Implementing Prioritized Circumscription by Computing Disjunctive Stable Models[⋆]

Emilia Oikarinen[⋆⋆] and Tomi Janhunen

Helsinki University of Technology TKK
Department of Information and Computer Science
P.O. Box 5400, FI-02015 TKK, Finland
{Emilia.Oikarinen,Tomi.Janhunen}@tkk.fi

Abstract. The stable model semantics of disjunctive logic programs is based on minimal models which assign atoms false by default. While this feature is highly useful and leads to concise problem encodings, it occasionally makes knowledge representation with disjunctive rules difficult. Lifschitz' parallel circumscription provides a remedy by introducing atoms that are allowed to vary or to have fixed values while others are falsified. Prioritized circumscription further refines this setting in terms of priority classes for atoms being falsified. In this paper, we present a linear and faithful transformation to embed prioritized circumscription into disjunctive logic programming in a systematic fashion. The implementation of the method enables the use of disjunctive solvers for computing prioritized circumscription. The results of an experimental evaluation indicate that the method proposed herein compares favorably with other existing implementations.

Keywords: Prioritized circumscription, disjunctive stable models, linear transformation, answer set programming.

1 Introduction

In *answer set programming* (ASP) a problem at hand is formalized as a logic program so that its answer sets, or more formally *(disjunctive) stable models* [2,3], correspond to the solutions of the problem—to be computed using a solver for disjunctive logic programs. Numerous applications of disjunctive logic programs have emerged since efficient solvers such as, for example, DLV [4], and GNT [5], for computing answer sets became available. The stable model semantics of disjunctive logic programs is based on *minimal models* which makes every atom appearing in a disjunctive logic program false by default. While this feature is highly useful—leading to concise encodings of problems as disjunctive programs—it occasionally makes knowledge representation with disjunctive rules difficult. For instance, the representation of Reiter-style minimal diagnoses [6] is complicated by the fact that in addition to abnormality atoms, also atoms describing the state of the system being diagnosed become subject to falsification.

[⋆] This research has been partially funded by the Academy of Finland under project #122399. A preliminary version of this work appeared in [1].

[⋆⋆] The financial support from Helsinki Graduate School in Computer Science and Engineering, Nokia Foundation, Finnish Foundation for Technology Promotion TES, Emil Aaltonen foundation, and Finnish Cultural Foundation is gratefully acknowledged.

This problem can be alleviated by a more refined control of minimization provided by *parallel circumscription* [7] which allows certain atoms to *vary* or to have *fixed* truth values. In particular, varying atoms enhance the knowledge representation capability over ordinary circumscription [8]. The scheme of *prioritized circumscription* [7] generalizes this setting with priority classes for atoms being minimized. Together, these principles provide an elegant solution to the problem raised in the first paragraph: atoms describing state should vary and priorities can be specified among those representing abnormality. We want to bring these enhanced notions of minimality to the realm of disjunctive logic programming so that they can be advisedly exploited in problem encodings. To this end, we have already addressed parallel circumscription and provided a *linear* and *faithful* translation into disjunctive logic programming [9].

The goal of this paper is to cover the prioritized case accordingly, extending the preliminary version of this work presented in [1]. Our translation-based approach effectively removes varying and fixed atoms as well as priority classes in terms of a transformation. We have also implemented the method, and give an experimental evaluation contrasting our tool with others developed for the same purpose [10,11,12].

The rest of this paper is organized as follows. In Section 2, we review the syntax and semantics of disjunctive logic programs. Section 3 is devoted to an introduction of parallel and prioritized circumscription [7] in the propositional case. A linear transformation from prioritized circumscription to disjunctive logic programs is presented in Section 4. The results from an experimental evaluation of the tool that implements the linear transformation are reported in Section 5. We end this paper with a brief discussion about related work in Section 6.

2 Disjunctive Logic Programs

We review in this section the basic concepts of disjunctive logic programs (DLPs) in the propositional case. A *disjunctive logic program* is a finite set of *disjunctive rules* of the form

$$a_1 \vee \cdots \vee a_n \leftarrow b_1, \ldots, b_m, {\sim}c_1, \ldots, {\sim}c_k \tag{1}$$

where $n, m, k \geq 0$, and $a_1, \ldots, a_n$, $b_1, \ldots, b_m$, and $c_1, \ldots, c_k$ are propositional atoms. Since the order of atoms is considered insignificant, we use $A \leftarrow B, {\sim}C$ as a shorthand for rules of form (1), where A, B, and C denote the sets of atoms $\{a_1, \ldots, a_n\}$, $\{b_1, \ldots, b_m\}$, and $\{c_1, \ldots, c_k\}$, respectively, and ${\sim}C = \{{\sim}c \mid c \in C\}$ for any set of atoms C.

The basic intuition behind (1) is that if each atom in the positive body B and none of the atoms in the negative body C can be inferred, then some atom in the head A can be inferred. If $C = \emptyset$, we have a *positive* rule written $A \leftarrow B$. When both B and C are empty, we have a *disjunctive fact*, written $A \leftarrow$. If A is empty, then we have a *constraint*, written $\bot \leftarrow B, {\sim}C$. Given a DLP Π, we write $\mathrm{At}(\Pi)$ for its *signature*, that is, the set of atoms appearing in the rules of Π.

Given any DLP Π, we define an *interpretation* M for Π as a subset of $\mathrm{At}(\Pi)$. An atom $a \in \mathrm{At}(\Pi)$ is *true* under M (symbolically $M \models a$) if and only if $a \in M$, otherwise a is *false* under M. For a negative literal ${\sim}a$, we define $M \models {\sim}a$ if and only if $M \not\models a$. A set L of literals is satisfied by M (denoted by $M \models L$) if and only if

$M \models l$, for every $l \in L$. We also define $M \models \bigvee L$, providing $M \models l$ for some $l \in L$. An interpretation $M \subseteq \mathrm{At}(\varPi)$ is a (*classical*) *model* of a DLP $\varPi$, denoted $M \models \varPi$, if and only if for every rule $A \leftarrow B, {\sim}C \in \varPi$, it holds that $M \models B \cup {\sim}C$ implies $M \models \bigvee A$. It is typical in logic programming that atoms are assumed false by default. In case of a *positive* DLP (PDLP) $\varPi$ this is formalized in terms of *minimal models* $M \models \varPi$ for which there is no $N \subset M$ such that $N \models \varPi$. We write $\mathrm{MM}(\varPi)$ for the set of minimal models associated with a PDLP $\varPi$. To extend the semantics for DLPs involving negation a *partial evaluation* technique proposed by Gelfond and Lifschitz [2] is applied.

Definition 1. *Given a DLP $\varPi$, an interpretation $M \subseteq \mathrm{At}(\varPi)$ is a* stable model *of $\varPi$ if and only if $M \in \mathrm{MM}(\varPi^M)$, where*

$$\varPi^M = \{A \leftarrow B \mid A \leftarrow B, {\sim}C \in \varPi \textit{ and } M \cap C = \emptyset\}$$

is the reduct of $\varPi$ with respect to M.

In the sequel, the set of stable models of a DLP $\varPi$ is denoted by $\mathrm{SM}(\varPi)$.

3 Parallel and Prioritized Circumscription

In this section we introduce *parallel* and *prioritized circumscription* [7] in the propositional case. Parallel circumscription is based on a notion of minimality which partitions atoms in three disjoint categories.

Definition 2. *Let $\varPi$ be a PDLP and let $P, V, F \subseteq \mathrm{At}(\varPi)$ be* disjoint *sets of atoms such that $\mathrm{At}(\varPi) = P \cup V \cup F$. A model $M \models \varPi$ is $\langle P, V, F\rangle$-minimal if and only if there is no $N \models \varPi$ such that (i) $N \cap P \subset M \cap P$ and (ii) $N \cap F = M \cap F$.*

By this definition, atoms in P are subject to minimization, that is, falsified as far as possible, while the truth values of atoms in V may vary freely and the truth values of atoms in F are kept fixed. Note that in the notation of $\langle P, V, F\rangle$-minimality one of the sets P, V, and F is actually redundant, as given any two of the sets, the third one is implicitly clear from the context. Example 1 illustrates the use of varying atoms in a concise representation for the model-based diagnosis [6] of digital circuits.

Example 1. Consider a positive DLP[1]

$$\begin{aligned}\varPi_{\mathrm{diag}} = \{a \vee b \vee ab_1.\;\; ab_1 \leftarrow a, b.\;\; b \vee c \vee ab_2.\;\; ab_2 \leftarrow b, c.\\ c \vee d \vee ab_3.\;\; ab_3 \leftarrow c, d.\;\; \bot \leftarrow a.\;\; \bot \leftarrow d\}\end{aligned}$$

representing a sequence of three *inverters* with observations $\neg a$ and $\neg d$ indicating a fault. By setting $P = \{ab_1, ab_2, ab_3\}$ and $V = \{a, b, c, d\}$ we obtain minimal diagnoses $\{ab_1, c\}$, $\{ab_2, b, c\}$, and $\{ab_3, b\}$ as $\langle P, V, \emptyset\rangle$-minimal models. In contrast, the $\langle P \cup V, \emptyset, \emptyset\rangle$-minimal models of $\varPi_{\mathrm{diag}}$, that is, the stable models of $\varPi_{\mathrm{diag}}$, include a spurious non-minimal diagnosis $\{ab_1, ab_2, ab_3\}$. ■

[1] A full stop is used to separate rules in a program and the symbol "$\leftarrow$" is omitted in the case of disjunctive facts, that is, when the body of a rule is empty.

The set of all $\langle P, V, F\rangle$-minimal models of Π is denoted by $\mathrm{MM}_{P,V,F}(\Pi)$. The conventional case that all atoms are subject to minimization is covered by $\mathrm{MM}(\Pi) = \mathrm{MM}_{\mathrm{At}(\Pi),\emptyset,\emptyset}(\Pi) = \mathrm{SM}(\Pi)$ for a PDLP Π. We write $\mathrm{Circ}(\Pi, P, V, F)$ to denote the parallel circumscription of a PDLP Π.

An extended notation $\mathrm{Circ}(\Pi, P_1 > \cdots > P_k, V, F)$ is introduced to represent the prioritized circumscription of Π which includes the parallel circumscription of Π as its special case, that is, when $k = 1$. The idea is that atoms in P_1 are falsified with the highest priority, those in P_2 with the next highest priority, and so on. Lifschitz [7] shows that $\mathrm{Circ}(\Pi, P_1 > \cdots > P_k, V, F)$ corresponds to the conjunction

$$\bigwedge_{i=1}^{k} \mathrm{Circ}(\Pi, P_i, P_{i+1} \cup \ldots \cup P_k \cup V, P_1 \cup \ldots \cup P_{i-1} \cup F). \tag{2}$$

The formula (2) does not have a direct interpretation as a DLP but such a representation can be obtained using the translation from [9]. A drawback is that $2k$ copies of Π must be created which gives a quadratic nature for the overall transformation [10] because $k \leq |\mathrm{At}(\Pi)| \leq \|\Pi\|$, that is, the length of Π in symbols. Therefore, we get better premises for the development of a linear representation if Definition 2 is generalized to the case of prioritized circumscription.

Definition 3 ([11]). *A model $M \models \Pi$ for a positive DLP Π is $\langle P_1 > \cdots > P_k, V, F\rangle$-minimal if and only if there is no $N \models \Pi$ such that*

(i) $N \cap (P_1 \cup \ldots \cup P_{i-1}) = M \cap (P_1 \cup \ldots \cup P_{i-1})$ *and* $N \cap P_i \subset M \cap P_i$ *for some* $1 \leq i \leq k$*; and*
(ii) $N \cap F = M \cap F$.

Example 2. Recalling Π_{diag} from Example 1, we note that it has a unique $\langle \{ab_1\} > \{ab_2\} > \{ab_3\}, V, \emptyset\rangle$-minimal model $M = \{ab_3, b\}$, that is, ab_1 is falsified first, then ab_2, and finally ab_3, but the last minimization fails as it holds $\Pi_{\mathrm{diag}} \cup \{\neg ab_1, \neg ab_2\} \models ab_3$. ■

Our next objective is to characterize $\langle P_1 > \cdots > P_k, V, F\rangle$-minimality for a model M of a PDLP Π in terms of propositional satisfiability. The idea is to check whether the set of disjunctive rules $\mathrm{Tr}_{\mathrm{U}}(\Pi, P_1 > \cdots > P_k, F, M)$ defined as

$$\begin{array}{l}\{(A \setminus F) \leftarrow (B \setminus F) \mid A \leftarrow B \in \Pi, M \not\models \bigvee(A \cap F), \text{and } M \models B \cap F\} \cup \\ \{e_0 \leftarrow\} \cup \{\bot \leftarrow e_k\} \cup \\ \bigcup_{i=1}^{k}\{e_i \leftarrow (P_i \cap M) \cup \{e_{i-1}\}\} \cup \bigcup_{i=1}^{k}\{\bot \leftarrow a, e_{i-1} \mid a \in P_i \setminus M\},\end{array} \tag{3}$$

is *unsatisfiable in the classical sense*. The new atoms $e_1, \ldots, e_k$ in $\mathrm{Tr}_{\mathrm{U}}(\Pi, P_1 > \cdots > P_k, F, M)$ correspond to the strata in $P_1 > \cdots > P_k$. The intuitive reading of e_i is that the truth values of all atoms in $P_1 \cup \ldots \cup P_i$ coincide with those assigned by M.

Lemma 1. *Given a positive DLP Π, a model $M \subseteq \mathrm{At}(\Pi)$ is $\langle P_1 > \cdots > P_k, V, F\rangle$-minimal if and only if $\mathrm{Tr}_{\mathrm{U}}(\Pi, P_1 > \cdots > P_k, F, M)$ is unsatisfiable.*

Proof sketch. ($\Longrightarrow$) Assume that there is $N \subseteq (\mathrm{At}(\Pi) \setminus F) \cup \{e_i \mid 0 \leq i \leq k\}$ such that $N \models \mathrm{Tr}_{\mathrm{U}}(\Pi, P_1 > \cdots > P_k, F, M)$, that is, $N \models (A \setminus F) \leftarrow (B \setminus F)$ for each $A \leftarrow B \in \Pi$ such that $M \not\models \bigvee(A \cap F)$ and $M \models B \cap F$; $N \models e_0$; and $N \not\models e_k$. Furthermore, there is $1 \leq i \leq k$ such that $N \models e_j$ and $N \models P_j \cap M$ for each $j < i$, $N \not\models a$ for each $a \in P_j \setminus M$ for $j \leq i$, and $N \not\models P_i \cap M$. It is easy to see that $M' = (N \cap \mathrm{At}(\Pi)) \cup (M \cap F)$ is a counter-example for the $\langle P_1 > \cdots > P_k, V, F\rangle$-minimality of M, as $M' \models \Pi$, $M' \cap (P_1 \cup \cdots \cup P_{i-1}) = M \cap (P_1 \cup \cdots \cup P_{i-1})$ and $M' \cap P_i \subset M \cap P_i$.

($\Longleftarrow$) Assume that $M \models \Pi$ and M is not $\langle P_1 > \cdots > P_k, V, F\rangle$-minimal, that is, there is $N \models \Pi$ such that for some $1 \leq i \leq k$, $N \cap (P_1 \cup \cdots \cup P_{i-1}) = M \cap (P_1 \cup \cdots \cup P_{i-1})$ and $N \cap P_i \subset M \cap P_i$. We define $N' = (N \setminus F) \cup \{e_j \mid 0 \leq j \leq i-1\}$. It is straightforward to verify that $N' \models \mathrm{Tr}_{\mathrm{U}}(\Pi, P_1 > \cdots > P_k, F, M)$. □

4 Translation-Based Approach to Prioritized Circumscription

In [9] we present a translation function which enables the removal of varying atoms from a PDLP Π in a *faithful* way, that is , the $\langle P, V, F\rangle$-minimal models M of Π and the stable models N of its translation are in a bijective relationship such that $M = N \cap \mathrm{At}(\Pi)$ holds for each pair of models. In [1] we propose a way to generalize this method to the case of prioritized circumscription. In this section, we present the details of the translation and justify the correctness of the method in general.

The translation $\mathrm{Tr}_{\mathrm{circ2dlp}}(\Pi, P_1 > \cdots > P_k, V, F)$ consists of two parts. Instead of presenting the parts as regular DLPs, we exploit the theory from [13] and describe the parts as *DLP-modules* or *DLP-functions* with input/output interface. Syntactically, a DLP-module is a triple $\langle \Pi, I, O\rangle$[2] where Π is a set of disjunctive rules and I and O are disjoint sets of atoms such that $\mathrm{At}(\Pi) = I \cup O$, and all occurrences of *input atoms* $a \in I$ are in the bodies of rules in Π. The idea is to keep the interpretation of input atoms fixed, that is, an interpretation $M \subseteq \mathrm{At}(\Pi)$ is a stable model of a DLP-module $\langle \Pi, I, O\rangle$ if and only if $M \in \mathrm{MM}_{O,\emptyset,I}(\Pi^M)$. One may notice that for DLP-modules with $I = \emptyset$, this definition results in the standard stable model semantics of DLPs. The *composition* or *join* of two DLP-modules $\langle \Pi_1, I_1, O_1\rangle$ and $\langle \Pi_2, I_2, O_2\rangle$ is defined syntactically as $\langle \Pi_1, I_1, O_1\rangle \sqcup \langle \Pi_2, I_2, O_2\rangle = \langle \Pi_1 \cup \Pi_2, (I_1 \cup I_2) \setminus (O_1 \cup O_2), O_1 \cup O_2\rangle$. The overall translation $\mathrm{Tr}_{\mathrm{circ2dlp}}(\Pi, P_1 > \cdots > P_k, V, F)$ is a join of two DLP-modules:

$$\mathrm{Tr}_{\mathrm{gen}}(\Pi, P_1 > \cdots > P_k, V, F) \sqcup \mathrm{Tr}_{\mathrm{min}}(\Pi, P_1 > \cdots > P_k, V, F). \tag{4}$$

For the sake of brevity, we omit the sets $P_1, \ldots, P_k$, V, and F when they are clear from the context.

Next, we describe the structure of modules $\mathrm{Tr}_{\mathrm{gen}}(\Pi)$ and $\mathrm{Tr}_{\mathrm{min}}(\Pi)$ in more detail. They involve a number of atoms which are new to Π:

[2] In [13] a more general setting is introduced in which a DLP-function may contain *hidden atoms*, which are local to the module in question. See [13] for more details, for example, of the conditions under which the composition of two DLP-modules is defined.

Module	:	$\mathrm{Tr}_{\mathrm{gen}}(\Pi, P_1 > \cdots > P_k, V, F)$
Input	:	$\emptyset$
Output	:	$\mathrm{At}(\Pi) \cup \overline{\mathrm{At}(\Pi)}$
1	:	$a \leftarrow {\sim}\overline{a}$ for each $a \in V \cup F$
2	:	$\overline{a} \leftarrow {\sim}a$ for each $a \in \mathrm{At}(\Pi)$
3	:	$(A \cap P) \leftarrow (B \cap P), {\sim}(A \setminus P), {\sim}\overline{(B \setminus P)}$ for each $A \leftarrow B \in \Pi$

Fig. 1. Module $\mathrm{Tr}_{\mathrm{min}}(\Pi, P_1 > \cdots > P_k, V, F)$ from Definition 4 with $P = P_1 \cup \cdots \cup P_k = \mathrm{At}(\Pi) \setminus (V \cup F)$

- An atom $\overline{a}$ denoting that a is false is introduced for each $a \in \mathrm{At}(\Pi)$.
- For each atom $a \in \mathrm{At}(\Pi)$, a renamed copy a^* of a is created in order to formulate the test for $\langle P_1 > \cdots > P_k, V, F\rangle$-minimality. Thus, the meaning of a^* is that a is true in a potential counter-model for $\langle P_1 > \cdots > P_k, V, F\rangle$-minimality.
- An atom e_i is introduced for each priority class P_i. The intuitive reading of e_i is the same as in (3), that is, the truth values of all atoms in $P_1 \cup \ldots \cup P_i$ in the potential counter-example coincide with those assigned by the model candidate.
- A renamed copy a^{d} is introduced for each atom $a \in P_1 \cup \cdots \cup P_k$. The meaning of a^{d} is that the model candidate and the potential counter-example for its $\langle P_1 > \cdots > P_k, V, F\rangle$-minimality assign *different* truth values to atom $a \in P_i$.
- Finally, an atom u is introduced to denote the unsatisfiability of (3).

We use shorthands $\overline{A} = \{\overline{a} \mid a \in A\}$ and $A^* = \{a^* \mid a \in A\}$ for any $A \subseteq \mathrm{At}(\Pi)$. Likewise A^{d} denotes $\{a^{\mathrm{d}} \mid a \in A\}$ for any $A \subseteq P_1 \cup \cdots \cup P_k$. The modules $\mathrm{Tr}_{\mathrm{gen}}(\Pi)$ and $\mathrm{Tr}_{\mathrm{min}}(\Pi)$ forming the join (4) are as follows.

Definition 4. *Let Π be a PDLP subject to a prioritized circumscription* $\mathrm{Circ}(\Pi, P_1 > \cdots > P_k, V, F)$. *The translation* $\mathrm{Tr}_{\mathrm{circ2dlp}}(\Pi) = \langle \Pi_{\mathrm{g}} \cup \Pi_{\mathrm{m}}, \emptyset, O_{\mathrm{g}} \cup O_{\mathrm{m}}\rangle$ *is the join of the DLP-modules* $\mathrm{Tr}_{\mathrm{gen}}(\Pi) = \langle \Pi_{\mathrm{g}}, \emptyset, O_{\mathrm{g}}\rangle$ *presented in Fig. 1 and* $\mathrm{Tr}_{\mathrm{min}}(\Pi) = \langle \Pi_{\mathrm{m}}, O_{\mathrm{g}}, O_{\mathrm{m}}\rangle$ *presented in Fig. 2.*

The module $\mathrm{Tr}_{\mathrm{gen}}(\Pi)$ takes no input but it produces a model candidate for circumscription as its output. The rules in lines 1–2 choose truth values for atoms in $V \cup F$ and define the complementary atoms $\overline{a}$ for all atoms $a \in \mathrm{At}(\Pi)$. The rules in the third line make sure that Π is satisfied. As a matter of optimization, the satisfaction of rules is focussed on the atoms subject to minimization, that is, those in $P = P_1 \cup \ldots \cup P_k$.

An actual input for $\mathrm{Tr}_{\mathrm{min}}(\Pi)$ is a candidate for a $\langle P_1 > \cdots > P_k, V, F\rangle$-minimal model of Π but represented as a set $M \cup \overline{(\mathrm{At}(\Pi) \setminus M)}$ instead of $M \subseteq \mathrm{At}(\Pi)$. Rules in line 4 create a renamed copy of Π to check the $\langle P_1 > \cdots > P_k, V, F\rangle$-minimality of M. The atoms in F are not renamed to maintain the semantics of fixed atoms. The rules in lines 5–7 are activated for each set P_i only if e_j is true for all $j < i$. Each rule in line 5 captures a rule $e_i \leftarrow (P_i \cap M) \cup \{e_{i-1}\}$ from (3). This rule *depends dynamically* on M and effectively states, using the disjunction $\bigvee P_i^{\mathrm{d}}$, the *falsity* of at least one atom a that is both subject to minimization with priority i ($a \in P_i$) and true in M ($a \in M$).

Module	:	$\mathrm{Tr_{min}}(\Pi, P_1 > \cdots > P_k, V, F)$
Input	:	$\mathrm{At}(\Pi) \cup \overline{\mathrm{At}(\Pi)}$
Output	:	$\mathrm{At}(\Pi)^* \cup P_1^{\mathrm{d}} \cup \cdots \cup P_k^{\mathrm{d}} \cup \{e_i \mid 0 \leq i \leq k\} \cup \{u\}$
4	:	$(A \setminus F)^* \cup \{u\} \leftarrow (B \setminus F)^*, {\sim}(A \cap F), {\sim}\overline{(B \cap F)}$ for each $A \leftarrow B \in \Pi$
5	:	$P_i^{\mathrm{d}} \cup \{e_i, u\} \leftarrow e_{i-1}$ for each P_i
6	:	$u \leftarrow a^{\mathrm{d}}, {\sim}a, e_{i-1}$ and $u \leftarrow a^*, {\sim}a, e_{i-1}$ for each P_i and $a \in P_i$
7	:	$u \leftarrow a^{\mathrm{d}}, a^*, {\sim}\overline{a}, e_{i-1}$ and $u \vee a^{\mathrm{d}} \vee a^* \leftarrow {\sim}\overline{a}, e_{i-1}$ for each P_i and $a \in P_i$
8	:	$a^* \leftarrow u$ for each $a \in \mathrm{At}(\Pi)$
9	:	$a^{\mathrm{d}} \leftarrow u$ for each $a \in P_1 \cup \cdots \cup P_k$
10	:	$e_i \leftarrow u$ for $0 \leq i \leq k$
11	:	$u \leftarrow e_k$; $e_0 \vee u \leftarrow$; and $\bot \leftarrow {\sim}u$

Fig. 2. Module $\mathrm{Tr_{min}}(\Pi, P_1 > \cdots > P_k, V, F)$ from Definition 4

The rules in line 6 cover the case that a is false in M ($a \in P_i \setminus M$). Conforming to (3), both a^{d} and a^* are implicitly assigned to false, as they imply u. Otherwise, a is true in M which activates the rules in line 7, enforcing a^{d} equivalent to the negation of a^*. The net effect of the rules in lines 5–7 is that any *potential counter-model* $N \models \Pi$ for the $\langle P_1 > \cdots > P_k, V, F\rangle$-minimality of M, expressed in $(\mathrm{At}(\Pi) \setminus F)^* \cup (\mathrm{At}(\Pi) \cap F)$ instead of $\mathrm{At}(\Pi)$, must satisfy conditions (i) and (ii) from Definition 3. The rules in lines 8–11 are directly related to the unsatisfiability check which effectively proves that counter-models like N above do not exist.

Finally, we need to justify the *faithfulness* of the translation $\mathrm{Tr_{circ2dlp}}(\Pi)$, that is, to show that the $\langle P_1 > \cdots > P_k, V, F\rangle$-minimal models of a PDLP Π are in a bijective relationship with the stable models of $\mathrm{Tr_{circ2dlp}}(\Pi)$. A key observation is that the stable model semantics is *compositional* for the join of DLP-modules, that is, stable models for $\mathrm{Tr_{circ2dlp}}(\Pi)$ can be computed for one submodule at a time. The following lemma is a direct consequence of the *module theorem* [13, Theorem 1].

Lemma 2. *Let* $M \subseteq \mathrm{At}(\mathrm{Tr_{gen}}(\Pi))$ *and* $N \subseteq \mathrm{At}(\mathrm{Tr_{min}}(\Pi)) \setminus (\mathrm{At}(\Pi) \cup \overline{\mathrm{At}(\Pi)})$. *Then* $M \in \mathrm{SM}(\mathrm{Tr_{gen}}(\Pi))$ *and* $M \cup N \in \mathrm{SM}(\mathrm{Tr_{min}}(\Pi))$ *if and only if* $M \cup N \in \mathrm{SM}(\mathrm{Tr_{circ2dlp}}(\Pi))$.

The classical models of Π and the stable models of $\mathrm{Tr_{gen}}(\Pi)$ are in bijective correspondence.

Lemma 3. *For a PDLP* Π *subject to a prioritized circumscription* $\mathrm{Circ}(\Pi, P_1 > \cdots > P_k, V, F)$ *and* $M \subseteq \mathrm{At}(\Pi)$, $M \models \Pi$ *if and only if* $M' = M \cup \overline{(\mathrm{At}(\Pi) \setminus M)} \in \mathrm{SM}(\mathrm{Tr_{gen}}(\Pi))$.

Next, we characterize the connection between the existence of stable models for $\mathrm{Tr_{min}}(\Pi)$ and the unsatisfiability of the translation $\mathrm{Tr_U}(\Pi, P_1 > \cdots > P_k, F, M)$ in (3).

Lemma 4. *Let Π be a PDLP subject to a prioritized circumscription* $\mathrm{Circ}(\Pi, P_1 > \cdots > P_k, V, F)$, *$M \subseteq \mathrm{At}(\Pi)$ a model of Π, and $M' = M \cup \overline{(\mathrm{At}(\Pi) \setminus M)}$.*

(i) If $\mathrm{Tr}_{\mathrm{min}}(\Pi)$ has a stable model N such that $N \cap (\mathrm{At}(\Pi) \cup \overline{\mathrm{At}(\Pi)}) = M'$, then $N = M' \cup P^{\mathrm{d}} \cup E \cup \{u\} \cup \mathrm{At}(\Pi)^*$ where $P = P_1 \cup \ldots \cup P_k$ and $E = \{e_i \mid 0 \leq i \leq k\}$.

(ii) The set of disjunctive rules $\mathrm{Tr}_{\mathrm{U}}(\Pi, P_1 > \cdots > P_k, F, M)$ is unsatisfiable if and only if there is $N \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{min}}(\Pi))$ such that $N \cap (\mathrm{At}(\Pi) \cup \overline{\mathrm{At}(\Pi)}) = M'$.

The proof of Lemma 4 is similar to the proof of [9, Proposition 2] as the same technique is used to encode propositional unsatisfiability check with the primitives of DLPs [14].

The correctness of the translation $\mathrm{Tr}_{\mathrm{circ2dlp}}(\Pi)$ now follows from Lemmas 1–4.

Theorem 1. *Given a PDLP Π, M is a $\langle P_1 > \cdots > P_k, V, F\rangle$-minimal model of Π if and only if $N \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{circ2dlp}}(\Pi))$ where $N \cap (\mathrm{At}(\Pi) \cup \overline{\mathrm{At}(\Pi)}) = M \cup \overline{(\mathrm{At}(\Pi) \setminus M)}$.*

Proof sketch. ($\Longrightarrow$) Assume that M is a $\langle P_1 > \cdots > P_k, V, F\rangle$-minimal model of Π. Since $M \models \Pi$, we have $N = M \cup \overline{(\mathrm{At}(\Pi) \setminus M)} \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{gen}}(\Pi))$ by Lemma 3. Since M is $\langle P_1 > \cdots > P_k, V, F\rangle$-minimal, $\mathrm{Tr}_{\mathrm{U}}(\Pi, P_1 > \cdots > P_k, F, M)$ is unsatisfiable by Lemma 1. By Lemma 4 (ii), there is $N' \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{min}}(\Pi))$ such that $N' \cap (\mathrm{At}(\Pi) \cup \overline{\mathrm{At}(\Pi)}) = N$, and by Lemma 4 (i) $N' = N \cup P_1^{\mathrm{d}} \cup \cdots \cup P_k^{\mathrm{d}} \cup \{e_i \mid 0 \leq i \leq k\} \cup \{u\} \cup \mathrm{At}(\Pi)^*$. Furthermore, by Lemma 2, $N' \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{circ2dlp}}(\Pi))$. ($\Longleftarrow$) Assume $M \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{circ2dlp}}(\Pi))$. By Lemma 2, $M' = M \cap \mathrm{At}(\mathrm{Tr}_{\mathrm{gen}}(\Pi)) \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{gen}}(\Pi))$ and $M \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{min}}(\Pi))$. By Lemma 3, we have $N = M' \cap \mathrm{At}(\Pi) \models \Pi$. By Lemma 4 (ii), $\mathrm{Tr}_{\mathrm{U}}(\Pi, P_1 > \cdots > P_k, F, N)$ is unsatisfiable, and finally, by Lemma 1, N is a $\langle P_1 > \cdots > P_k, V, F\rangle$-minimal model of Π. □

The following example illustrates the use of the translation for computing prioritized circumscription.

Example 3. Recall the positive DLP Π_{diag} in Example 1. In order to compute the $\langle\{ab_1\} > \{ab_2\} > \{ab_3\}, \{a, b, c, d\}, \emptyset\rangle$-minimal models of Π_{diag}, we consider the translation $\mathrm{Tr}_{\mathrm{circ2dlp}}(\Pi_{\mathrm{diag}}, \{ab_1\} > \{ab_2\} > \{ab_3\}, \{a, b, c, d\}, \emptyset)$ which is the join of modules $\mathrm{Tr}_{\mathrm{gen}}(\Pi_{\mathrm{diag}})$ and $\mathrm{Tr}_{\mathrm{min}}(\Pi_{\mathrm{diag}})$ presented in Figure 3. The translation $\mathrm{Tr}_{\mathrm{circ2dlp}}(\Pi_{\mathrm{diag}})$ has a unique stable model

$$N = \{\overline{a}, b, \overline{c}, \overline{d}, \overline{ab_1}, \overline{ab_2}, ab_3\} \cup \mathrm{At}(\Pi_{\mathrm{diag}})^* \cup \{ab_1^{\mathrm{d}}, ab_2^{\mathrm{d}}, ab_3^{\mathrm{d}}, e_0, e_1, e_2, e_3, u\}.$$

By Theorem 1, $N \cap \mathrm{At}(\Pi_{\mathrm{diag}}) = \{ab_3, b\} = M$ is the unique $\langle\{ab_1\} > \{ab_2\} > \{ab_3\}, \{a, b, c, d\}, \emptyset\rangle$-minimal model of Π_{diag} as already discussed in Example 2. ■

5 Experiments

We use the problem of finding Reiter-style minimal diagnoses [6] for digital circuits encoded as parallel/prioritized circumscription as a benchmark. The circuits are generated as follows. First a random tree is generated. The leaves of the tree, that is, the inputs of the circuit, are assigned random Boolean values. The intermediate nodes are assigned random logical operations which correspond to the gates of the circuit. The gate at the

Module :	$\mathrm{Tr}_{\mathrm{gen}}(\Pi_{\mathrm{diag}}, \{ab_1\} > \{ab_2\} > \{ab_3\}, \{a,b,c,d\}, \emptyset)$
Input :	$\emptyset$
Output :	$\mathrm{At}(\Pi_{\mathrm{diag}}) \cup \overline{\mathrm{At}(\Pi_{\mathrm{diag}})}$
1 :	$a \leftarrow \sim\overline{a}.\ \ b \leftarrow \sim\overline{b}.\ \ c \leftarrow \sim\overline{c}.\ \ d \leftarrow \sim\overline{d}$
2 :	$\overline{a} \leftarrow \sim a.\ \ \overline{b} \leftarrow \sim b.\ \ \overline{c} \leftarrow \sim c.\ \ \overline{d} \leftarrow \sim d.\ \ \overline{ab_1} \leftarrow \sim ab_1.\ \ \overline{ab_2} \leftarrow \sim ab_2.$ $\overline{ab_3} \leftarrow \sim ab_3$
3 :	$ab_1 \leftarrow \sim a, \sim b.\ \ ab_1 \leftarrow \sim\overline{a}, \sim\overline{b}.\ \ ab_2 \leftarrow \sim b, \sim c.\ \ ab_2 \leftarrow \sim\overline{b}, \sim\overline{c}.$ $ab_3 \leftarrow \sim c, \sim d.\ \ ab_3 \leftarrow \sim\overline{c}, \sim\overline{d}.\ \ \bot \leftarrow \sim\overline{a}.\ \ \bot \leftarrow \sim\overline{d}$
Module :	$\mathrm{Tr}_{\mathrm{min}}(\Pi_{\mathrm{diag}}, \{ab_1\} > \{ab_2\} > \{ab_3\}, \{a,b,c,d\}, \emptyset)$
Input :	$\mathrm{At}(\Pi_{\mathrm{diag}}) \cup \overline{\mathrm{At}(\Pi_{\mathrm{diag}})}$
Output :	$\mathrm{At}(\Pi_{\mathrm{diag}})^* \cup \{ab_1^{\mathrm{d}}, ab_2^{\mathrm{d}}, ab_3^{\mathrm{d}}, e_0, e_1, e_2, e_3, u\}$
4 :	$a^* \vee b^* \vee ab_1^* \vee u.\ \ ab_1^* \vee u \leftarrow a^*, b^*.\ \ b^* \vee c^* \vee ab_2^* \vee u.\ \ ab_2^* \vee u \leftarrow b^*, c^*.$ $c^* \vee d^* \vee ab_3^* \vee u.\ \ ab_3^* \vee u \leftarrow c^*, d^*.\ \ u \leftarrow a^*.\ \ u \leftarrow d^*$
5 :	$ab_1^{\mathrm{d}} \vee e_1 \vee u \leftarrow e_0.\ \ ab_2^{\mathrm{d}} \vee e_2 \vee u \leftarrow e_1.\ \ ab_3^{\mathrm{d}} \vee e_3 \vee u \leftarrow e_2$
6 :	$u \leftarrow ab_1^{\mathrm{d}}, \sim ab_1, e_0.\ \ u \leftarrow ab_1^*, \sim ab_1, e_0.\ \ u \leftarrow ab_2^{\mathrm{d}}, \sim ab_2, e_1.$ $u \leftarrow ab_2^*, \sim ab_2, e_1.\ \ u \leftarrow ab_3^{\mathrm{d}}, \sim ab_3, e_2.\ \ u \leftarrow ab_3^*, \sim ab_3, e_2$
7 :	$u \leftarrow ab_1^{\mathrm{d}}, ab_1^*, \sim\overline{ab_1}, e_0.\ \ u \vee ab_1^{\mathrm{d}} \vee ab_1^* \leftarrow \sim\overline{ab_1}, e_0.$ $u \leftarrow ab_2^{\mathrm{d}}, ab_2^*, \sim\overline{ab_2}, e_1.\ \ u \vee ab_2^{\mathrm{d}} \vee ab_2^* \leftarrow \sim\overline{ab_2}, e_1.$ $u \leftarrow ab_3^{\mathrm{d}}, ab_3^*, \sim\overline{ab_3}, e_2.\ \ u \vee ab_3^{\mathrm{d}} \vee ab_3^* \leftarrow \sim\overline{ab_3}, e_2$
8 :	$a^* \leftarrow u.\ \ b^* \leftarrow u.\ \ c^* \leftarrow u.\ \ d^* \leftarrow u.\ \ ab_1^* \leftarrow u.\ \ ab_2^* \leftarrow u.\ \ ab_3^* \leftarrow u$
9 :	$ab_1^{\mathrm{d}} \leftarrow u.\ \ ab_2^{\mathrm{d}} \leftarrow u.\ \ ab_3^{\mathrm{d}} \leftarrow u$
10 :	$e_0 \leftarrow u.\ \ e_1 \leftarrow u.\ \ e_2 \leftarrow u.\ \ e_3 \leftarrow u$
11 :	$u \leftarrow e_3.\ \ e_0 \vee u.\ \ \bot \leftarrow \sim u$

Fig. 3. The modules for translation $\mathrm{Tr}_{\mathrm{circ2dlp}}(\Pi_{\mathrm{diag}})$ in Example 3

root node produces the output for the entire circuit. Its value is calculated and flipped to obtain faulty behavior for the circuit. The $\langle\{\mathsf{ab}_i \mid i \leq N\}, \{\mathsf{high}_i \mid i \leq N\}, \emptyset\rangle$-minimal models of the resulting program correspond to minimal diagnoses, where N is the number of nodes in the tree forming the circuit. For each value of N we select priorities for the atoms $\mathtt{ab}_j$ according to the following scheme. An atom $\mathtt{ab}_j$ is given priority i, if $(i-1)\cdot\lfloor N/k\rfloor + 1 \leq j \leq i \cdot \lfloor N/k\rfloor$, where k is the number of priority classes.

The tool CIRC2DLP (v. 2.1)[3] implements the translation $\mathrm{Tr}_{\mathrm{circ2dlp}}$ described in Section 4. We compare the performance of CIRC2DLP with our previous translator PRIO_CIRC2DLP [10] which implements Lifschitz' scheme (2). We use GNT (v. 2.1) and DLV (2006-07-14) for the computation of stable models. We also compare the performance of our translation-based approach with that of CIRCUM2 system [12]. The

[3] See `http://www.tcs.hut.fi/Software/circ2dlp/` for binaries and benchmarks.

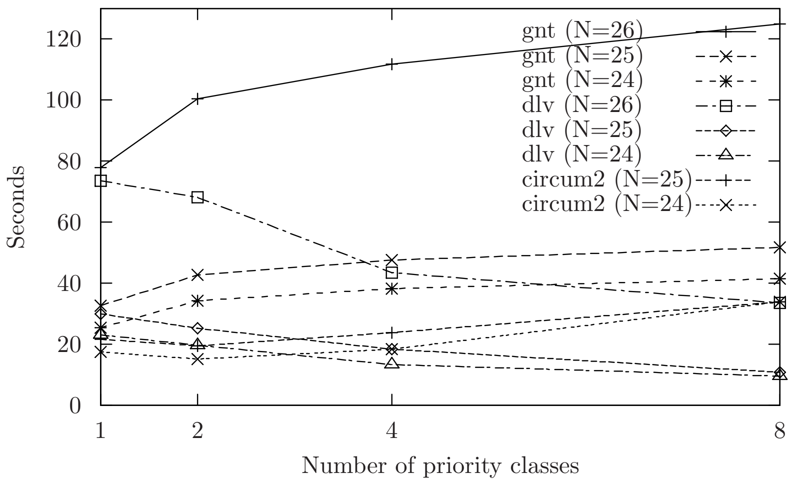

Fig. 4. The averages of running times for computing *all* minimal diagnoses for faulty digital circuits for $N = 24, 25, 26$ nodes with the numbers of priority classes $k = 1, 2, 4, 8$ for minimization

measured running time for CIRC2DLP and PRIO_CIRC2DLP is the sum of the translation time and the time of finding the stable models using GNT/DLV. The translation times are negligible, however. For CIRCUM2 the measured running time is the duration of the search for models of circumscription using the tool with DLV as its back-end. We report the sum of user and system time of `/usr/bin/time`. All the tests were run under Linux on a 1.7GHz AMD Athlon XP 2000+ with a timeout 1800 seconds and a memory limit 512MB.

First, we compare the performances of CIRC2DLP and PRIO_CIRC2DLP with an instance with $N = 15$ nodes in the circuit and use GNT for the computation of stable models. The running times for $k = 1, 2, 3, 4$ priorities were 0.29, 0.38, 0.39 and 0.43 seconds for CIRC2DLP, and respectively 0.30, 44.3, 88.8 and 127.25 seconds for PRIO_CIRC2DLP. The performance of PRIO_CIRC2DLP is poor for $k > 1$ even with a very small circuit with $N = 15$ and CIRC2DLP shows a promising improvement in the performance.

Next, we use CIRC2DLP with GNT and DLV as back-ends to see whether the choice of a solver has an effect on the running times. We also compare the performance of CIRC2DLP to that of CIRCUM2. To see how different approaches scale on the number of priority classes $k = 1, 2, 4, 8$ we generate randomly 20 circuits for each of the values $N = 24, 25, 26$. The average running times from this experiment are shown in Figure 4.[4] First, observe that in contrast to the experimental results in [10] computing models with DLV is faster than with GNT. This illustrates the flexibility of the translation-based method compared to developing a specialized solver as one can directly benefit from solver development. For values $k = 1, 2$ CIRCUM2 is slightly

[4] The average running times for $N = 26$ cannot be reported for CIRCUM2, since it was not able to solve all instances within the memory limit.

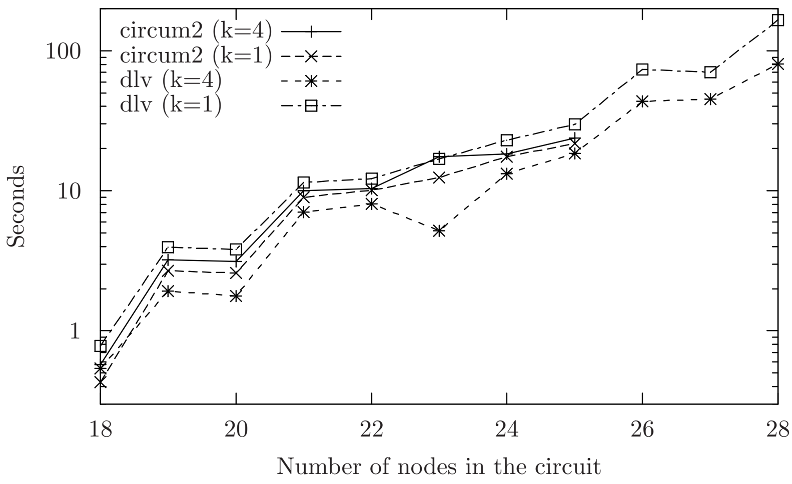

Fig. 5. The averages of running times for computing *all* minimal diagnoses for faulty digital circuits for $N = 18, \ldots, 28$ nodes with $k = 1, 4$ priorities for minimization

faster than CIRC2DLP, but CIRCUM2 consumed over 512MB of memory with some rather moderately sized instances with $N = 26$. For $k = 4, 8$, CIRC2DLP+DLV becomes faster than CIRCUM2. Also, we note that the average running times of CIRC2DLP+DLV decrease when the number of priorities k increases. This reflects the fact that eventually, that is, when $k = |\mathrm{At}(\Pi) \setminus (V \cup F)|$, it becomes easier to decide $\langle \{a_1\} > \cdots > \{a_k\}, V, F\rangle$-minimality using a total order of atoms $a_1, \ldots, a_k$ under minimization [15].

Finally, we study how the performances of CIRC2DLP+DLV[5] and CIRCUM2 scale up when the number of nodes N grows. We consider $k = 1$ and $k = 4$ priorities for minimization, and generate randomly 20 circuits with $N = 18, \ldots, 28$ nodes. The average running times are presented in Figure 5. Again, CIRCUM2 could not solve all instances for $N \geq 26$ without exceeding the memory limit. Both systems performed similarly for values $N = 18, \ldots, 25$, but using CIRC2DLP we could easily compute models for the respective circumscriptions up to $N = 28$.

6 Discussion

We present a transformation from prioritized circumscription to DLPs. The translation function $\mathrm{Tr}_{\mathrm{circ2dlp}}$ generalizes the one designed for parallel circumscription [9] and it has a distinctive combination of features: (i) arbitrary propositional theories Π subject to prioritized circumscription are covered, (ii) the translation $\mathrm{Tr}_{\mathrm{circ2dlp}}(\Pi, P_1 > \cdots >$

[5] As computing stable models with DLV is faster than with GNT, the choice of DLV as the back-end reflects the best performance of our approach. Notice, however, that the results of the previous experiment show that even with GNT as back-end, CIRC2DLP can handle circuits with at least $N = 26$ nodes.

$P_k, V, F)$ can be produced in linear time and space before computing any models for it, (iii) the models of $\mathrm{Circ}(\Pi, P_1 > \cdots > P_k, V, F)$ and the stable models of its translation are in a bijective relationship, (iv) the signature $\mathrm{At}(\Pi)$ is preserved under $\mathrm{Tr}_{\mathrm{circ2dlp}}$, and (v) there is no need for incremental updating.

In contrast, all previous approaches lack some of these features. De Kleer and Konolige [16] present the basic technique for eliminating fixed predicates. The case of varying predicates is addressed by Cadoli et al. [17] but a query-based equivalence rather than an exact correspondence of models is of their interest. In addition to these general results, a number of attempts to reduce parallel/prioritized circumscription into logic programming have been made. Gelfond and Lifschitz [18] address prioritized circumscription but their translation scheme is amenable to *stratified* circumscriptive theories only. The translation of parallel circumscription presented by Sakama and Inoue [19] is based on *characteristic clauses* resulting in exponential space and time complexity in the worst case. In [20], the same authors embed prioritized circumscription into *prioritized logic programming* based on a different semantics. Lee and Lin [21] characterize parallel circumscription in terms of *loop formulas* and exploit them to obtain an embedding in disjunctive logic programming. However, the number of loops can be exponential in the worst case. Thus, it remains open whether an efficient translation is feasible in general using their approach. Wakaki and Inoue [11] concentrate on prioritized circumscription and design a two-phase procedure for the computation of minimal models. The first phase generates model candidates which are then tested for minimality in the sense of prioritized circumscription. Both the model generator and the tester are represented as *separate* disjunctive logic programs. There is an implementation of the procedure, named CIRCUM1, but it is rather inefficient since all model candidates are computed first. Wakaki and Tomita [12] improve the procedure by Wakaki and Inoue [11] and integrate the generating and testing programs into one in analogy to our approach [9]. However, this is not a one-shot transformation because the answer sets of the generating program have to be computed and counted before the testing part can be created. The resulting implementation, CIRCUM2 was used as one of the reference systems in our experiments.

The experiments reported in Section 5 suggest that our translation-based approach compares favorably with CIRCUM2. Due to linearity of the transformation, CIRC2DLP with a disjunctive solver as its back-end needs far less memory than CIRCUM2 and thus it eventually scales up better as demonstrated in our experiments. The performance of CIRC2DLP with disjunctive solvers is encouraging—suggesting that there is no need to develop dedicated solvers for prioritized circumscription. Furthermore, we can take full advantage of the ongoing development of disjunctive solvers.

Our results enable the use of prioritized circumscription as a primitive in disjunctive logic programming. Consequently, we expect that more concise encodings can be devised in applications like model-based diagnosis [6] formalized in the experiments of Section 5. In fact, we can now view prioritized circumscription as syntactic sugar as it can be translated away using $\mathrm{Tr}_{\mathrm{circ2dlp}}$. However, it may be wise to store the original representation, rather than the translation, for easier maintainability. A further goal is the generalization of stable models with prioritized minimization of models. In fact, the design of CIRC2DLP already includes support for negative body literals in rules.

This readily enables the computation of $\langle P_1 > \cdots > P_k, V, F\rangle$*-stable models* M of an arbitrary (not just positive) DLP Π based on the reduct Π^M.

References

1. Oikarinen, E., Janhunen, T.: A linear transformation from prioritized circumscription to disjunctive logic programming. In: Dahl, V., Niemelä, I. (eds.) ICLP 2007. LNCS, vol. 4670, pp. 440–441. Springer, Heidelberg (2007)
2. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing 9(3/4), 365–385 (1991)
3. Przymusinski, T.: Stable semantics for disjunctive programs. New Generation Computing 9(3/4), 401–424 (1991)
4. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Transactions on Computational Logic 7(3), 499–562 (2006)
5. Janhunen, T., Niemelä, I., Seipel, D., Simons, P., You, J.H.: Unfolding partiality and disjunctions in stable model semantics. ACM Transactions on Computational Logic 7(1), 1–37 (2006)
6. Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence 32(1), 57–95 (1987)
7. Lifschitz, V.: Computing circumscription. In: Joshi, A.K. (ed.) Proceedings of the 9th International Joint Conference on Artificial Intelligence, Los Angeles, California, August 18–23, 1985, pp. 121–127. Morgan Kaufmann, San Francisco (1985)
8. McCarthy, J.: Circumscription — A form of non-monotonic reasoning. Artificial Intelligence 13(1–2), 27–39 (1980)
9. Janhunen, T., Oikarinen, E.: Capturing parallel circumscription with disjunctive logic programs. In: Alferes, J.J., Leite, J.A. (eds.) JELIA 2004. LNCS (LNAI), vol. 3229, pp. 134–146. Springer, Heidelberg (2004)
10. Oikarinen, E., Janhunen, T.: CIRC2DLP — translating circumscription into disjunctive logic programming. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS (LNAI), vol. 3662, pp. 405–409. Springer, Heidelberg (2005)
11. Wakaki, T., Inoue, K.: Compiling prioritized circumscription into answer set programming. In: Demoen, B., Lifschitz, V. (eds.) ICLP 2004. LNCS, vol. 3132, pp. 356–370. Springer, Heidelberg (2004)
12. Wakaki, T., Tomita, K.: Compiling prioritized circumscription into general disjunctive programs. In: Provetti, A., Son, T.C. (eds.) Proceedings of PREFS 2006: Preferences and their Applications in Logic Programming Systems, August 16, 2006, pp. 1–15 (2006)
13. Janhunen, T., Oikarinen, E., Tompits, H., Woltran, S.: Modularity aspects of disjunctive stable models. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS (LNAI), vol. 4483, pp. 175–187. Springer, Heidelberg (2007)
14. Eiter, T., Gottlob, G.: On the computational cost of disjunctive logic programming: Propositional case. Annals of Mathematics and Artificial Intelligence 15(3–4), 289–323 (1995)
15. Gottlob, G.: The complexity of default reasoning under the stationary fixed point semantics. Information and Computation 121(1), 81–92 (1995)
16. de Kleer, J., Konolige, K.: Eliminating the fixed predicates from a circumscription. Artificial Intelligence 39(3), 391–398 (1989)
17. Cadoli, M., Eiter, T., Gottlob, G.: An efficient method for eliminating varying predicates from a circumscription. Artificial Intelligence 54(2), 397–410 (1992)

18. Gelfond, M., Lifschitz, V.: Compiling circumscriptive theories into logic programs. In: Reinfrank, M., Ginsberg, M.L., de Kleer, J., Sandewall, E. (eds.) Non-Monotonic Reasoning 1988. LNCS, vol. 346, pp. 74–99. Springer, Heidelberg (1988)
19. Sakama, C., Inoue, K.: Embedding circumscriptive theories in general disjunctive programs. In: Marek, V.W., Truszczyński, M., Nerode, A. (eds.) LPNMR 1995. LNCS, vol. 928, pp. 344–357. Springer, Heidelberg (1995)
20. Sakama, C., Inoue, K.: Prioritized logic programming and its application to commonsense reasoning. Artificial Intelligence 123(1–2), 185–222 (2000)
21. Lee, J., Lin, F.: Loop formulas for circumscription. Artificial Intelligence 170(2), 160–185 (2006)