \text{Conseq}([\delta_0, \ldots, \delta_{i-1}]) \vdash \text{Prereq}_E(\delta_i), \quad (25)$$

$$\begin{align*}
W \cup \text{Conseq}([\delta_0, \ldots, \delta_i]) \cup \text{Justif}_C([\delta_0, \ldots, \delta_i]) &\cup \text{NotJustif}_E(\delta_k) \\
&\text{for } k \in \{0, \ldots, i\}
\end{align*} \quad (26)$$

It is instructive to verify that this specification reduces to that given in Theorem 2.1 when dealing with normal default theories (no matter which translation is used).

As another example, consider the instantiation of this definition for constrained default logic: while condition (25) as well as the specification of $E$ in (24) remain the same, the definition of $C$ reduces to $C = \text{Th}(E \cup \text{Justif}_C(D'))$, thus dealing with deductively closed
sets. Another simplification is observed when regarding condition (26) due to the absence of $E$-justifications: 45

$$W \cup \text{Conseq}([\delta_0, \ldots, \delta_{i-1}]) \cup \text{Justif}_C([\delta_0, \ldots, \delta_{i-1}]) \not\vdash -\text{Conseq}(\delta_i) \lor -\text{Justif}_C(\delta_i).$$

This consistency condition is actually closely related to that found in normal default theories (cf. Definition 2.1). While in normal default theories justifications coincide with consequents, they must be distinguished in a full-fledged default logic. In fact, in constrained default logic, this distinctive treatment is done in such a way that we can take over the whole machinery developed for consistency checking in normal default theories by simply replacing each manipulation of a consequent $\gamma$ by that of its conjunction with the corresponding justification $\beta \land \gamma$. As a result, we can address compatibility checking in constrained default logic by maintaining a single model along with a single model-clause-set.

In analogy to Definition 2.2, a default proof for a formula $\varphi$ from a contextual default theory $(D, W)$ is then a finite sequence of contextual default rules $(\delta_i)_{i \in I}$ with $\delta_i \in D$ for all $i \in I$ such that $W \cup \{\text{Conseq}(\delta_i) \mid i \in I\} \vdash \varphi$ and conditions (25) and (26) are satisfied for all $i \in I$. Clearly, the first two conditions, that is, the derivation of $\varphi$ from $W$ and $(\delta_i)_{i \in I}$ and also groundedness of $(\delta_i)_{i \in I}$, are treated as developed in the previous sections. So that we can concentrate on the implementation of condition (26). This is accomplished in the next subsection.

6.2. Generalizing model-based consistency checking

In the sequel, we extend the model-based approach introduced in Section 4 to the general setting described above. For normal default theories, it was sufficient to furnish a single model of the premises in $W$ satisfying all default conclusions in a proof at hand. 46 In the presence of putatively contradictory $E$-justifications, however, we need more complex model structures for guaranteeing compatibility. In fact, we might now need several models of $W$, all of which must entail the consequents and the $C$-justifications of the default rules in the current derivation, while there must be at least one model of each $E$-justification among them. Observe that the models covering $E$-justifications are not necessarily distinct; distinctness is only necessary in the presence of contradictory $E$-justifications.

Let us make this precise in the sequel. For a formula $\phi$ and a set of models $M$, we write $M \models \phi$ if $m \models \phi$ for all $m \in M$; and $M \not\models \phi$ if $m \models \neg \phi$ for some $m \in M$. For a set of formulas $S$, we define its set of models as $\text{Mod}(S)$. First, we account for the semantic counterpart of the notion of pointwise closure (cf. Definition 6.1): for sets of formulas $S$ and $T$, we define

$$\text{Mod}_S(T) = \begin{cases} \bigcup_{\phi \in T} \text{Mod}(S \cup \{\phi\}) & \text{if } T \neq \emptyset, \\ \text{Mod}(S) & \text{otherwise.} \end{cases}$$

45 To be precise, all $E$-justifications are tautological rather than non-existent.

46 This extends actually to default logics employing $C$-justifications only, such as constrained default logic.
For a set of formulas $W$ and a sequence of contextual default rules $(\delta_i)_{i \in I}$ of form (23), we are then interested in the set of models $Mod_5(T)$ obtained by taking

$$S = W \cup \text{Conseq}(\{\delta_i | i \in I\}) \cup \text{Justif}_E(\{\delta_i | i \in I\}) \quad \text{and} \quad T = \text{Justif}_E(\{\delta_i | i \in I\}).$$

For readability, we abbreviate this set of models by $M_W(I)$; in analogy, we denote its subset

$$\{m \in M_W(I) | m \models \text{Justif}_E(\delta_i)\}$$

by $M^i_W(I)$ for $i \in I$. In fact, for non-empty $I$, $M_W(I)$ equals $\bigcup_{i \in I} M^i_W(I)$, each of which covers a different $E$-justification $\text{Justif}_E(\delta_i)$ in $\text{Justif}_E(\{\delta_i | i \in I\})$.

Consider the semantic counterpart of the pointwisely closed set given in (20):

$$\text{Mod}_v(\{\text{eggs}, \neg \text{milk}, \text{omelette}, \text{porridge}\}(\{\text{eggs}, \text{milk}\})).$$

This set of models is actually composed of two distinct sets: the model set satisfying eggs $\land \neg$ milk and the one satisfying milk $\land \neg$ eggs; they actually comply with $M^i_W((1, 2))$ and $M^i_W((1, 2))$, where $(1, 2)$ is the index set corresponding to the default rules obtained by applying $\Phi_{DL}$ to default theory (19). Such sets of models furnish the domain from which we select individual models witnessing the compatible application of default rules.

Now, in order to characterize compatible default proofs $(\delta_i)_{i \in I}$ from a set of premises $W$, we consider non-empty subsets $M$ of $M_W(I)$ such that $M \cap M^i_W(I) \neq \emptyset$ for all $i \in I$; and we use $\subseteq_I$ to indicate by writing $M \subseteq_I M_W(I)$ that this structural set inclusion property holds. Observe that for non-empty $I$ the existence of such a set $M$ implies that all underlying sets $M^i_W(I)$ are non-empty. This guarantees that $M$ contains at least one model for each $E$-justification $\text{Justif}_E(\delta_i)$. In case $I$ is empty, we also deal with a non-empty subset $M$ of $M_W(\emptyset) = \text{Mod}(W)$; we write $M \subseteq_\emptyset M_W(\emptyset)$. The non-emptiness of $M$ is guaranteed, since $W$ is assumed to be consistent.

For a contextual default rule

$$\delta_i = \frac{|\alpha| : \beta_C | \beta_E}{\gamma}$$

and some index set $I = K \cup \{i\}$, function $\nabla$ addresses condition (26) in Theorem 6.1 by mapping triples of form $(M, W, (\delta_k)_{k \in K})$ with $M \subseteq_K M_W(K)$ onto triples of the same format if condition (26) is true; it yields $\bot$ if condition (26) is false:

$$\nabla(\delta_i, (M, W, (\delta_k)_{k \in K})) = \begin{cases} 
(M, W, (\delta_i)_{i \in I}) & \text{if } M \models \gamma \land \beta_C \text{ and } m \models \beta_E \text{ for some } m \in M, \\
(M', W, (\delta_i)_{i \in I}) & \text{if } M \not\models \gamma \land \beta_C \text{ or } m \not\models \beta_E \text{ for all } m \in M \\
&M' \subseteq_K M_W(K), \\
&M' \models \gamma \land \beta_C \text{ and } m' \models \beta_E \text{ for some } m' \in M', \\
&\bot & \text{if there is no } M'' \subseteq_K M_W(K), \\
&M'' \models \gamma \land \beta_C \text{ and } m'' \models \beta_E \text{ for some } m'' \in M''. 
\end{cases}$$
Observe that $M' \subseteq_K M_W(K)$ implies $M' \neq \emptyset$ even though $K = \emptyset$ due to the consistency of $W$. As anticipated in the discussion of Theorem 6.1, we may restrict our attention to singleton sets $M'$ in the absence of $E$-justifications; this amounts more or less to the approach presented in Section 4 (except for the integration of justifications differing from the associated consequents). $M'$ must contain multiple models when dealing with inconsistent $E$-justifications. In the worst case, that is when dealing with $n$ pairwisely inconsistent $E$-justifications, $M'$ includes at most $n$ distinct models.

The following result shows that this approach is in accord with the conception of consistency expressed in Definition 6.1.

**Theorem 6.2.** Let $W$ be a set of formulas and $(\delta_i)_{i \in I}$ a sequence of contextual default rules such that $\delta_i \in D^*$ for all $i \in I$. Then, we have for all $i \in I$ and $K = \{0, \ldots, i - 1\}$ and $L = \{0, \ldots, i\}$ that if there is a set of models $M \subseteq_K M_W(K)$, then there is either a non-empty set of models $M' \in M_W(L)$ such that

\[
\forall (\delta_i, (M, W, (\delta_k)_{k \in K})) = (M', W, (\delta_i)_{i \in I}) \text{ iff condition (26) is true}
\]

or

\[
\forall (\delta_i, (M, W, (\delta_k)_{k \in K})) = \bot \text{ iff condition (26) is false}.
\]

Observe that $M$ and $M'$ need not be distinct; thus covering the first two cases of $\forall$.

As argued in Section 4, we must provide efficient means for supporting model searching. For this purpose, we use extended model-clause sets of form

\[
(M, [M_i]_{i \in I}),
\]

where $M$ is a (compact) clausal representation of

\[
W \cup \text{Conseq}([\delta_i | i \in I]) \cup \text{Justif}_C([\delta_i | i \in I])
\]

and the $M_i$ are (compact) clause sets representing $\text{Justif}_E(\delta_i)$, respectively, for some default proof fragment $(\delta_i)_{i \in I}$ from $(D, W)$. Similar to Section 4, we start with an extended model-clause-set of form $(C_W, \emptyset)$ and a singular set of models $M = \{m\}$ for some $m \models W$. Also in analogy to Section 4, coexisting model sets and extended model-clause-sets are invariably coupled via satisfiability. That is, for a model set $M$ and an extended model-clause-set $(M, [M_i]_{i \in I})$, we have

\[
\forall i \in I. \exists m \in M. m \models M \cup M_i.
\]

(This is trivially true for tautological $E$-justifications yielding $M_i = \emptyset$.) It is important to observe that a model like $m$ may cover multiple clause sets of form $M_i$ whenever they are jointly consistent with $M$. That is, the number of involved $E$-justifications, $|I|$, is only an upper bound for the number of models in $M$.

The important reduction of the search space for models was obtained in Section 4 by appeal to continued model-preserving reductions of the model-clause-sets along with information supported by the DME-derivations by means of lemmas. Actually, all these techniques apply also in the general case, although we must pay some more attention to their scope of applicability. This is due to the fact that we share $M$ with all clause sets of form $M_i$. In fact, a separate treatment of all instances of form $M \cup M_i$ would allow for applying all techniques developed in Section 4 in a straightforward way, yet at the cost
\[ \langle C_W, \emptyset \rangle \]
where
\[ C_W = \{ \{ \neg EG, \neg MI \} \} \]

\[ M_0 = \{ \{ \neg MI \} \} \]

\[ \langle M', \{ M'_1 \} \rangle \]
where
\[ M' = \{ \{ \neg EG, \neg MI \}, \{ OM_\delta \} \} \]
\[ M'_1 = \{ \{ EG_\delta \}, \{ OM_\delta \} \} \rightsquigarrow \{ \{ EG_\delta \} \} \]

\[ M_1 = \{ \{ \neg MI, OM_\delta, EG_\delta \} \} \]

\[ \langle M'', \{ M''_1, M''_2 \} \rangle \]
where
\[ M'' = \{ \{ \neg EG, \neg MI \}, \{ OM_\delta \}, \{ PO_\delta \} \} \]
\[ M''_1 = \{ \{ EG_\delta \} \} \]
\[ M''_2 = \{ \{ MI_\delta \}, \{ PO_\delta \} \} \rightsquigarrow \{ \{ MI_\delta \} \} \]

\[ M_2 = \left\{ \begin{array}{l}
\{ \neg MI, OM_\delta, PO_\delta, EG_\delta \} \\
\{ \neg EG, OM_\delta, PO_\delta, MI_\delta \}
\end{array} \right\} \]

Fig. 14. Governing compatibility while deriving OM \land PO from \( \Phi_{\text{JDL}}(D, W) \).

of more redundancy. As a consequence, we allow for applying freely unit-reductions and subsumptions on \( M \), while modifications resulting from the same reductions on \( M \cup M_1 \) are restricted to \( M_1 \). We restrict lemma usage to those depending on default rules involved in the current DME-derivation only. This is reasonable and actually highly efficient since these are trivially compatible. They are addable to \( M \) which allows for reductions in the entire extended model-clause-set, including clause sets like \( M_1 \).

Let us illustrate this by verifying compatibility of default proof

\[
\left( \begin{array}{c|c|c|c}
\mid & \text{omelette} & \text{eggs} \land \text{omelette} & \\hline
\text{omelette} & \mid & \text{porridge} & \text{milk} \land \text{porridge} \\hline
\text{porridge} & \end{array} \right)
\]

(\( \delta_1, \delta_2 \) for short) for omelette \land porridge from \( \Phi_{\text{JDL}}(D, W) \). We start by putting an arbitrary model of \( C_W \) into our model set, \( M_0 \). The extended model-clause-set \( \langle C_W, \emptyset \rangle \) is given in the first line of Fig. 14. Applying one of the above rules after the other, yields:

\[ \nabla(\delta_1, \langle M_0, W, () \rangle) = M_1 \quad \text{and} \quad \nabla(\delta_2, \langle M_1, W, \langle \delta_1 \rangle \rangle) = M_2, \]

where model sets \( M_0, M_1 \) and \( M_2 \) are given in Fig. 14 (while abbreviating milk, eggs, omelette and porridge by MI, EG, OM and PO, respectively). \( M_1 \) is obtained from \( M_0 \) simply by extending the only model; this model ensures the compatible application of \( \delta_1 \). \( M_2 \) necessitates the generation of another model satisfying the \( E \)-justification of \( \delta_2 \).

Let us take a closer look at the underlying extended model-clause-sets. Applying \( \delta_1 \) makes us add its \( C \)-justification \( \text{omelette}_\delta \) to \( C_W \), resulting in \( M' \), whereas its \( E \)-justification \( \text{eggs}_\delta \land \text{omelette}_\delta \) engenders the creation of \( M'_1 \). Observe that \( M' \) itself is not reducible. However, we may reduce \( M' \cup M'_1 \) by subsumption. But even though there are two alternatives for deletion, such a reduction must only affect clauses in \( M'_1 \). This is why \( \{ OM_\delta \} \) is deleted in \( M'_1 \) and not in \( M' \) (this and all following reductions are indicated by the symbol \( \rightsquigarrow \)). The same type of reduction is applied in the following step. Actually,
when applying $\delta_2$, we are forced to generate an alternative model since the $E$-justifications of $\delta_1$ and $\delta_2$ are contradictory. This does however not prevent their joint application, as explained above. Finally, model set $M_2$ contains a model for $M'' \cup M'_1$ and another for $M'' \cup M'_2$.

For a complement, let us see why $(\{\text{eggs} \land \text{omelette}\}, \{\text{milk} \land \text{porridge}\})$ is no compatible default proof for $\text{omelette} \land \text{porridge}$ from $\Phi_{\text{CDL}}(D, W)$. The proceeding is illustrated in Fig. 15. First of all, we observe that this example does not comprise any (non-tautological) $E$-justifications; hence there are no secondary clause sets in the extended model-clause-set, which allows us to restrict our attention to singleton model sets only. Applying $\delta_2$ makes us add its $C$-justification $\text{eggs}_{\delta_1'} \land \text{omelette}_{\delta_1'}$ to $C_W$, resulting in $M'$, from which we obtain a singular model set $M_1 = \{\{-\text{MI}, \text{OM}_{\delta_1'}, \text{EG}_{\delta_1'}\}\}$. $M'$ is then reduced by unit-reduction. Next, our underlying DME-derivation makes us check compatibility for $\delta_1'$. The model in $M_1$ does not satisfy the $C$-justification of $\delta_1'$, that is,

$$\{\{-\text{MI}, \text{OM}_{\delta_1'}, \text{EG}_{\delta_1'}\}\} \not\models \text{MI}_{\delta_2'} \land \text{PO}_{\delta_2'}.$$

Thus, we look for a new model testifying joint compatibility of $\delta_1'$ and $\delta_2'$ (see above for an explanation). For this purpose, we extend the last model-clause-set by $\delta_2'$'s $C$-justification, resulting in $M''$. Applying immediate reductions yields a clause set with an empty clause, which indicates inconsistency. In this case, we were thus able to detect inconsistency without performing an actual consistency check. In terms of function $\nabla$, we have

$$\nabla(\delta_1', \langle M_0, W, \emptyset \rangle) = M_1 \quad \text{and} \quad \nabla(\delta_2', \langle M_1, W, \langle \delta_1' \rangle \rangle) = \bot$$

where $M_0$ and $M_1$ are given in Fig. 15.
For implementing the generalized approach to model-based consistency checking, we follow by and large the approach described in Section 5. This necessitates extending the datastructure encapsulated by functor m/2; also, we had to conceive additional means for treating and notably reusing as many models as possible at each compatibility check. All this is detailed in [21] (see also [83]).

6.3. Experimental results

The approach has been implemented for enhancing the XRay system [77], whose overall performance has already been discussed in Section 5.3. Hence, we concentrate in what follows on the implementation of our model-based approach to consistency checking. For this purpose, we have developed a tool (see [59]) to build generic contextual default theories in order to be able to parameterize the number of model generations during query answering. Since we deal with different sorts of justifications, it is moreover interesting to study the influence of different compatibility tests.

For abstracting from the underlying inferences, all generated default theories have a fixed inferential structure. As a result, all default rules are always applied in the same order for proving the query; it is just the fact whether a call to predicate compatible results in reusing or regenerating a model that varies in the test cases. This gives us a constant effort for inferencing no matter how many model switches are provoked. The different number of model generations is achieved by reordering the literals in the set of premises (and thus in the extended model-clause-sets) in view of the fixed application order of δ-rules and the known search strategy of the model generator. As an additional constraint, we imposed that all E-justifications are pairwisely inconsistent so that different E-justifications give rise to different models.

For brevity, we present here two exemplary test series, taken from [59], and refer the reader for more details to [21] (or even [83]): one series with 50 and another with 100 default rules, each of which contains additionally 50 and 100 binary clauses, respectively, provoking a fixed number of model switches. This gives a putative search space for models of $2^{50}$ and $2^{100}$. The number of effectuated model generations is indicated in the first column of the subsequent tables. We distinguish three major test cases comprising default rules having (i) only C-justifications, (ii) only E-justifications, (iii) both C- and E-justifications. These cases are listed in order in the columns headed by DC, BE and DC + BE in Table 3. The column headed by DC + BE distinguishes furthermore between test cases where model generations are originated by C-justifications (left column) and those where model generations are caused by E-justifications (right column). Each item contains a time measure\(^{47}\) in seconds.

Note that the two limiting cases are given in the first and last line, representing a single model generation and consecutive ones. Also, we see that model switches are generally more expensive in the presence of both types of justifications. This is reflected by the fact that we encounter higher figures in the two columns put together under $\beta_C + \beta_E$ than in the corresponding ones headed by $\beta_C$ and $\beta_E$, respectively.

\(^{47}\) As above, comprising system and user time.
We observe that model switches caused by $E$-justifications are more expensive than those caused by $C$-justifications. This is related to the fact that the latter impose a stronger consistency constraint than the former. That is, the consistency of a $C$-justification concerns all current models, while an $E$-justification needs a single model only. The failure of a model for warranting consistency is thus immediate if it falsifies a $C$-justification, while this involves some subsequent search for $E$-justifications: as witnessed by the second case in the specification of $V$, we verify that $m \not\models \beta_E$ for all models $m$ before the ultimate failure of consistency of $\beta_E$ can be confirmed. (Since our aim is to provoke model generations rather than recycling them, such a search is never successful on these particular test cases.)

Note also that the sole use of $C$-justifications makes us treat in turn a single yet different model, while in our setting the exclusive usage of $E$-justifications provokes that each proof including $n$ default rules involves handling $n$ distinct models (due to the pairwise inconsistency of $E$-justifications). Hence, except for the case of a single model generation, the sole usage of $C$-justifications is better than using $E$-justifications only, because an individual model can clearly be maintained in a much easier way than $n$ different models. The aforementioned exception is due to the fact that $C$-justifications are added to the primary clause set which is subject to consecutive reductions. These efforts however are rapidly amortized with an increasing number of model switches. Finally, we mention that we observed on non-artificial examples like those discussed in Section 5.3 very few model switches which indicates the putative feasibility of our approach in practice. A more detailed experimental analysis along with implementation details are given in [21] (see also [83]).

7. Discussion and concluding remarks

We showed how Prolog technology can be used for implementing efficient default reasoning systems. This was accomplished by appeal to the approach taken by Stickel’s PTTP. We described how a default theory $(D, W)$ along with a query $\varphi$ has to be transformed into a Prolog program $P_{D, W, \varphi}$ along with a Prolog query $\text{query}$ such that $\text{query}$ is derivable from $P_{D, W, \varphi}$ iff $\varphi$ has a finite default proof from $(D, W)$. A particularity of our approach stems from centering it around local proof procedures that allow for verifying the validity of each inference step when it is performed. For simplicity, we have restricted the first part of our exposition to normal default theories, as the arguably simplest fragment of default logic supporting local proof procedures, although
the approach applies to any semi-monotonic default logic, as detailed in Section 6. Note that the treatment of such full-fledged default logics is identical to that of normal default theories, except for the compatibility check. In this way, our approach can be regarded as a general methodology for implementing semi-monotonic default logics, as a prime candidate for default logics supporting local proof procedures.

From another perspective, we have shown how existing (high-performance) PTTP-based theorem provers can be enhanced by means for handling default information. For providing theoretical underpinnings for the underlying compilation techniques, we have proposed a top-down proof procedure based on model-elimination. We called the resulting method Default Model Elimination. This proof procedure has its roots in an approach to default query answering based on the connection method. This consequent integration of classical automated theorem proving techniques and novel default logic technology is a salient feature of the approach, it is in particular responsible for the overall performance of the resulting system. As a byproduct, we have put forward appropriate enhancements of well-known concepts in automated theorem proving, like blockwise regularity and diverse forms of lemma handling, for further improving the performance of our inference engine.

And finally, one can view our contribution also as a (propositional) logic programming system integrating disjunction, classical as well as default negation. That is, via the standard interpretation of logic program clauses (cf. [56]), such as

\[ p \leftarrow q_1, \ldots, q_m, \lnot(r_1), \ldots, \lnot(r_n), \]

where \( p, q_i, r_i \) are atomic formulas, through default rules of form

\[
\frac{q_1 \land \cdots \land q_m : \lnot r_1, \ldots, \lnot r_n}{p},
\]

we may actually experiment with the resulting default rules under the different interpretations, given in Section 6 (here for \( n = 1 \), a generalization to multiple justifications is however straightforward). This allows furthermore for investigating extensions, where \( p, q_i, r_i \) are non-atomic formulas, such as disjunctive logic programming.

Our approach is thus quite different from other approaches found in the literature:

First, except for \([2,17,72,81]\), all other approaches, like \([5,25,45,63,71]\), abstract from an underlying inference engine. We integrate both groundedness and (incremental) consistency into an existing standard theorem prover using a standard sub-prover for generating models. Poole proposes in \([69]\) an approach dealing with the fragment of prerequisite-free normal default rules that is also based on PTTP. In addition to the restriction to normal default rules, however, the restriction to prerequisite-free default rules renders an implementation of groundedness unnecessary, so that PTTP serves mainly as an underlying theorem prover (also used for consistency checking via failing derivations). On the other hand, the resulting Theorist framework is rather advanced as regards variable handling, whose treatment is arguably rooted in \([66]\).

Second, the methods described in \([25,45,50,63,81]\) aim primarily at computing entire extensions in Reiter's default logic; queries are then answerable from such an extension. This is somewhat unavoidable in Reiter's default logic, since it lacks semi-monotonicity and it does thus not allow for local proof procedures. This is why we concentrate on query answering from semi-monotonic default logics, which is actually much more apt to the standard query answering process of PTTP.
Third, consistency checking is treated differently: for instance, Reiter [71] puts forward a belated consistency check; Schwind and Risch [81] computes first entire sets of "consistent default rules" and verifies groundedness separately.

Moreover, our approach differs from all of the aforementioned ones in pursuing a model-based treatment of consistency checking. This novel approach aims at minimizing computational efforts by reusing models as compact representations of former consistency checks. We have demonstrated, on the one hand, that the crucial task of consistency checking can benefit from keeping models to restrict computational efforts to ultimately necessary exhaustive consistency checks. In our framework, such an exhaustive (exponential) check amounts to the generation of a new model: otherwise, we rather aim at reusing existing models for performing a fast (linear) satisfiability test. In Section 4, we have seen that even though three different defaults were used, a new model had to be generated only once. The consistent application of the two other defaults was warranted by two satisfiability tests. On the other hand, we have demonstrated that the use of model-clause-sets together with a rigorous application of reductions, results in a significant reduction of the underlying search space. Furthermore, we have shown that the concept of default lemmas allows for additional avoidance of redundancy in model-clause-sets. In this way, we achieved a synergistic and potentially parallel treatment of theorem proving and satisfiability checking.

Our handling of lemmas is related to that of so-called nogoods in assumption-based truth maintenance systems [29]. For instance, Sattar and Goebel [74] use such nogoods for amending Theorist for recognizing inconsistent hypothesis. In our setting, a nogood is a kind of lemma consisting of literals that correspond to the justifications of defaults, whose inconsistency has been detected by a previous compatibility check. Whenever a DME-derivation contains each element of such a nogood as δ-extension-resulting literal, this derivation can be rejected prior to checking compatibility. The integration of such a technique in our implementation remains an interesting piece of future work.

The extended approach of Section 6 has provided us with a general framework for implementing query answering in default logics supporting local proof procedures. This was accomplished by encompassing the variety of consistency checks found in existing default logics. Actually, apart from capturing diverse full-fledged default logics, this approach allows us moreover to combine these variants in an arbitrary fashion. The resulting system is thus unique in offering simultaneously the expressiveness of multiple default logics.

For finding new models, our implementation uses an adapted variant of the Davis–Putnam procedure [28], which is currently one of the fastest complete methods for finding propositional models. Our current experiments on "meaningful" (to be more precise, rules expressing taxonomies) and artificial examples (like graph problems, random problems, and other parameterizable examples) show that a model has to be changed quite rarely in the course of the proof search. That is, on many examples, we observed that the resulting default proofs contained only few occasions for distracting the theorem prover by choosing incompatible δ-rules. For instance, for deciding the Hamiltonian cycle problem, we observed on many examples very few model switches. This is an argument in favor of a compatibility check that reuses information gathered on previous compatibility checks.

Actually, current work includes benchmark generators tailored to query answering in semi-monotonic default logics, since the existing ones, aiming at the computation of entire
extensions in Reiter's default logic, do not apply. On the whole, our experiments have shown that the pursued avenue is quite promising. On different families of examples, our current implementation outperforms previous ones that do not rely on PTTP by an order of magnitude. In fact, credulous reasoning in propositional default logic is $\Sigma^P_2$-complete [42], whose two sources of exponentiality are reflected by an extended PTTP inference engine along with model handling capacities. Interestingly, the approach is quite orthogonal to that of computing entire extensions. For instance, DeReS [25] performs very well on graph-based examples, like Hamiltonian cycle problems. Similar problem sets are also feasibly manageable with our system (albeit on a smaller scale); the performance is however subject to the ordering of the rules in the default logic program. On the other hand, we have observed impressively rapid answers due to short proofs on taxonomic knowledge bases, which made DeReS collapse.

Finally, the natural questions arise, why choose PTTP when it is not unproblematic as regards the number of generated Prolog rules? And, are there alternatives to the choice of PTTP? For the latter case, we remark that while there are approaches that could be used in place of PTTP, it is certainly the simplest (for instance, the approach of Umrigar and Pitchumani [91] allows a simpler coding via an extended set of connectives; this, however, does not allow to apply compilation techniques). For the former, we note that the blow-up of the resulting code is still linear in the number of clauses, default rules and propositional symbols. Practically speaking, we have so far not encountered any space problems for compiled knowledge bases. Nonetheless, it remains an interesting topic for future research how approaches avoiding the generation of all contrapositives like for instance [7,8] and correspondingly the resulting systems [9] can be enhanced in a similar way. Another major avenue for future research is the cross-fertilization between top-down and bottom-up approaches to default reasoning, manifested by the distinction among query answering and the computation of extensions. In this paper, we relied on the property of semi-monotonicity for obtaining local proof procedures. It will now be interesting to see how techniques like splitting [90] and in particular stratification [26], which aim at computing extensions in a local fashion, carry over to a query-oriented framework.

Acknowledgements

We are grateful to Mark Stickel for providing us with PTTP and for fruitful discussions during his stay at Darmstadt. Great thanks are also due to the anonymous referees for their detailed comments. This extends to Philippe Besnard and Pascal Nicolas, who provided a constant valuable feedback.

Appendix A. Compilation results on examples

For ultimate transparency, we provide the authentic knowledge base corresponding to Example (4) in Fig. A.1. The resulting (pretty-printed) Prolog code is given in Fig. A.2, whereby the Prolog code stemming from the query is put in a separate file in order to ease
re-compilation of subsequent queries. This code was obtained from a compilation without lemma handling.
These files along with the ones dealing with Example (10/11) can be retrieved at [83].

Appendix B. Proofs of theorems

B.1. Proofs of theorems in Section 2

Proof of Theorem 2.1. This is an immediate consequence of the general result expressed in Theorem 6.1. □

Proof of Theorem 2.2. The result follows from Theorem 2.1 by compactness and semi-monotonicity. □

B.2. Proofs of theorems in Section 3

For the proof of Theorem 3.2 we need the following lemma:

Lemma B.1. Let \( W \) be a set of formulas and \( \delta_1, \ldots, \delta_n \) be a sequence of default rules. Then,
\[
\forall i \in \{1, \ldots, n\} : W \cup \text{Conseq}(\delta_1, \ldots, \delta_{i-1}) \vdash \text{Prereq}(\delta_i)
\]
iff
\[
\forall i \in \{1, \ldots, n\} : W \cup \{\text{Prereq}(\delta_1) \rightarrow \text{Conseq}(\delta_1), \ldots, \text{Prereq}(\delta_{i-1}) \rightarrow \text{Conseq}(\delta_{i-1})\} \vdash \text{Prereq}(\delta_i).
\]

---

Fig. A.1. The knowledge base of Example (4).
Fig. A.2. The Prolog code resulting from Example (4).
Proof. The only-if part of Lemma B.1 is trivial since $\text{Conseq}(\delta_j)$ subsumes $\text{Prereq}(\delta_j) \rightarrow \text{Conseq}(\delta_j)$. To prove the if-part, let $j$ be an arbitrary number in $\{1, \ldots, i - 1\}$. In what follows, we abbreviate $\text{Prereq}(\delta_k)$ by $p_k$ and $\text{Conseq}(\delta_k)$ by $c_k$. Since 

$$W \cup \{p_1 \rightarrow c_1, \ldots, p_{j-1} \rightarrow c_{j-1}\} \vdash p_j \quad (B.3)$$

we obviously have:

$$W \cup \{p_1 \rightarrow c_1, \ldots, p_{j-1} \rightarrow c_{j-1}, p_j \rightarrow c_j, p_j, p_{j+1} \rightarrow c_{j+1}, \ldots, p_i \rightarrow c_{i-1}\} \vdash p_i. \quad (B.4)$$

Because of (i) $p_j \rightarrow c_j, p_j \vdash c_j$ and (ii) $c_j$ subsumes $p_j \rightarrow c_j$, $(B.4)$ is equivalent to

$$W \cup \{p_1 \rightarrow c_1, \ldots, p_{j-1} \rightarrow c_{j-1}, c_j, p_j, p_{j+1} \rightarrow c_{j+1}, \ldots, p_i \rightarrow c_{i-1}\} \vdash p_i. \quad (B.5)$$

But due to $(B.3)$, $(B.5)$ is equivalent to

$$W \cup \{p_1 \rightarrow c_1, \ldots, p_{j-1} \rightarrow c_{j-1}, c_j, p_j, p_{j+1} \rightarrow c_{j+1}, \ldots, p_{i-1} \rightarrow c_{i-1}\} \vdash p_i. \quad (B.6)$$

Applying this argumentation for each $j \in \{1, \ldots, i - 1\}$, we get Lemma B.1. □

Proof of Theorem 3.2. In what follows, let $(D', W')$ be the atomic format of $(D, W)$ and let $C_W$ be the clausal representation of $W'$. Due to [79, Theorem 3.3] it is sufficient to prove the theorem for default theory $(D', W')$.

(if-part) Let $R$ be a DME-refutation for $M$ with top-clause $c = \{\varphi^c\}$. Let $D = \{-\alpha_1, y_1\}, \ldots, \{-\alpha_k, y_k\}$ be the set of $\delta$-clauses used throughout $R$. In what follows, a $\delta$-clause $\{-\alpha_i, y_i\}$ is abbreviated by $d_i$. Let $\delta_i$ be the default corresponding to $d_i$. The tableau generated by $R$ is denoted by $T$.

Since $c$ is the top-clause of $R$, and a DME-refutation for $M$ can be viewed as a restricted ME-refutation for $M$, we know (since ME is sound) that $C_W \cup D \cup \{c\}$ is inconsistent. Since for each $i \in \{1, \ldots, k\}$, $d_i$ is subsumed by $y_i$, it follows that $W' \cup \{y_1, \ldots, y_n\} \vdash \varphi$.

It remains to prove (see Definition 2.2) that conditions (2) and (3) of Theorem 2.1 can be guaranteed. Condition (3) is obviously fulfilled if $W' \cup \{y_1, \ldots, y_n\}$ is consistent. But this is guaranteed by the compatibility-restriction of $\delta$-extension steps (note that the only possibility to use a $d_i$ in a derivation is the application of a compatible $\delta$-extension step).

To show that condition (2) holds is more difficult. This is due to the fact that in one subrefutation of $R$, a clause $d_j$ might be used to prove a subgoal $\neg\alpha_i$, whereas in another subrefutation of $R$, the $\delta$-clause $d_i$ is used to prove $\neg\alpha_j$. Hence the order of the $\delta$-clauses in the tableau generated by $R$ does not automatically give use the desired order of the defaults $\delta_1, \ldots, \delta_k$. However, such an ordered sequence $\delta_{i_1}, \ldots, \delta_{i_k}$ of defaults (which satisfies condition (2)) can be constructed in an iterative manner as follows:

Let $TC = \{t_1, \ldots, t_l\}$ be the tableau clauses in $T$ that were generated by $\delta$-extension steps. Let $c = \{-\alpha_i, y_i\}$ be an element from $TC$ with a maximal depth in $T$. Due to the definition of reduction steps, no reduction step used in the subrefutation $R_i$ of $\neg\alpha_i$ uses an ancestor literal of $\neg\alpha_i$. Hence, the subrefutation $R_i$ constitutes a refutation of $\neg\alpha_i$. Further (due to the maximal depth of $c$), all the clauses needed for $R_i$ of $\neg\alpha_i$ are clauses from $W'$. Hence, $W' \vdash \alpha_i$ and we can set $i_1 = i$. 


To determine $i_2$ we first modify $R$ (and $T$): every subrefutation of $\neg \alpha_1$ in $R$ is replaced by $R_i$. This is possible because due to the definition of reduction steps, no reduction step in $R_i$ can use an ancestor goal of $\neg \alpha_1$. Now, let $TC = \{t_1, \ldots, t_l\}$ be the tableau clauses in $T$ that were generated by $\delta$-extension steps using clauses from $\{d_1, \ldots, d_k\} - d_1$, and let $c = (-\alpha_i, y_1)$ be an element from $TC$ with a maximal depth in $T$. Again, (with the same argumentation as above) the subrefutation $R_i$ of $\neg \alpha_i$ constitutes a refutation of $\neg \alpha_i$. Then, due to the above modification of $R$ it is guaranteed that $R_i$ only uses clauses from $W$ and the $\delta$-clause $d_i$. With Lemma B.1 it follows that $W' \cup \{y_1\} \vdash \alpha_i$ and we can set $i_2 = i$.

This procedure can be iterated $k$ times (since only $k$ different $\delta$-clauses were used throughout $R$). The resulting sequence $\delta_1, \ldots, \delta_k$ fulfills condition (2) which completes the proof of the if-part.

(only-if-part) The proof of the only-if-part is an induction of the number of defaults used in the default proof for $\phi$. Let $c = \{\phi^c\}$.

($n = 0$) Since the default proof for $\phi$ uses no default rules, we have $W' \vdash \phi$. Due to the completeness of ME, there must exist a ME-refutation $R$ of $C_W \cup \{c\}$ with top-clause $c$. Since we have shown in Section 3.1 that a ME-derivation equals a DME-derivation without $\delta$-extension steps, $R$ is a DME-refutation, too.

($n > 0$) Let $\delta_1, \ldots, \delta_n$ be the sequence of default rules used in the default proof $DP$ for $\phi$. Note, that the existence of $DP$ implies the existence of a default proof $DP'$ for $\phi$ from $(\{\delta_2, \ldots, \delta_n\}, W \cup \{y_1\})$. Since $DP'$ uses only $n - 1$ default rules, the induction hypothesis tells us that there exists a corresponding DME-refutation $R$ for $C_W - \{\neg \alpha_1, y_1\} \cup \{\{y_1\}, c\}$ with top-clause $c$.

In what follows we modify $R$ in such a way that the resulting DME-refutation $R'$ is a DME-refutation for $C_W \cup \{c\}$ with top-clause $c$. To this end, we make use of the fact, that there must exist a proof for $\alpha_1$ from $(\emptyset, W)$. Hence, there exists a DME-refutation $R_1$ of $C_W \cup \{\neg \alpha_1\}$ with top-clause $\neg \alpha_1$ which uses no $\delta$-extension steps. Further, we can assume that $R_1$ does not contain extension steps with clause $\neg \alpha_1$ since each open goal $\alpha_1$ can be solved via a reduction step to the top-clause literal $\neg \alpha_1$. Hence, clause $\neg \alpha_1$ is only used as top-clause in $R_1$.

Now it is easy to construct $R'$. We simply replace each tableau clause $\{y_1\}$ in $R$ by the tableau clause $\{\neg \alpha_1, y_1\}$. This replacement corresponds to a replacement of an extension step with unit clause $\{y_1\}$ by a $\delta$-extension step with $\delta$-clause $\{\neg \alpha_1, y_1\}$. The resulting open goals can be solved via a subrefutation corresponding to $R_1$.

That each of the $\delta$-extension steps used in $R'$ is compatible is a consequence of condition (3) of Theorem 2.1 and the fact that the set of $\delta$-extension literals in the tableau generated by $R'$ equals $\{y_1, \ldots, y_n\}$ (consider the construction of $R'$ given above).

**Proof of Theorem 3.3.** Suppose the conditions of Definition 3.7 are fulfilled. Let $M_W$ be the set of $\omega$-clauses in $M$, $M_D$ be the set of $\delta$-clauses in $M$, and let $D'$ be the set of defaults corresponding to the elements of $M_D$.

Then it is obvious that there exists a DME-refutation $R$ with top-clause $\lambda(o)$ that uses only clauses from $M_W \cup M_D \cup \{\{\lambda(o_1)\}\} \cup \cdots \cup \{\{\lambda(o_n)\}\}$ for extension steps. Further, the use of unit clauses $\{\lambda(o_i)\}$ ($i \in [1, \ldots, n]$) in $R$ can be restricted to $\omega$-extension steps that are applied to nodes which do not have ancestors which are $\delta$-resulting nodes (note that these unit clauses are only needed to "substitute" reduction steps during $D$; due to Definition 3.5,
these reduction steps were not used in a subrefutation of some $\delta$-resulting literal generated by $D$. This implies (see Theorem 3.2) that there is an enumeration $\langle \delta_i \rangle_{i \in \mathbb{N}}$ of $D$ such that:

\[
M_w \cup \text{Conseq}(\{\delta_1, \ldots, \delta_{i-1}\}) \vdash \text{Prereq}(\delta_i), \quad (B.7)
\]

\[
M_w \cup \text{Conseq}(\{\delta_1, \ldots, \delta_{i-1}\}) \not\vdash \neg \text{Conseq}(\delta_i). \quad (B.8)
\]

Now, it is quite easy to recognize that for every $i \in \{1, \ldots, n\}$ there exists a default proof of $\neg(\lambda(\alpha_i))$ from $(M', D')$ where

\[
M' = M_w \cup \{\lambda(\alpha_i)\} \cup \cdots \cup \{\lambda(\alpha_{i-1})\} \cup \{\lambda(\alpha_{i+1})\} \cup \cdots
\]

\[
\cup \{\lambda(\alpha_n)\} \cup \{\lambda(\alpha)\}.
\]

This is due to (B.7) and (B.8) and the fact that $M_w \cup M_D \cup \{\lambda(\alpha_0)\} \cup \cdots \cup \{\lambda(\alpha_n)\} \cup \{\lambda(\alpha)\}$ must be inconsistent (otherwise $R$ would not exist). Hence, we know that there exists a corresponding DME-refutation $R'$ with top-clause $\{\lambda(\alpha_i)\}$.

Furthermore, due to (B.7) and (B.8), we know that the subproofs of $\delta$-extension-resulting nodes in $R$ can be used to prove $\delta$-extension-resulting nodes generated during $R'$ without any modification. This implies, that in the course of $R'$, the unit clauses $\{\lambda(\alpha_j)\}$ ($j \in \{1, \ldots, i-1, i+1, \ldots, n\}$) and $\{\lambda(\alpha)\}$ are only used for extension steps to prove a $\omega$ extension resulting literal that has no node among its successors that is a $\delta$ extension resulting node.

Now we can turn to the proof of Theorem 3.3. For readability, we distinguish the nodes given in this theorem by attaching a prime to each identifier. We thus have:

Let $T$ be a tableau generated by a DME-derivation from $C_w \cup C_D$, and let $b = \langle o_1', \ldots, o_{v'}', \ldots, o_m' \rangle$ be a branch of $T$.

If $o_{v+1}', \ldots, o_m'$ are $\omega$-extension-resulting nodes and $l \subseteq \{\lambda(o_0'), \ldots, \lambda(o_m')\}$, then $b$ can be marked as closed without losing soundness.

Let $\{k_0, \ldots, k_n\}$ be a subset of $\{v, \ldots, m\}$ such that $l = \{\lambda(o_{k_0}'), \ldots, \lambda(o_{k_n}')\}$ and $k_0 < k_2 < \cdots < k_n$. For simplicity, we assume that $m = k_n$ (i.e., $o_{k_n}'$ is the leaf node of $b$) (otherwise, lemma 1 could have been applied earlier in course of the derivation).

Due to the above considerations, we know that there is a DME-refutation $R''$ with top-clause $\{\lambda(o_{k_n}')\}$ which uses clauses from $M_w$, $M_D$ and the unit clauses $\{\lambda(o_j')\}$ (where $j \neq n$) for extension steps. Further, these unit clauses are used during $R''$ only to prove $\omega$-extension-resulting literals with no $\delta$-extension-resulting literals among their successors. This, together with the fact that the $\delta$-clauses in $R''$ can be used for any $\delta$-extension step without violating consistency (see the conditions of the theorem), implies that $R''$ can be applied to $o_{v+1}'$, $\ldots$, $o_m'$ are $\omega$-extension-resulting literals). Hence, $b$ can be marked as closed. 

**Proof of Theorem 3.4.** Since $T$ is not blockwisely regular, it must violate condition (1) or condition (2) of Definition 3.10. First, we show that in case $T$ violates condition
(2), there exists a DME-refutation $R''$ that generates a tableau $T''$ which does not violate condition (2).

If $T$ violates condition (2), $T$ contains a branch $b = o_1, \ldots, o_n$ that violates condition (2), that is $b$ contains two $\delta$-extension resulting nodes, $o_i$ and $o_j$ ($i < j$) say, such that $\text{prev}(o_i)$ and $\text{prev}(\text{node}_j)$ are labeled with the same literal. Since $o_j$ is a $\delta$-extension resulting node, no reduction step that was applied to a successor node of $o_j$ uses an ancestor node of $o_j$ (note that no reduction step can be applied to $o_j$). But then, it is obvious that the DME-subrefutation applied to $\text{prev}(o_j)$ can be applied directly to $\text{prev}(o_i)$. The resulting DME-refutation $R_1$ is strictly smaller than $R$, and the tableau generated by $R_1$ is strictly smaller than $T$. Hence, this transformation can only be applied finitely many times and the resulting DME-refutation $R''$ generates a tableau $T''$ that agrees with condition (2).

Second, we show that in case $T''$ violates condition (1), there exists a DME-refutation $R'$ that is blockwisely regular. Suppose $T''$ violates condition (1), and let $o_1, \ldots, o_n$ be a block such that for two different nodes, $o_i$ and $o_j$ say, $\lambda^{T''}(o_i) = \lambda^{T''}(o_j)$. Then, let $S_1$ be a clause set containing

(i) $\{\lambda^{T''}(o_1)\}$,

(ii) each tableau clause whose corresponding nodes are successors of $o_1$ and belong to blocks starting with $o_1$, and

(iii) the unit clauses $[\lambda^{T''}(o_{k_1})], \ldots, [\lambda^{T''}(o_{k_l})]$, where $o_{k_1}, \ldots, o_{k_l}$ are extension nodes generated by $\delta$-extension steps such that for each $1 \leq i \leq l$, $o_{k_i}$ is a successor of $o_1$ and no node between $o_1$ and $o_{k_i}$ was generated by a $\delta$-extension step.

Since $T''$ is a closed tableau, it is quite easy to see that there exists an ME-refutation of $S_1$ with top-clause $\{\lambda^{T''}(o_1)\}$. But then there also exists a regular ME-refutation $R_S$ of $S_1$ with top-clause $\{\lambda^{T''}(o_1)\}$. Let $D_S$ be the subderivation applied to $\lambda^{T''}(o_1)$ that contains all derivation steps of $R_S$ that use clauses from $M$ (note that each of these clauses must be a $\omega$-clause). That is, the only derivation steps of $R_S$ that do not belong to $D_S$ are the initialization step and extension steps using the unit clauses in $[\lambda^{T''}(o_{k_1})], \ldots, [\lambda^{T''}(o_{k_l})]$.

Now, we construct a DME-refutation $R_2$ from $R''$ as follows: all derivation steps belonging to the subrefutation of $o_1$ in $R''$ are removed. The resulting derivation $D_1$ generates a tableau with one open goal, namely $\lambda^{T''}(o_1)$. Now, apply $D_S$ to this open goal. The resulting derivation $D_2$ contains open goals that are labeled with literals from $[\neg\lambda^{T''}(o_{k_1}), \ldots, \neg\lambda^{T''}(o_{k_l})]$ (in $R_S$, extension steps with the unit clauses $[\lambda^{T''}(o_{k_1})], \ldots, [\lambda^{T''}(o_{k_l})]$ are applied to these open goals). Finally, for each $1 \leq i \leq l$, the DME-subrefutation that was applied to $o_{k_i}$ in $R''$ is applied to the open goals labeled with $\lambda^{T''}(o_{k_i})$.

Note, that the tableau $T_2$ which is generated by the resulting DME-refutation $R_2$ cannot violate condition (2). This is because the transformation only affects $\omega$-extension steps and reduction steps to literals stemming from $\omega$-clauses. Inference steps that use literals from $\delta$-clauses remain unchanged. In particular, it is guaranteed that in case some literal $L$ in $T_2$ that stems from a $\delta$-clause and has an ancestor literal $L'$ that stems from a $\delta$-clause, too, this also holds for $T''$ (and therefore $T_2$ cannot violate condition (2)).

However, $R_2$ may be longer than $R''$ and $T_2$ may even contain more blocks than $T''$ that violate blockwise regularity. However, it is easy to see that the transformation can only be applied finitely many times. To this end, observe that the maximal number $m_1$ of blocks on one branch does not increase during the transformation. Since $m_1$ is finite and the maximal
number of literals per clause is finite, too, the maximal number $m_2$ of blocks in a tableau is limited by a fix constant. Hence, our transformation can be applied at most $m_2$ times. Afterwards, the resulting DME-refutation $R'$ must generate a blockwisely regular tableau $T'$.

**B.3. Proofs of theorems in Sections 4 and 6**

**Proof of Theorem 4.1.** Let $W$ be a set of formulas and $(\delta_i)_{i \in I}$ a sequence of normal default rules. Consider $\delta_i$ for some $i \in I$.

By definition of $\nabla$, we have

$$\nabla(\delta_i, (m, W, (\delta_j)_{j \leq i})) = (m', W, (\delta_j)_{j \leq i})$$

iff $m'$ is some model for $W \cup \{\text{Conseq}(\delta_j) | j \leq i\}$. This includes the case of $m = m'$ since $m$ is a model of $W \cup \text{Conseq}((\delta_0, \ldots, \delta_{i-1}))$ and by definition of $\nabla$, $m \models \text{Conseq}(\delta_i)$.

And

$$\nabla(\delta_i, (m, W, (\delta_j)_{j \leq i})) = \bot$$

iff there is no model for $W \cup \{\text{Conseq}(\delta_j) | j \leq i\}$.

We thus distinguish two cases:

1. $W \cup \text{Conseq}((\delta_0, \ldots, \delta_{i-1})) \not\models \neg \text{Conseq}(\delta_i)$. This is equivalent to $W \cup \text{Conseq}((\delta_0, \ldots, \delta_i)) \not\models \bot$. By propositional logic, this is itself equivalent to the existence of a model $m$ such that

$$m \models W \cup \{\text{Conseq}(\delta_j) | j \leq i\}$$

which proves the claim.

2. $W \cup \text{Conseq}((\delta_0, \ldots, \delta_{i-1})) \models \neg \text{Conseq}(\delta_i)$. This is equivalent to $W \cup \text{Conseq}((\delta_0, \ldots, \delta_i)) \not\models \bot$. By propositional logic, this is itself equivalent to the non-existence of a model for $W \cup \{\text{Conseq}(\delta_j) | j \leq i\}$ which establishes the claim. \qed

**Proof of Theorem 6.1.**

(if-part) Let $(E, C)$ be a contextual extension of semi-monotonic contextual default theory $(D, W)$. Consider the set of generating contextual default rules

$$\Gamma = \{ |\alpha| : \beta_C, |\beta_E| \in D | \alpha_E \in E, \neg \beta_C \notin C, \neg \beta_E \notin E \}. \quad (B.9)$$

The case where $\Gamma = \emptyset$ is trivial so that we concentrate on non-empty $\Gamma$: according to [12, Theorem 5.1], $\Gamma$ satisfies (24). That is,

$$E = \text{Th}(E) = \text{Th}(W \cup \text{Conseq}(\Gamma)), \quad (B.10)$$

$$C = \text{Th}_C(C) = \text{Th}_{W \cup \text{Conseq}(\Gamma) \cup \text{Justif}_C(\Gamma)}(\text{Justif}_E(\Gamma)). \quad (B.11)$$

Condition (25), that is groundedness, is also verified by $\Gamma$, due to the definition of a contextual extension.

For ensuring condition (26), we observe that $\text{Justif}_C(\Gamma) \neq \emptyset$ and $\text{Justif}_E(\Gamma) \neq \emptyset$ provided that $\Gamma \neq \emptyset$. We have

$$\neg \beta_C \notin \text{Th}_{W \cup \text{Conseq}(\Gamma) \cup \text{Justif}_C(\Gamma)}(\text{Justif}_E(\Gamma))$$
for all $\beta_C \in \text{Justif}_C(\Gamma')$. That is,

$$\neg \beta_C \notin \bigcup_{\beta_E \in \text{Justif}_E(\Gamma)} \text{Th}(W \cup \text{Conseq}(\Gamma') \cup \text{Justif}_C(\Gamma') \cup \{\beta_E\}).$$

Or,

$$W \cup \text{Conseq}(\Gamma') \cup \text{Justif}_C(\Gamma') \cup \{\beta_E\} \not\vdash \neg \beta_C$$

for all $\beta_E \in \text{Justif}_E(\Gamma)$. By $\beta_C \in \text{Justif}_C(\Gamma')$, we get

$$W \cup \text{Conseq}(\Gamma') \cup \text{Justif}_C(\Gamma') \not\vdash \neg \beta_E$$

for all $\beta_E \in \text{Justif}_E(\Gamma)$. By monotonicity, this property holds for any subset $\Gamma' \subseteq \Gamma$, too, thus establishing condition (26).

Finally, suppose that $\Gamma$ is not maximal. Then, there is some $\Gamma' \supseteq \Gamma$ with some

$$|\alpha| : \beta_C | \beta_E | \not\in \Gamma' \setminus \Gamma$$

satisfying conditions (25) and (26). Condition (25) implies $\alpha \in E$. Moreover, we have

$$W \cup \text{Conseq}(\Gamma') \cup \text{Justif}_C(\Gamma') \not\vdash \neg \beta_E$$

for all $\beta_E \in \text{Justif}_E(\Gamma')$. The fact $\Gamma'$ satisfies condition (24), yields by inverting the above manipulations that $\beta_C \notin C$; also, we get from the previous non-derivability proposition that $\beta_E \notin E$ by monotonicity. Consequently,

$$|\alpha| : \beta_C | \beta_E | \in \Gamma;$$

a contradiction.

(only-if-part) Let $\langle \delta_i \rangle_{i \in I}$ be an enumeration of some maximal $D' \subseteq D$ satisfying conditions (24)–(26).

Consider the sequence of default theories

$$((\delta_j \in D \mid j < i), W)$$

for $i \in I$ along with a family of pairs of sets of formulas $((E_i, C_i))_{i \in I}$ such that

$$E_i = \text{Th}(W \cup \text{Conseq}((\delta_j \in D \mid j < i))),$$

$$C_i = \text{Th}_{E_i \cup \text{Justif}_C((\delta_j \in D \mid j < i))}((\text{Justif}_E((\delta_j \in D \mid j < i)))).$$

With these, we define

$$\Gamma_i = \left\{ |\alpha| : \beta_C | \beta_E | \in D \mid \alpha_E \in E_i, \neg \beta_C \notin C_i, \neg \beta_E \notin E_i \right\}.$$

By construction of $((E_i, C_i))_{i \in I}$ and $(\Gamma_i)_{i \in I}$, we have

$$E = \bigcup_{i \in I} E_i, \quad C = \bigcup_{i \in I} C_i, \quad \Gamma = \bigcup_{i \in I} \Gamma_i$$

where $E$ and $C$ are defined in (24) and $\Gamma$ is defined in (B.9).
We prove by induction on \( I \) that \((E_{i+1}, C_{i+1})\) is an extension of \(((\delta_j \in D \mid j \leq i), W)\) and that \(\{\delta_j \in D \mid j \leq i\} \subseteq \Gamma_{i+1}\) for all \( i \in I \). From this, it follows by semi-monotonicity that \((E, C)\) is a contextual extension of \((D', W)\).

Clearly, \((\text{Th}(W), \text{Th}_W(\emptyset))\), or equivalently \((\text{Th}(W), \text{Th}(W))\), is an extension of \((\emptyset, W)\).

Now, suppose \((E_i, C_i)\) is an extension of \(((\delta_j \in D \mid j < i), W)\) with \(\{\delta_j \in D \mid j < i\} \subseteq \Gamma_i\). By semi-monotonicity, there is contextual extension \((E', C')\) of \(((\delta_j \in D \mid j < i), W)\) such that \(E_i \subseteq E'\) and \(C_i \subseteq C'\).

Depending on whether
\[
\delta_i = \frac{|\alpha| : \beta C | \beta E|}{\gamma}
\]
contributes to \((E', C')\), we must consider two cases:

- **We have** \(E' \neq E_i\) and \(C' \neq C_i\) and so either \(\alpha E \notin E'\) or \(-\beta C \notin C'\) or \(-\beta E \notin E'\).

  Because \(W \subseteq E_i \subseteq E'\) and \(\{\delta_j \in D \mid j < i\} \subseteq \Gamma_i\) imply \(\alpha E \in E'\) by (25), we thus have \(-\beta C \notin C'\) or \(-\beta E \notin E'\).

  According to (26)
  \[
  W \cup \text{Conseq}((\delta_0, \ldots, \delta_i)) \cup \text{Justif}_C((\delta_0, \ldots, \delta_i)) \not\models \neg \beta
  \]
  for all \(\beta \in \text{Justif}_E((\delta_0, \ldots, \delta_i))\).

  By the fact that \(E' = E_i = \text{Th}(W \cup \text{Conseq}((\delta_j \in D \mid j < i)))\) and monotonicity, we deduce that \(-\beta E \notin E'\) (recall that \(\beta E = \text{Justif}_E(\delta_i)\)).

  Rewriting the above non-derivability statement as done in the first part, yet in reverse order, yields:
  \[
  -\beta C \notin \text{Th}_W(\text{Conseq}((\delta_0, \ldots, \delta_i)) \cup \text{Justif}_C((\delta_0, \ldots, \delta_i)) \cup \text{Justif}_E((\delta_0, \ldots, \delta_i)))
  \]
  for all \(\beta \in \text{Justif}_C((\delta_0, \ldots, \delta_i))\). As in the case of \(E'\) and \(\beta E\), we obtain by appeal to the definition of \(C'\) and monotonicity that \(-\beta C \notin C'\) (recall that \(\beta C = \text{Justif}_C(\delta_i)\)).

  This is in contradiction to the assumption that \(E' = E_i\) and \(C' = C_i\).

- **We have** \(E' = E_{i+1}\) and \(C' = C_{i+1}\) and so \(\alpha E \in E'\) and \(-\beta C \notin C'\) and \(-\beta E \notin E'\).

  This implies that \((E_{i+1}, C_{i+1})\) is a contextual extension of \(((\delta_j \in D \mid j \leq i), W)\) and that \(\{\delta_j \in D \mid j \leq i\} \subseteq \Gamma_{i+1}\).

We have shown that \((E, C)\) is a contextual extension of \((D', W)\). By maximality of \(D'\), any contextual default rule \(\delta \in D \setminus D'\) violates either (25) or (26). This disqualifies any such \(\delta\) to contribute to any contextual extension \((E', C')\) with \(E \subseteq E'\) and \(C \subseteq C'\) that could be obtainable by semi-monotonicity. Hence, \((E, C)\) is also a contextual extension of \((D, W)\). \(\square\)

**Proof of Theorem 6.2.** Let \(W\) be a set of formulas and \((\delta_i)_{i \in I}\) a sequence of contextual default rules such that \(\delta_i \in D^*\) for all \(i \in I\). Consider
\[
\delta_i = \frac{|\alpha| : \beta C | \beta E|}{\gamma}
\]
for some \(i \in I\) and \(K = \{0, \ldots, i - 1\}\) and \(L = \{0, \ldots, i\}\).

By definition of \(\nabla\), we have
\[
\nabla(\delta_i, (M, W, (\delta_k)_{k \in K})) = (M', W, (\delta_l)_{l \in L})
\]
iff there is a non-empty set of models $M' \subseteq L M_W(L)$. This comprises the case where $M = M'$ because by assumption $M \subseteq_K M_W(K)$, that is,

$$M \subseteq \bigcup_{k \in K} M^k_W(K) \quad \text{and} \quad M \cap M^k_W(K) \not= \emptyset \quad \text{for all } k \in K.$$ 

Because $M \models \gamma \land \beta_C$, we have for all $k \in K$ that

$$M^k_W(K) = \{ m \mid m \models W \cup \text{Conseq}(K) \cup \text{Justif}_C(K) \cup \{\text{Justif}_E(\delta_k)\} \} = \{ m \mid m \models W \cup \text{Conseq}(L) \cup \text{Justif}_C(L) \cup \{\text{Justif}_E(\delta_k)\} \} = M^k_W(L),$$

while $m \models \beta_E$ for some $m \in M$ gives us (by recalling that $\beta_E = \text{Justif}_E(\delta_i)$)

$$M \cap M^k_W(K) \not= \emptyset.$$ 

Hence, $M$ is also a non-empty set of models such that $M \subseteq L M_W(L)$.

And

$$\nabla(\delta_i, \langle M, W, \langle \delta_k \rangle_{k \in K} \rangle) = \bot$$

iff there is no set of models $M$ such that $M \subseteq L M_W(L)$. This follows from the fact that there is no $M'' \subseteq_K M_W(K)$ satisfying $\gamma \land \beta_C$ and including some $m$ with $m \models \beta_E$.

In analogy to Proof 4.1, we consider two cases: we thus distinguish two cases:

1. Condition (26) is true. That is,

$$W \cup \text{Conseq}((\delta_0, \ldots, \delta_i)) \cup \text{Justif}_C((\delta_0, \ldots, \delta_i)) \not\vdash \neg \text{Justif}_E(\delta_k)$$

for all $k \in \{0, \ldots, i\}$. Or equivalently,

$$W \cup \text{Conseq}((\delta_0, \ldots, \delta_i)) \cup \text{Justif}_C((\delta_0, \ldots, \delta_i)) \cup \{\text{Justif}_E(\delta_k)\} \not\vdash \bot$$

for all $k \in \{0, \ldots, i\}$.

By propositional logic, this is equivalent to the existence of a model $m_k$ such that

$$m_k \models W \cup \text{Conseq}((\delta_0, \ldots, \delta_i)) \cup \text{Justif}_C((\delta_0, \ldots, \delta_i)) \cup \{\text{Justif}_E(\delta_k)\}$$

for all $k \in \{0, \ldots, i\}$.

Defining $M = \bigcup_{k \in \{0, \ldots, i\}} \{ m_k \}$ establishes the claim.

2. Condition (26) is false. That is,

$$W \cup \text{Conseq}((\delta_0, \ldots, \delta_i)) \cup \text{Justif}_C((\delta_0, \ldots, \delta_i)) \vdash \neg \text{Justif}_E(\delta_k)$$

for some $k \in \{0, \ldots, i\}$. Or equivalently,

$$W \cup \text{Conseq}((\delta_0, \ldots, \delta_i)) \cup \text{Justif}_C((\delta_0, \ldots, \delta_i)) \cup \{\text{Justif}_E(\delta_k)\} \vdash \bot$$

for some $k \in \{0, \ldots, i\}$.

By propositional logic, this is equivalent to the non-existence of a model $m_k$ satisfying

$$m_k \models W \cup \text{Conseq}((\delta_0, \ldots, \delta_i)) \cup \text{Justif}_C((\delta_0, \ldots, \delta_i)) \cup \{\text{Justif}_E(\delta_k)\}$$

for some $k \in \{0, \ldots, i\}$.

In other words, $M^k_W(L) = \emptyset$ for some $k \in \{0, \ldots, i\}$. That is, there cannot be any set of models $M$ satisfying $M \subseteq L M_W(L)$ since $M \cap M^k_W(L) = \emptyset$ for some $k \in \{0, \ldots, i\}$. □
References

89–148.
(Ed.), Proceedings Conference on Automated Deduction, Lecture Notes in Artificial Intelligence, Vol. 607,
[5] F. Baader, B. Hollunder, Embedding defaults into terminological knowledge representation formalisms,
in: B. Nebel, C. Rich, W. Swartout (Eds.), Proceedings 3rd International Conference on the Principles of
Conference on Automated Deduction, Lecture Notes in Artificial Intelligence, Vol. 814, Springer, Berlin,
(Ed.), Proceedings Conference on Automated Deduction, Lecture Notes in Artificial Intelligence, Vol. 814,
M. Ornaghi (Eds.), Proceedings Tableaux-96, Lecture Notes in Artificial Intelligence, Vol. 1071, Springer,
(Eds.), Proceedings 9th International Symposium on Methodologies for Intelligent Systems, Lecture Notes
XRay system, Technical Report, Institute of Informatics, University of Potsdam, 1998. (This is an extended
version of [59].) Full version in preparation.
[65] D. Poole, A Theorist to Prolog Compiler, Logic Programming and Artificial Intelligence Group, Department of Computer Science, University of Waterloo, October 1987.


