Fundamenta Informaticae XX (2016) 1–29
DOI 10.3233/FI-2015-0000
IOS Press

Answer Set Programming modulo Acyclicity [*]

Jori Bomanson
Aalto University, HIIT

Tomi Janhunen
Aalto University, HIIT

Torsten Schaub[†]
University of Potsdam
INRIA Rennes

Martin Gebser [C]
University of Potsdam

Benjamin Kaufmann
University of Potsdam

Abstract. Acyclicity constraints are prevalent in knowledge representation and applications where acyclic data structures such as DAGs and trees play a role. Recently, such constraints have been considered in the satisfiability modulo theories (SMT) framework, and in this paper we carry out an analogous extension to the answer set programming (ASP) paradigm. The resulting formalism, ASP modulo acyclicity, offers a rich set of primitives to express constraints related to recursive structures. In the technical results of the paper, we relate the new generalization with standard ASP by showing (i) how acyclicity extensions translate into normal rules, (ii) how weight constraint programs can be instrumented by acyclicity extensions to capture stability in analogy to unfounded set checking, and (iii) how the gap between supported and stable models is effectively closed in the presence of such an extension. Moreover, we present an efficient implementation of acyclicity constraints by incorporating a respective propagator into the state-of-the-art ASP solver CLASP. The implementation provides a unique combination of traditional unfounded set checking with acyclicity propagation. In the experimental part, we evaluate the interplay of these orthogonal checks by equipping logic programs with supplementary acyclicity constraints. The performance results show that native support for acyclicity constraints is a worthwhile addition, furnishing a complementary modeling construct in ASP itself as well as effective means for translation-based ASP solving.

[*]This work extends a preliminary workshop paper [1] presented at ASPOCP'15. A short version [2] appeared at LPNMR'15.
[C]Corresponding author
[†]Affiliated with Simon Fraser University, Canada, and IIIS Griffith University, Australia.

1. Introduction

Acyclic data structures such as DAGs and trees occur frequently in applications. For instance, Bayesian [3] and Markov [4, 5] network learning, circuit layout [6], and phylogeny reconstruction [7] are based on respective conditions. When logical formalisms are used for the specification of such structures, dedicated *acyclicity constraints* are called for. Recently, such constraints have been introduced in the *satisfiability modulo theories* (SMT) framework [8] for extending Boolean *satisfiability* (SAT) [9] in terms of graph-theoretic properties [10, 11]. The idea of *satisfiability modulo acyclicity* [12] is to view certain Boolean variables as conditional edges of a graph and to require that the graph remains acyclic under variable assignments. Moreover, the respective theory propagators for acyclicity have been implemented in the contemporary SAT solvers MINISAT [13] and GLUCOSE [14], which offer a promising machinery for solving applications involving acyclicity constraints.

In this paper, we consider acyclicity constraints in the context of *answer set programming* (ASP) [15], featuring rule-based languages for knowledge representation. The languages used in ASP offer primitives to express, e.g., recursive definitions, defaults, and first-order specifications that are effectively Booleanized by contemporary ASP tools before the search for answer sets. In fact, one of the first applications of SAT modulo acyclicity was the implementation of ASP using appropriate translations of logic programs [12] along with SAT solvers extended by explicit acyclicity constraints and propagators. The goal of this paper, however, is to go beyond this idea and to incorporate acyclicity constraints as additional primitives into ASP. Thus, acyclicity constraints become readily available in the context of extended rule types [16] as well as more demanding reasoning tasks like answer set enumeration and optimization. The resulting formalism, coined *ASP modulo acyclicity* in this paper, offers a rich set of primitives to express constraints related to recursive structures. From this perspective, the new extension of ASP provides more than what is available in the SAT modulo acyclicity approach as such.

In the technical results, we are mainly interested in relating the new generalization with standard ASP and a number of results are worked out in this respect. First, we show how acyclicity extensions can be translated away using normal rules in case a back-end solver does not support acyclicity constraints natively. Second, by adapting the translations from ASP to SAT modulo acyclicity [12], we obtain techniques to instrument logic programs with internal acyclicity extensions to capture stability in analogy to unfounded set checking. In this paper, there are particular contributions concerning the formalization of *well-supporting* rules and the generalization of such internal instrumentation to weight rules. Depending on how strict constraints are imposed on well-supporting rules, we obtain two alternative translations of a weight constraint program P, denoted by $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ and $\mathrm{Tr}_{\mathrm{ACYC+}}(P)$ in the paper, both resulting in weight constraint programs (modulo acyclicity). Given that $\mathrm{Tr}_{\mathrm{ACYC}}(P) \subseteq \mathrm{Tr}_{\mathrm{ACYC+}}(P)$, there is a trade-off between the length of the translation and the strength of constraints over well-supporting rules. Third, we make an interesting observation regarding the semantic gap between supported and stable models that effectively disappears in the presence of such instrumentation. This is why $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ and $\mathrm{Tr}_{\mathrm{ACYC+}}(P)$ lend themselves for subsequent translation into other back-end formalisms such as difference logic or integer programming, and further enable the implementation of ASP (modulo acyclicity) with existing solver technology available in neighboring fields. Finally, we present an efficient implementation of acyclicity constraints obtained as an extension to the state-of-the-art ASP solver CLASP [17]. The implementation offers a unique combination of traditional unfounded set [18] checking and acyclicity propagation [10]. To understand the interplay of both reasoning mechanisms in practice, we conduct an experimental evaluation comparing different propagation principles, translations, and back-end solvers.

The rest of this paper is structured as follows. Basic notions of answer set programming are recalled in Section 2. The extension by explicit acyclicity constraints is then worked out in Section 3, characterizing the relationships between ASP modulo acyclicity and standard ASP, as outlined above, and paving the way for solving methods implemented in the contemporary ASP solver CLASP. Section 4 is dedicated to the experimental evaluation of the new extension, considering a variety of back-end solver variants in comparison to state-of-the-art ASP solvers. Finally, the results of the paper are discussed in Section 5.

2. Background

We consider logic programs built from rules of the following forms:

$$a \leftarrow b_1, \ldots, b_n, \texttt{not}\ c_1, \ldots, \texttt{not}\ c_m. \tag{1}$$

$$\{a\} \leftarrow b_1, \ldots, b_n, \texttt{not}\ c_1, \ldots, \texttt{not}\ c_m. \tag{2}$$

$$a \leftarrow k \leq [b_1 = w_1, \ldots, b_n = w_n, \texttt{not}\ c_1 = w_{n+1}, \ldots, \texttt{not}\ c_m = w_{n+m}]. \tag{3}$$

Symbols a, $b_1, \ldots, b_n$, and $c_1, \ldots, c_m$ where $n \geq 0$ and $m \geq 0$ stand for (propositional) *atoms*, k and $w_1, \ldots, w_{n+m}$ for non-negative integers, and not for (default) *negation*. Atoms like b_i and negated atoms like not c_i are called *positive* and *negative literals*, respectively. For a *normal* (1), *choice* (2), or *weight* (3) *rule* r, we denote its *head* atom by $\mathrm{head}(r) = a$ and its *body* by $\mathrm{B}(r)$. By $\mathrm{B}(r)^+ = \{b_1, \ldots, b_n\}$ and $\mathrm{B}(r)^- = \{c_1, \ldots, c_m\}$, we refer to the *positive* and *negative body* atoms of r. When r is a weight rule, the respective sequence of *weighted literals* is denoted by $\mathrm{WL}(r)$, and its restrictions to positive or negative literals are indicated by $\mathrm{WL}(r)^+$ and $\mathrm{WL}(r)^-$. A normal rule r such that $\mathrm{head}(r) \in \mathrm{B}(r)^-$ is called an *integrity constraint*, or *constraint* for short, and we below skip $\mathrm{head}(r)$ and not $\mathrm{head}(r)$ for brevity, where $\mathrm{head}(r)$ is an arbitrary atom occurring in r only. A *weight constraint program* P, or simply a *program*, is a finite set of rules; P is a *choice program* if it consists of normal and choice rules only, and a *positive program* if it involves neither negation nor choice rules.

Given a program P, let $\mathrm{head}(P) = \{\mathrm{head}(r) \mid r \in P\}$ and $\mathrm{At}(P) = \mathrm{head}(P) \cup \bigcup_{r \in P}(\mathrm{B}(r)^+ \cup \mathrm{B}(r)^-)$ denote the sets of head atoms or all atoms, respectively, occurring in P. The *defining rules* of an atom $a \in \mathrm{At}(P)$ are $\mathrm{Def}_P(a) = \{r \in P \mid \mathrm{head}(r) = a\}$. An *interpretation* $I \subseteq \mathrm{At}(P)$ satisfies $\mathrm{B}(r)$ for a normal or choice rule r iff $\mathrm{B}(r)^+ \subseteq I$ and $\mathrm{B}(r)^- \cap I = \emptyset$. The weighted literals of a weight rule r evaluate to $\mathrm{v}_I(\mathrm{WL}(r)) = \sum_{1 \leq i \leq n, b_i \in I} w_i + \sum_{1 \leq i \leq m, c_i \notin I} w_{n+i}$, and I satisfies $\mathrm{B}(r)$ iff $k \leq \mathrm{v}_I(\mathrm{WL}(r))$. For any rule r, we write $I \models \mathrm{B}(r)$ iff I satisfies $\mathrm{B}(r)$, and $I \models r$ iff $I \models \mathrm{B}(r)$ implies $\mathrm{head}(r) \in I$. (A choice rule r is actually satisfied by any interpretation I, also in case that $I \models \mathrm{B}(r)$ but $\mathrm{head}(r) \notin I$, yet we do not apply the notation $I \models r$ to choice rules in the sequel.) The *supporting rules* of P with respect to I are $\mathrm{SR}_P(I) = \{r \in P \mid \mathrm{head}(r) \in I, I \models \mathrm{B}(r)\}$. Moreover, I is a *model* of P, denoted by $I \models P$, iff $I \models r$ for every $r \in P$ such that r is a normal or weight rule. A model I of P is a *supported model* of P when $\mathrm{head}(\mathrm{SR}_P(I)) = I$. Any positive program P possesses a unique *least model*, denoted by $\mathrm{LM}(P)$, which can be computed via the repeated application of the T_P operator defined by $\mathrm{T}_P(I) = \mathrm{head}(\{r \in P \mid I \models \mathrm{B}(r)\})$. Namely, letting $\mathrm{T}_P \uparrow 0 = \emptyset$ and $\mathrm{T}_P \uparrow i+1 = \mathrm{T}_P(\mathrm{T}_P \uparrow i)$ for $i \geq 0$, the least fixpoint of T_P is obtained in a finite number of steps and coincides with $\mathrm{LM}(P)$.

For a normal or choice rule r, $\mathrm{B}(r)^I = \mathrm{B}(r)^+$ denotes the reduct of $\mathrm{B}(r)$ with respect to an interpretation I, and the reduct $\mathrm{B}(r)^I$ for a weight rule r of the form (3) is defined as $\max\{0, k - \mathrm{v}_I(\mathrm{WL}(r)^-)\} \leq$

$\mathrm{WL}(r)^+$. The *reduct* of a program P with respect to an interpretation I is $P^I = \{\mathrm{head}(r) \leftarrow \mathrm{B}(r)^I \mid r \in \mathrm{SR}_P(I)\}$. Then, an interpretation I is a *stable model* of P iff $I \models P$ and $\mathrm{LM}(P^I) = I$. While any stable model of P is a supported model of P as well, the converse does not hold in general. However, the following concept provides a tighter notion of support achieving such a correspondence.

Definition 2.1. A model $I \subseteq \mathrm{At}(P)$ of a program P is *well-supported* by a set $R \subseteq \mathrm{SR}_P(I)$ of rules iff $\mathrm{head}(R) = I$ and there is some ordering $r_1, \dots, r_n$ of R such that, for each $1 \leq i \leq n$, $\mathrm{head}(\{r_1, \dots, r_{i-1}\}) \models \mathrm{B}(r_i)^I$.

Proposition 2.2. A (supported) model $I \subseteq \mathrm{At}(P)$ of a program P is stable iff I is well-supported by some $R \subseteq \mathrm{SR}_P(I)$.

Since atoms may have several defining rules in a program, it is often the case that several well-supporting sets of rules exist. The notion of well-support counteracts circularity in the *positive dependency graph* $\mathrm{DG}^+(P) = \langle \mathrm{At}(P), \succeq \rangle$ of P, whose edge relation $a \succeq b$ holds for all $a, b \in \mathrm{At}(P)$ such that $\mathrm{head}(r) = a$ and $b \in \mathrm{B}(r)^+$ for some rule $r \in P$. If $a \succeq b$, we also write $\langle a, b \rangle \in \mathrm{DG}^+(P)$. The *strongly connected components* (SCCs) of $\mathrm{DG}^+(P)$ are maximal subsets $C \subseteq \mathrm{At}(P)$ such that all contained atoms are connected to one another by directed paths in $\mathrm{DG}^+(P)$. For an atom $a \in \mathrm{At}(P)$, we denote the SCC containing a by $\mathrm{SCC}_P(a)$. The SCCs $S_1, \dots, S_n$ of a program P are central for the modularization and the compositionality of stable model semantics. Each S_i gives rise to a *module* $P_i = \bigcup_{a \in S_i} \mathrm{Def}_P(a)$ and, consequently, the program P can be partitioned into *disjoint* modules $P_1, \dots, P_n$. An interpretation $I \subseteq \mathrm{At}(P_i)$ is a *stable model* of the module P_i iff $I = \mathrm{LM}({P_i}^I \cup \{a \leftarrow \mid a \in I \setminus S_i\})$. A collection of interpretations $I_i \subseteq \mathrm{At}(P_i)$ for $1 \leq i \leq n$ is *compatible* iff, for each $i, j \in \{1, \dots, n\}$, $I_i \cap \mathrm{At}(P_j) = I_j \cap \mathrm{At}(P_i)$. The *module theorem* [19] states that the stable models of P match the compatible collections of stable models for the modules $P_1, \dots, P_n$.

While SCCs provide the finest possible granularity for the module theorem to hold, it is clear that we can form arbitrary unions of SCC-based modules, so that the stable models of the unions and the stable models of the entire program still match through the compatibility condition. In particular, we will exploit the traditional scenario of two programs [20], called the *bottom* B and the *top* T, such that the SCCs of $B \cup T$ are local to B and T, i.e., there is no SCC S of $B \cup T$ such that both $S \cap \mathrm{head}(B) \neq \emptyset$ and $S \cap \mathrm{head}(T) \neq \emptyset$. This implies that $\mathrm{head}(B) \cap \mathrm{head}(T) = \emptyset$, and no two atoms $a \in \mathrm{head}(B)$ and $b \in \mathrm{head}(T)$ can mutually depend on each other in $\mathrm{DG}^+(B \cup T)$. Moreover, one usually picks B and T such that $\mathrm{head}(T) \cap \mathrm{At}(B) = \emptyset$, i.e., the bottom B is a stand-alone program that does not refer to atoms defined in T, and thus T can intuitively be viewed to be on top of B. Given this, the following proposition rephrases the *splitting set theorem* from [20].

Proposition 2.3. An interpretation $I \subseteq \mathrm{At}(B \cup T)$ is a stable (resp. supported) model of $B \cup T$ iff

1. $I_B = I \cap \mathrm{At}(B)$ is a stable (resp. supported) model of B and
2. $I_T = I \cap \mathrm{At}(T)$ is a stable (resp. supported) model of $T \cup \{a \leftarrow \mid a \in I_B \cap \mathrm{At}(T)\}$.

3. Acyclicity Constraints

In previous work [10], the SAT problem has been extended by explicit acyclicity constraints. The basic idea is to label edges of a directed graph with dedicated Boolean variables. While satisfying the clauses of a SAT instance referring to these labeling variables, also the directed graph consisting of edges whose labeling variables are true must be kept acyclic. Thus, the graph behind the labeling variables imposes an additional constraint on satisfying assignments. In what follows, we propose a similar extension of logic programs subject to stable model semantics.

Definition 3.1. An acyclicity extension of a logic program P is a pair $\langle V, e\rangle$, where

1. V is a set of nodes and
2. $e : \mathrm{At}(P) \to V \times V$ is a partial injection that maps atoms of P to edges.

In the sequel, a program P is called an *acyclicity program* if it is accompanied by an acyclicity extension $\langle V, e\rangle$. To define the semantics of acyclicity programs, we identify the graph of the acyclicity check as follows. Given an interpretation $I \subseteq \mathrm{At}(P)$, we write $e(I)$ for the set of edges $e(a)$ induced by atoms $a \in I$ for which $e(a)$ is defined. For a given acyclicity extension $\langle V, e\rangle$, the graph $e(\mathrm{At}(P))$ is the maximal one that can be obtained under any interpretation and is therefore likely to contain cycles. (Otherwise, the extension can be neglected altogether as no cycles can arise.) To be precise about the acyclicity condition being imposed, we recall that a graph $\langle V, E\rangle$ with the set $E \subseteq V \times V$ of edges has a *cycle* iff there is a non-trivial directed path from any node $v \in V$ back to itself via the edges in E. An *acyclic* graph $\langle V, E\rangle$ has no cycles of this kind.

Definition 3.2. Let P be an acyclicity program with an acyclicity extension $\langle V, e\rangle$. An interpretation $M \subseteq \mathrm{At}(P)$ is a stable (or supported) model of P subject to $\langle V, e\rangle$ iff M is a stable (or supported) model of P such that the graph $\langle V, e(M)\rangle$ is acyclic.

Example 3.3. Consider a directed graph $G = \langle V, E\rangle$ and the task of finding a Hamiltonian cycle through the graph, i.e., a cycle that visits each node in V exactly once. Let us encode the graph G by introducing the fact $\mathtt{node}(v)$ for each $v \in V$ and the fact $\mathtt{edge}(v, u)$ for each $\langle v, u\rangle \in E$. Then, it is sufficient (i) to pick beforehand an arbitrary initial node, say v_0, for the cycle, (ii) to select for each node exactly one outgoing and one incoming edge to be on the cycle, and (iii) to check that the cycle is not completed before the path spanning along the selected edges returns to v_0. Assuming that a predicate $\mathtt{hc}$ is used to represent selected edges, the following (first-order) rules[1] similar to those in [21] express (ii):

$$1\{\mathtt{hc}(v,u) : \mathtt{edge}(v,u)\}1 \leftarrow \mathtt{node}(v). \tag{4}$$

$$1\{\mathtt{hc}(u,v) : \mathtt{edge}(u,v)\}1 \leftarrow \mathtt{node}(v). \tag{5}$$

To enforce (iii), we simply define an acyclicity extension $\langle V, e\rangle$, where V is the set of nodes of G and e maps an atom $\mathtt{hc}(v, u)$ to an edge $\langle v, u\rangle$ whenever v and u are different from v_0. (A simple mechanism to implement such acyclicity extensions in practice is described in Section 4.) ■

Our next objective is to relate acyclicity programs to ordinary logic programs in terms of translations. Clearly, the fact that logic programs subject to stable model semantics can express reachability in graphs implies that also acyclicity is expressible. To this end, we present a translation based on *elimination orderings* [22], which are closely related to topological orderings for directed graphs.

[1] Grounders map choice rules with lower and upper bounds to rules of the form (2) along with constraints of the form (3) at the propositional level.

Definition 3.4. Let P be an acyclicity program with an acyclicity extension $\langle V, e\rangle$. The translation $\mathrm{Tr}_{\mathrm{EL}}(P, V, e)$ extends P as follows.

1. For each atom $a \in \mathrm{At}(P)$ such that $e(a) = \langle v, u\rangle$, the rules:

$$\texttt{el}(v, u) \leftarrow \texttt{not}\ a. \tag{6}$$
$$\texttt{el}(v, u) \leftarrow \texttt{el}(u). \tag{7}$$

2. For each node $v \in V$ such that $\langle v, u_1\rangle, \dots, \langle v, u_k\rangle$ are the edges in $e(\mathrm{At}(P))$ starting from v:

$$\texttt{el}(v) \leftarrow \texttt{el}(v, u_1),\ \dots,\ \texttt{el}(v, u_k). \tag{8}$$
$$\leftarrow \texttt{not}\ \texttt{el}(v). \tag{9}$$

Moreover, we define the set of new atoms introduced by the translation as

$$\mathrm{At}_{\mathrm{EL}}(P, V, e) = \{\texttt{el}(v, u) \mid \langle v, u\rangle \in e(\mathrm{At}(P))\} \cup \{\texttt{el}(v) \mid v \in V\}.$$

The intuitive reading of the new atom $\texttt{el}(v, u)$ is that the edge $\langle v, u\rangle \in e(\mathrm{At}(P))$ has been eliminated, meaning that it cannot belong to any cycle. Analogously, the atom $\texttt{el}(v)$ denotes the elimination of a node $v \in V$. By the rule (6), an edge $\langle v, u\rangle$ is eliminated when the atom a such that $e(a) = \langle v, u\rangle$ is false, while the rule (7) is applicable once the end node u is eliminated. Then, the node v gets eliminated by the rule (8) if all edges starting from it are eliminated. Finally, the constraint (9) ensures that all nodes are eliminated. That is, the success of the acyclicity test presumes that $\texttt{el}(v, u)$ or $\texttt{el}(v)$, respectively, is derivable for each edge $\langle v, u\rangle \in e(\mathrm{At}(P))$ and each node $v \in V$. The fact that $\mathrm{Tr}_{\mathrm{EL}}(P, V, e) \setminus P$ indeed implements an acyclicity check for the acyclicity program P is made precise below.

Lemma 3.5. Let P be an acyclicity program with an acyclicity extension $\langle V, e\rangle$, $I \subseteq \mathrm{At}(P)$ an interpretation, and L the set of atoms $a \in I$ for which $e(a)$ is defined. Then, the graph $\langle V, e(I)\rangle$ is acyclic iff $L \cup \mathrm{At}_{\mathrm{EL}}(P, V, e)$ is a stable model of the program $(\mathrm{Tr}_{\mathrm{EL}}(P, V, e) \setminus P) \cup \{a \leftarrow \mid a \in L\}$.

Proof:
Define $Q = (\mathrm{Tr}_{\mathrm{EL}}(P, V, e) \setminus P) \cup \{a \leftarrow \mid a \in L\}$ and $E = \mathrm{At}_{\mathrm{EL}}(P, V, e)$. It is well-known that any graph, including $\langle V, e(I)\rangle$, is acyclic iff its nodes V can be ordered topologically into a sequence $v_1, \dots, v_n$ such that, for all $1 \le i \le j \le n$, $\langle v_i, v_j\rangle \notin e(I)$.

($\Longrightarrow$) Let the graph $\langle V, e(I)\rangle$ be acyclic and $v_1, \dots, v_n$ be some topological ordering of V. Observe that the interpretation $L \cup E$ clearly satisfies all rules of the forms (6) to (9). Now consider the least model $N = \mathrm{LM}(Q^{L\cup E})$. Certainly, $L \subseteq N$ and $N \subseteq \mathrm{head}(Q) = L \cup E$. We prove by induction that also $E \subseteq N$. Consider each atom v_i in the ordering $v_1, \dots, v_n$ and the induction hypothesis (IH) that $\{\texttt{el}(v_1), \dots, \texttt{el}(v_{i-1})\} \subseteq N$. The atom $\texttt{el}(v_i)$ is defined in $Q^{L\cup E}$ by a single positive rule r of the form (8) with $\mathrm{head}(r) = \texttt{el}(v_i)$. Let us take any $\texttt{el}(v_i, v_j) \in \mathrm{B}(r)^+$ and consider the two cases $j < i$ and $j \ge i$. If $j < i$, by IH, $\texttt{el}(v_j) \in N$, and since N satisfies all of the rules (7), $\texttt{el}(v_i, v_j) \in N$. If $j \ge i$, the topological ordering gives $\langle v_i, v_j\rangle \notin e(I)$, and correspondingly for a such that $e(a) = \langle v_i, v_j\rangle$, we obtain $a \notin L$. Then, a rule (6) in Q turns into the fact $\texttt{el}(v_i, v_j)$ in the reduct $Q^{L\cup E}$ satisfied by N, yielding $\texttt{el}(v_i, v_j) \in N$. Thus, in either case, $\texttt{el}(v_i, v_j) \in N$. Hence, $N \models \mathrm{B}(r)$ and $\texttt{el}(v_i) \in N$. As the argument applies to all $v_1, \dots, v_n$, we have that $\{\texttt{el}(v) \mid v \in V\} \subseteq N$. In view of the consequent satisfaction of the bodies of rules (7), also $\{\mathtt{el}(v,u) \mid \langle v,u\rangle \in e(\mathrm{At}(P))\} \subseteq N$. This completes the proof of $E \subseteq N$, and therefore also that of $L \cup E = N = \mathrm{LM}(Q^{L\cup E})$. As noted before, $L \cup E \models Q$, and hence $L \cup E$ is a stable model of Q.

($\Longleftarrow$) Let $N = L \cup E$ be a stable model of Q. Since N is the least model of Q^N, each $a \in N$ has a unique index $i \geq 1$ such that $a \in (\mathrm{T}_{Q^N} \uparrow i) \setminus (\mathrm{T}_{Q^N} \uparrow i-1)$. These indices yield a partial strict ordering of N with the property that, in any compatible total ordering $<_N$ of N, each atom $a \in N$ is derivable by at least one rule $r \in \mathrm{Def}_Q(a)^N$ such that

$$\{b \in \mathrm{B}(r)^+ \mid b <_N \mathrm{head}(r)\} \models \mathrm{B}(r). \tag{10}$$

Take any edge $\langle v,u\rangle \in e(I)$ associated with an atom $a \in L$ such that $e(a) = \langle v,u\rangle$. The atom $\mathtt{el}(v,u)$ is derivable by at least one rule $r \in \mathrm{Def}_Q(\mathtt{el}(v,u))^N$ such that (10) holds, which can only be of one of two forms. Either the rule is $\mathtt{el}(v,u) \leftarrow$, which has been reduced from a rule of the form (6) containing the negative body atom a. Since this leads to the contradiction $a \notin L$, the rule must be $\mathtt{el}(v,u) \leftarrow \mathtt{el}(u)$, as in (7), in which case (10) implies that $\mathtt{el}(u) <_N \mathtt{el}(v,u)$. On the other hand, the atom $\mathtt{el}(v)$ is derivable only by a rule of the form $\mathtt{el}(v) \leftarrow \mathtt{el}(v,u_1), \ldots, \mathtt{el}(v,u_k)$, as in (8), such that $u \in \{u_1,\ldots,u_k\}$. Hence, (10) implies that $\mathtt{el}(v,u) <_N \mathtt{el}(v)$. These observations cover all edges $\langle v,u\rangle \in e(I)$, so that $\mathtt{el}(u) <_N \mathtt{el}(v)$ holds. That is, an ordering by $<_N$ of the nodes in $\langle V, e(I)\rangle$ is topological, and therefore the graph is acyclic. □

Theorem 3.6. Let P be an acyclicity program with an acyclicity extension $\langle V,e\rangle$, and $\mathrm{Tr}_{\mathrm{EL}}(P,V,e)$ its translation into an ordinary logic program.

1. If M is a stable model of P subject to $\langle V,e\rangle$, then the interpretation $N = M \cup \mathrm{At}_{\mathrm{EL}}(P,V,e)$ is a stable model of $\mathrm{Tr}_{\mathrm{EL}}(P,V,e)$.

2. If N is a stable model of $\mathrm{Tr}_{\mathrm{EL}}(P,V,e)$, then $N \setminus \mathrm{At}(P) = \mathrm{At}_{\mathrm{EL}}(P,V,e)$ and the projection $M = N \cap \mathrm{At}(P)$ is a stable model of P subject to $\langle V,e\rangle$.

Proof:
The proof is based on the equivalence of the following conditions:

1. M is a stable model of P subject to $\langle V,e\rangle$.

2. M is a stable model of P and the graph $\langle V, e(M)\rangle$ is acyclic.

3. M is a stable model of P and $L \cup \mathrm{At}_{\mathrm{EL}}(P,V,e)$ is a stable model of the program $(\mathrm{Tr}_{\mathrm{EL}}(P,V,e) \setminus P) \cup \{a \leftarrow \mid a \in L\}$, where L is the set of atoms $a \in M$ for which $e(a)$ is defined.

4. $M \cup \mathrm{At}_{\mathrm{EL}}(P,V,e)$ is a stable model of $\mathrm{Tr}_{\mathrm{EL}}(P,V,e)$.

The first two conditions are equivalent by Definition 3.2. The second and third item are equivalent by Lemma 3.5. The last two conditions can be proven equivalent via Proposition 2.3. Indeed, take the program P for the bottom B, $\mathrm{Tr}_{\mathrm{EL}}(P,V,e) \setminus P$ for the top T, the interpretation $M \cup \mathrm{At}_{\mathrm{EL}}(P,V,e)$ for the interpretation I in the proposition, the interpretation M for I_B, and $L \cup \mathrm{At}_{\mathrm{EL}}(P,V,e)$ for I_T. □

Example 3.7. We will use the following program P as running example for illustrating translations:

$$\begin{array}{lllll} r_1\colon\ p \leftarrow q. & r_3\colon\ q \leftarrow p. & r_5\colon\ s \leftarrow p. & r_7\colon\ t \leftarrow p. & r_9\colon\ x \leftarrow \texttt{not}\ y. \\ r_2\colon\ p \leftarrow s,\, t. & r_4\colon\ \{q\} \leftarrow y. & r_6\colon\ \{s\} \leftarrow y. & r_8\colon\ \{t\} \leftarrow y. & r_{10}\colon\ y \leftarrow \texttt{not}\ x. \end{array}$$

Taken as ordinary logic program, P admits five stable models: $\{x\}$, $\{y\}$, $\{s, y\}$, $\{t, y\}$, and $\{p, q, s, t, y\}$. Let us now augment P with an acyclicity extension $\langle V, e\rangle$ such that $V = \{p, q, s, t\}$ and e is the mapping to edges given by $e(p) = \langle q, p\rangle$, $e(q) = \langle p, q\rangle$, $e(s) = \langle p, s\rangle$, and $e(t) = \langle p, t\rangle$. (That is, the nodes and edges form a subgraph of the positive dependency graph of P.) Then, $\{p, q, s, t, y\}$ is no longer a stable model of the resulting acyclicity program because it induces the graph $\langle V, \{\langle q, p\rangle, \langle p, q\rangle, \langle p, s\rangle, \langle p, t\rangle\}\rangle$, which contains a cycle between the nodes p and q. The acyclicity condition on $\langle V, e\rangle$ can also be expressed in terms of the ordinary program $\mathrm{Tr}_{\mathrm{EL}}(P, V, e)$, extending P by rules of the forms (6)–(9):

$$\begin{array}{llll} \texttt{el}(q,p) \leftarrow \texttt{not}\ p. & \texttt{el}(p,q) \leftarrow \texttt{not}\ q. & \texttt{el}(p,s) \leftarrow \texttt{not}\ s. & \texttt{el}(p,t) \leftarrow \texttt{not}\ t. \\ \texttt{el}(q,p) \leftarrow \texttt{el}(p). & \texttt{el}(p,q) \leftarrow \texttt{el}(q). & \texttt{el}(p,s) \leftarrow \texttt{el}(s). & \texttt{el}(p,t) \leftarrow \texttt{el}(t). \end{array}$$

$$\begin{array}{rrrr} \texttt{el}(p) \leftarrow \texttt{el}(p,q),\, \texttt{el}(p,s),\, \texttt{el}(p,t). & \texttt{el}(q) \leftarrow \texttt{el}(q,p). & \texttt{el}(s). & \texttt{el}(t). \\ \leftarrow \texttt{not}\ \texttt{el}(p). & \leftarrow \texttt{not}\ \texttt{el}(q). & \leftarrow \texttt{not}\ \texttt{el}(s). & \leftarrow \texttt{not}\ \texttt{el}(t). \end{array}$$

As stated in Theorem 3.6, $\mathrm{Tr}_{\mathrm{EL}}(P, V, e)$ captures the stable models M of P subject to $\langle V, e\rangle$, i.e., $\{x\}$, $\{y\}$, $\{s, y\}$, and $\{t, y\}$, in terms of corresponding stable models $M \cup \mathrm{At}_{\mathrm{EL}}(P, V, e)$, where $\mathrm{At}_{\mathrm{EL}}(P, V, e) = \{\texttt{el}(q,p), \texttt{el}(p,q), \texttt{el}(p,s), \texttt{el}(p,t), \texttt{el}(p), \texttt{el}(q), \texttt{el}(s), \texttt{el}(t)\}$. That is, each of the four stable models of $\mathrm{Tr}_{\mathrm{EL}}(P, V, e)$ includes all atoms of the form $\texttt{el}(v, u)$ or $\texttt{el}(v)$ introduced in the above program part $\mathrm{Tr}_{\mathrm{EL}}(P, V, e) \setminus P$. Moreover, note that $\{p, q, s, t, y\} \cup \mathrm{At}_{\mathrm{EL}}(P, V, e)$ is not a stable model of $\mathrm{Tr}_{\mathrm{EL}}(P, V, e)$ because there are no well-supporting rules (with respect to the interpretation at hand) to derive the atoms $\texttt{el}(q,p)$, $\texttt{el}(p,q)$, $\texttt{el}(p)$, and $\texttt{el}(q)$. In general, $\mathrm{Tr}_{\mathrm{EL}}(P, V, e)$ reflects a cycle in the graph $\langle V, e(M)\rangle$ by lack of well-support under $M \cup \mathrm{At}_{\mathrm{EL}}(P, V, e)$ for atoms $\texttt{el}(v, u)$ and $\texttt{el}(v)$ corresponding to the edges or nodes, respectively, on the cycle. ■

Transformations in the other direction are of interest as well, i.e., the goal is to capture stable models by exploiting the acyclicity constraint. While the existing translation from ASP into SAT modulo acyclicity [12] provides a starting point for such a transformation, the target syntax is here given by rules, including weight rules of the form (3), rather than clauses only.

Definition 3.8. Let P be a weight constraint program. The acyclicity translation of P consists of $\mathrm{Tr}_{\mathrm{ACYC}}(P) = \bigcup_{a\in\mathrm{At}(P)} \mathrm{Tr}_{\mathrm{ACYC}}(P, a)$ with an acyclicity extension $\langle \mathrm{At}(P), e\rangle$ such that $e(\texttt{dep}(a, b)) = \langle a, b\rangle$ for each edge $\langle a, b\rangle \in \mathrm{DG}^+(P)$, where $\mathrm{Tr}_{\mathrm{ACYC}}(P, a)$ extends $\mathrm{Def}_P(a)$ for each atom $a \in \mathrm{At}(P)$ as follows.

1. For each edge $\langle a, b\rangle \in \mathrm{DG}^+(P)$, the choice rule:

$$\{\texttt{dep}(a, b)\} \leftarrow b. \tag{11}$$

2. For each defining rule $r \in \mathrm{Def}_P(a)$ of the form (1) or (2), the rule:

$$\texttt{ws}(r) \leftarrow \texttt{dep}(a, b_1),\, \ldots,\, \texttt{dep}(a, b_n),\, \texttt{not}\ c_1,\, \ldots,\, \texttt{not}\ c_m. \tag{12}$$

3. For each defining rule $r \in \mathrm{Def}_P(a)$ of the form (3), the rule:

$$\begin{aligned}\mathtt{ws}(r) \leftarrow k \leq [\mathtt{dep}(a, b_1) = w_1, \ldots, \mathtt{dep}(a, b_n) = w_n,& \\ \mathtt{not}\ c_1 = w_{n+1}, \ldots, \mathtt{not}\ c_m = w_{n+m}].& \quad (13)\end{aligned}$$

4. For $\mathrm{Def}_P(a) = \{r_1, \ldots, r_k\}$, the constraint:

$$\leftarrow a, \mathtt{not}\ \mathtt{ws}(r_1), \ldots, \mathtt{not}\ \mathtt{ws}(r_k). \quad (14)$$

The rules (12) and (13) specify when a defining rule r provides well-support for the head atom a, i.e., the dependency of a on $\mathrm{B}(r)^+ = \{b_1, \ldots, b_n\}$ is non-circular. The constraint (14) expresses that $a \in \mathrm{At}(P)$ must have a well-supporting rule $r \in \mathrm{Def}_P(a)$ whenever a is true. To this end, respective dependencies have to be established in terms of choice rules (11). The enforcement of well-support connects the supported models of the translation, subject to acyclicity, to stable models of the original program. As is the case with sets of well-supporting rules, in general, the sets of rules captured by the $\mathtt{ws}(\cdot)$ predicate are not necessarily unique for a given stable model. Moreover, the translation aims especially at sets of well-supporting rules that are compatible with the following constructive characterization based on the T_P operator.

Given a stable model M of a program P, there is a unique *strong level ranking* [23, 24] of M for P that maps atoms $a \in M$ to indices $i \geq 1$ such that $a \in (\mathrm{T}_{P^M} \uparrow i) \setminus (\mathrm{T}_{P^M} \uparrow i - 1)$. For any $a \in \mathrm{At}(P)$ and a set $D \subseteq M$ of atoms, let $\mathrm{WS}_P^M(a, D) = \{r \in \mathrm{Def}_P(a) \mid M \models \mathrm{B}(r), D \models \mathrm{B}(r)^M\}$ denote the set of rules that can be used to derive a, while positively depending on atoms in D only. Then, minimal (in terms of subset inclusion) choices of D sufficient to derive a according to its level rank can be characterized as follows.

Definition 3.9. Let M be a stable model of a program P, and $a \in M$ an atom mapped to index $i \geq 1$ in the strong level ranking of M for P. A T_{P^M}*-induced* set of positive dependencies of a is a minimal set $D \subseteq \mathrm{T}_{P^M} \uparrow i - 1$ of atoms such that $\mathrm{WS}_P^M(a, D) \neq \emptyset$. Moreover, the set of all T_{P^M}-induced sets of positive dependencies of a is denoted by $\mathcal{D}_P^M(a)$.

Note that, for every atom a in a stable model M of P, we have that $\mathcal{D}_P^M(a) \neq \emptyset$. Given some choice of $D_a \in \mathcal{D}_P^M(a)$ for each $a \in M$, the entire model M is well-supported by $R = \bigcup_{a \in M} \mathrm{WS}_P^M(a, D_a)$. In fact, the well-support provided by R is witnessed by any ordering $r_1, \ldots, r_n$ of R such that the level ranks of respective head atoms are monotonically increasing. Regarding the acyclicity translation $\mathrm{Tr}_{\mathrm{ACYC}}(P)$, the set $R \cup \bigcup_{a \in \mathrm{At}(P)} \mathrm{WS}_P^M(a, \emptyset)$ of rules allows for expressing the stability of M in terms of the $\mathtt{ws}(\cdot)$ predicate, where $\bigcup_{a \in \mathrm{At}(P)} \mathrm{WS}_P^M(a, \emptyset)$ accounts for applicable choice rules whose head atoms need not be included in M. We will make use of such sets of rules to show the correctness of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ below.

Example 3.10. Consider the following weight constraint program P:

$$\begin{array}{lll} \{a\}. & b \leftarrow a. & c \leftarrow a. \\ d \leftarrow b, \mathtt{not}\ e. & d \leftarrow 1 \leq [b = 1, e = 1]. & d \leftarrow c. \end{array}$$

Let us verify that $M = \{a, b, c, d\}$ is a stable model of P by computing $\mathrm{LM}(P^M)$ via T_{P^M}. The respective T_{P^M}-induced sets of positive dependencies and corresponding well-supporting rules are:

i	$\mathrm{T}_{P^M} \uparrow i$	$\mathcal{D}_P^M(\cdot)$	$\mathrm{WS}_P^M(\cdot,\cdot)$
0	$\emptyset$		
1	$\{a\}$	$\mathcal{D}_P^M(a) = \{\emptyset\}$	$\mathrm{WS}_P^M(a,\emptyset) = \{\{a\}\}$
2	$\{a,b,c\}$	$\mathcal{D}_P^M(b) = \{\{a\}\}$	$\mathrm{WS}_P^M(b,\{a\}) = \{b \leftarrow a\}$
		$\mathcal{D}_P^M(c) = \{\{a\}\}$	$\mathrm{WS}_P^M(c,\{a\}) = \{c \leftarrow a\}$
3	$\{a,b,c,d\}$	$\mathcal{D}_P^M(d) = \{\{b\},\{c\}\}$	$\mathrm{WS}_P^M(d,\{b\}) = \{d \leftarrow b, \mathtt{not}\ e;$ $d \leftarrow 1 \leq [b=1,\ e=1]\}$
			$\mathrm{WS}_P^M(d,\{c\}) = \{d \leftarrow c\}$

That is, the positive dependencies given by T_{P^M}-induced sets of a, b, and c are unique, and likewise the respective sets of well-supporting rules. The atom d has two T_{P^M}-induced sets, $\{b\}$ and $\{c\}$, leading to two alternative sets of well-supporting rules. In particular, $\mathrm{WS}_P^M(d,\{b\})$ consists of two rules that are both applicable based on the dependency to b. ■

Theorem 3.11. Let P be a weight constraint program, and $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ its translation into an acyclicity program with an acyclicity extension $\langle \mathrm{At}(P), e\rangle$.

1. If M is a stable model of P and, for each $a \in M$, some $D_a \in \mathcal{D}_P^M(a)$ is fixed, then $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ subject to $\langle \mathrm{At}(P), e\rangle$ has the supported model
$$\begin{aligned} N = M &\cup \{\mathtt{ws}(r) \mid a \in \mathrm{At}(P), r \in \mathrm{WS}_P^M(a,\emptyset)\} \\ &\cup \{\mathtt{ws}(r) \mid a \in M, r \in \mathrm{WS}_P^M(a,D_a)\} \\ &\cup \{\mathtt{dep}(a,b) \mid a \in M, b \in D_a\}. \end{aligned}$$

2. If N is a supported model of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ subject to $\langle \mathrm{At}(P), e\rangle$, then $M = N \cap \mathrm{At}(P)$ is a stable model of P that is well-supported by $R = \{r \mid \mathtt{ws}(r) \in N, \mathrm{head}(r) \in M\}$.

Proof:
(1.) Let M be a stable model of P and N the interpretation defined as above. Since $M = N \cap \mathrm{At}(P)$ is a stable model of $P \subseteq \mathrm{Tr}_{\mathrm{ACYC}}(P)$, N satisfies P and M is supported by P. To show that N is a supported model of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$, we continue by further verifying that N satisfies $\mathrm{Tr}_{\mathrm{ACYC}}(P) \setminus P$ and $N \setminus M$ is supported by $\mathrm{Tr}_{\mathrm{ACYC}}(P) \setminus P$. Then, we prove that $\langle \mathrm{At}(P), e(N)\rangle$ is acyclic.

For satisfaction, consider atoms $a \in \mathrm{At}(P)$ defined by rules $r \in \mathrm{Def}_P(a)$ involved in the heads $\mathtt{ws}(r)$ of rules r' of the form (12) or (13). Let $D = \{b \mid \mathtt{dep}(a,b) \in N\}$ and $D' = \{\mathtt{dep}(a,b) \mid b \in D\}$, and assume that $N \models \mathrm{B}(r')$. Due to the form of r', this implies that $M \models \mathrm{B}(r)$, $D' \models \mathrm{B}(r')^M$, and $D \models \mathrm{B}(r)^M$. Given that $D \neq \emptyset$ yields $a \in M$ and $D = D_a$ by the definition of N, we have that $r \in \mathrm{WS}_P^M(a,D)$. This shows that $\mathtt{ws}(r) \in N$ and $N \models r'$. Moreover, for each atom $a \in M$, the constraint r' of the form (14) is satisfied because the stability of M guarantees the existence of some rule $r \in \mathrm{WS}_P^M(a,D_a)$, so that $\mathtt{ws}(r) \in N \cap \mathrm{B}(r')^-$.

Regarding support, each atom of the form $\mathtt{dep}(a,b) \in N$ is supported by a rule of the form (11) because $D_a \subseteq M$. For any atom of the form $\mathtt{ws}(r)$, assume that $\mathtt{ws}(r) \in N$. This is only possible if $r \in \mathrm{WS}_P^M(a,D)$, where $a \in \mathrm{At}(P)$ and $D = \emptyset$ or $a \in M$ and $D = D_a$. In both cases, $M \models \mathrm{B}(r)$ and $D \models \mathrm{B}(r)^M$, which yields $\{\mathtt{dep}(a,b) \mid b \in D\} \models \mathrm{B}(r')^N$, $N \models \mathrm{B}(r')^N$, and $N \models \mathrm{B}(r')$.

The stability of M yields level ranks for all atoms in M. Moreover, any ordering $a_1, \dots, a_n$ of M in which the level ranks are monotonically increasing is topological in the following way: for each $1 \leq i \leq n$, $\texttt{dep}(a_i, a_j) \in N$ implies $a_j \in \{a_1, \dots, a_{i-1}\}$, so that $j < i$ for all $\langle a_i, a_j \rangle \in e(N)$. Consequently, the graph $\langle \mathrm{At}(P), e(N) \rangle$ is acyclic.

(2.) Let N be a supported model of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ subject to $\langle \mathrm{At}(P), e \rangle$. Consider $M = N \cap \mathrm{At}(P)$ and $R = \{r \mid \texttt{ws}(r) \in N, \mathrm{head}(r) \in M\}$. We show that $M \models P$, $R \subseteq \mathrm{SR}_P(M)$, $\mathrm{head}(R) = M$, and construct an ordering of R to prove that M is well-supported by R.

The facts that $N \models \mathrm{Tr}_{\mathrm{ACYC}}(P)$, $P \subseteq \mathrm{Tr}_{\mathrm{ACYC}}(P)$, and $M = N \cap \mathrm{At}(P)$ imply that $M \models P$.

To see that $R \subseteq \mathrm{SR}_P(M)$, take any $r \in R$ and its unique counterpart $r' \in \mathrm{Tr}_{\mathrm{ACYC}}(P)$ of the form (12) or (13) with $\mathrm{head}(r') = \texttt{ws}(r)$. Given that N is a supported model of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$, in which $\texttt{ws}(r)$ is defined by r' alone, and since each atom of the form $\texttt{dep}(a, b)$ is likewise defined by a single rule $\{\texttt{dep}(a, b)\} \leftarrow b$, we have that $N \models \mathrm{B}(r')$ and $\{b \mid \texttt{dep}(a, b) \in N\} \subseteq M$. This implies that $M \models \mathrm{B}(r)$ and $r \in \mathrm{SR}_P(M)$.

For any atom $a \in M$, the satisfaction of a rule of the form (14) by N implies that $\texttt{ws}(r) \in N$ for some $r \in \mathrm{Def}_P(a)$. Thus, we have that $r \in R$ and $a \in \mathrm{head}(R)$, which in turn yields $\mathrm{head}(R) = M$.

Since the graph $\langle \mathrm{At}(P), e(N) \rangle$ is acyclic, the atoms in $\mathrm{At}(P)$ have a topological ordering $a_1, \dots, a_n$ such that $j < i$ for any $\texttt{dep}(a_i, a_j) \in N$. Therefore, the rules in R have an ordering $r_1, \dots, r_m$ in which $\{b \mid \texttt{dep}(\mathrm{head}(r_i), b) \in N\} \subseteq \{\mathrm{head}(r_j) \mid j < i\}$ for any $1 \leq i \leq m$. That is, $\{\texttt{dep}(\mathrm{head}(r_i), b) \in N \mid b \in \mathrm{head}(\{r_1, \dots, r_{i-1}\})\} \models \mathrm{B}(r'_i)^M$ for the unique rule $r'_i \in \mathrm{Tr}_{\mathrm{ACYC}}(P)$ such that $\mathrm{head}(r'_i) = \texttt{ws}(r_i)$. This implies that $\mathrm{head}(\{r_1, \dots, r_{i-1}\}) \models \mathrm{B}(r_i)^M$, so that M is well-supported by R. □

It is well-known that supported and stable models coincide for *tight* logic programs [25, 26]. The following theorem shows that translations produced by $\mathrm{Tr}_{\mathrm{ACYC}}$ possess an analogous property subject to the acyclicity extension $\langle \mathrm{At}(P), e \rangle$. This opens up an interesting avenue for investigating the efficiency of stable model computation—using either unfounded set checking, the acyclicity constraint, or both.

Proposition 3.12. Let P be a weight constraint program, $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ its translation into an acyclicity program with an acyclicity extension $\langle \mathrm{At}(P), e \rangle$, and $M \subseteq \mathrm{At}(\mathrm{Tr}_{\mathrm{ACYC}}(P))$ an interpretation. Then, M is a supported model of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ subject to $\langle \mathrm{At}(P), e \rangle$ iff M is a stable model of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ subject to $\langle \mathrm{At}(P), e \rangle$.

Proof:
Suppose that M is subject to the acyclicity extension $\langle \mathrm{At}(P), e \rangle$, i.e., $\langle \mathrm{At}(P), e(M) \rangle$ is acyclic.

($\Longrightarrow$) Let M be a supported model of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$. Since M is subject to $\langle \mathrm{At}(P), e \rangle$, by the second item of Theorem 3.11, we have that $M_B = M \cap \mathrm{At}(P)$ is a stable model of P. For $M_T = M \cap \mathrm{At}(\mathrm{Tr}_{\mathrm{ACYC}}(P) \setminus P)$ and the set $F = \{a \leftarrow \mid a \in M_B \cap M_T\}$ of facts, Proposition 2.3 yields that M_T is a supported model of $(\mathrm{Tr}_{\mathrm{ACYC}}(P) \setminus P) \cup F$. Since $\mathrm{Tr}_{\mathrm{ACYC}}(P) \setminus P$ is tight, M_T is also a stable model of $(\mathrm{Tr}_{\mathrm{ACYC}}(P) \setminus P) \cup F$. By Proposition 2.3, it follows that M is a stable model of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$.

($\Longleftarrow$) The stability of M with respect to $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ implies that M is a supported model too. □

Example 3.13. The acyclicity translation $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ extends P from Example 3.7 by the following rules of the forms (11)–(14):

$$
\begin{array}{llll}
\{\texttt{dep}(p,q)\} \leftarrow q. & \{\texttt{dep}(p,s)\} \leftarrow s. & \{\texttt{dep}(p,t)\} \leftarrow t. & \{\texttt{dep}(q,y)\} \leftarrow y. \\
\{\texttt{dep}(q,p)\} \leftarrow p. & \{\texttt{dep}(s,p)\} \leftarrow p. & \{\texttt{dep}(t,p)\} \leftarrow p. & \{\texttt{dep}(s,y)\} \leftarrow y. \\
 & & & \{\texttt{dep}(t,y)\} \leftarrow y.
\end{array}
$$

$$
\begin{array}{lll}
\texttt{ws}(r_1) \leftarrow \texttt{dep}(p,q). & \texttt{ws}(r_2) \leftarrow \texttt{dep}(p,s),\ \texttt{dep}(p,t). & \leftarrow p,\ \texttt{not}\ \texttt{ws}(r_1),\ \texttt{not}\ \texttt{ws}(r_2). \\
\texttt{ws}(r_3) \leftarrow \texttt{dep}(q,p). & \texttt{ws}(r_4) \leftarrow \texttt{dep}(q,y). & \leftarrow q,\ \texttt{not}\ \texttt{ws}(r_3),\ \texttt{not}\ \texttt{ws}(r_4). \\
\texttt{ws}(r_5) \leftarrow \texttt{dep}(s,p). & \texttt{ws}(r_6) \leftarrow \texttt{dep}(s,y). & \leftarrow s,\ \texttt{not}\ \texttt{ws}(r_5),\ \texttt{not}\ \texttt{ws}(r_6). \\
\texttt{ws}(r_7) \leftarrow \texttt{dep}(t,p). & \texttt{ws}(r_8) \leftarrow \texttt{dep}(t,y). & \leftarrow t,\ \texttt{not}\ \texttt{ws}(r_7),\ \texttt{not}\ \texttt{ws}(r_8). \\
 & \texttt{ws}(r_9) \leftarrow \texttt{not}\ y. & \leftarrow x,\ \texttt{not}\ \texttt{ws}(r_9). \\
 & \texttt{ws}(r_{10}) \leftarrow \texttt{not}\ x. & \leftarrow y,\ \texttt{not}\ \texttt{ws}(r_{10}).
\end{array}
$$

Along with the acyclicity extension $\langle \mathrm{At}(P), e\rangle$ such that $e(\texttt{dep}(a,b)) = \langle a,b\rangle$ for all $\langle a,b\rangle \in \mathrm{DG}^+(P)$, the additional rules in $\mathrm{Tr}_{\mathrm{ACYC}}(P) \setminus P$ encode well-supports for $\mathrm{At}(P) = \{p,q,s,t,x,y\}$. To this end, rules (12), with heads of the form $\texttt{ws}(r)$ for $r \in P$, rely on $\texttt{dep}(\mathrm{head}(r), b)$ for all positive body atoms $b \in \mathrm{B}(r)^+$. (A sufficient amount of such atoms is required to establish the bound k in the counterpart (13) of a weight rule; such conditions will be discussed in Example 3.17 below.) By mapping these prerequisites to edges, the acyclicity constraint enforces the conditions of well-supporting rules. To illustrate this idea further, let us inspect the sets of edges related to particular interpretations $M \subseteq \mathrm{At}(P)$:

1. Considering the stable model $\{x\}$ of P, the choice rules for $\texttt{dep}(a,b)$ atoms are inapplicable, while $\texttt{ws}(r_9)$ holds because y is false. Hence, the only stable model of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ subject to $\langle \mathrm{At}(P), e\rangle$ that corresponds to $\{x\}$ is $\{x\} \cup \{\texttt{ws}(r_9)\}$, inducing the graph $\langle \mathrm{At}(P), \emptyset\rangle$.

2. The evidently analogous stable model $\{y\}$ of P is captured by the stable model $\{y\} \cup \{\texttt{ws}(r_{10})\}$ of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ subject to $\langle \mathrm{At}(P), e\rangle$. However, the prerequisites of choice rules for $\texttt{dep}(q,y)$, $\texttt{dep}(s,y)$, and $\texttt{dep}(t,y)$ hold in this case, and adding edges associated with these atoms to the graph $\langle \mathrm{At}(P), \emptyset\rangle$ does not lead to any cycle. Thus, any combination of $\texttt{dep}(q,y)$, $\texttt{dep}(s,y)$, and $\texttt{dep}(t,y)$ along with respective atoms among $\texttt{ws}(r_4)$, $\texttt{ws}(r_6)$, and $\texttt{ws}(r_8)$ derived in turn yields an alternative representation of $\{y\}$, where $\{y\} \cup \{\texttt{dep}(q,y), \texttt{dep}(s,y), \texttt{dep}(t,y), \texttt{ws}(r_4), \texttt{ws}(r_6), \texttt{ws}(r_8), \texttt{ws}(r_{10})\}$ constitutes the maximum of the available options. In total, we obtain eight stable models N of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ subject to $\langle \mathrm{At}(P), e\rangle$ such that $N \cap \mathrm{At}(P) = \{y\}$.

3. A similar situation as with $\{y\}$ above applies to the stable model $\{s,y\}$ of P. Here, $\{s,y\}$ must be augmented with $\{\texttt{dep}(s,y), \texttt{ws}(r_6), \texttt{ws}(r_{10})\}$ for expressing well-support, while any combination of $\texttt{dep}(p,s)$, $\texttt{dep}(q,y)$, and $\texttt{dep}(t,y)$, the latter two accompanied by $\texttt{ws}(r_4)$ or $\texttt{ws}(r_8)$, respectively, does not yield a cycle when it is included in addition. Hence, we again obtain eight stable models N of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ subject to $\langle \mathrm{At}(P), e\rangle$ that represent $N \cap \mathrm{At}(P) = \{s,y\}$, where $\{s,y\} \cup \{\texttt{dep}(s,y), \texttt{ws}(r_6), \texttt{ws}(r_{10})\} \subseteq N \subseteq \{s,y\} \cup \{\texttt{dep}(p,s), \texttt{dep}(q,y), \texttt{dep}(s,y), \texttt{dep}(t,y), \texttt{ws}(r_4), \texttt{ws}(r_6), \texttt{ws}(r_8), \texttt{ws}(r_{10})\}$.

4. The case of the stable model $\{t,y\}$ of P is analogous to the previous one, yet with $\texttt{dep}(t,y)$ and $\texttt{ws}(r_8)$ playing the roles of $\texttt{dep}(s,y)$ and $\texttt{ws}(r_6)$, and vice versa. That is, there are also eight stable models N of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ subject to $\langle \mathrm{At}(P), e\rangle$ such that $N \cap \mathrm{At}(P) = \{t,y\}$.

5. For the final stable model $\{p,q,s,t,y\}$ of P, there are even further ways to reflect its well-supports in terms of atoms $\texttt{dep}(p,q)$, $\texttt{dep}(p,s)$, $\texttt{dep}(p,t)$, $\texttt{dep}(q,p)$, $\texttt{dep}(s,p)$, $\texttt{dep}(t,p)$, $\texttt{dep}(q,y)$,

$\texttt{dep}(s, y)$, and $\texttt{dep}(t, y)$ associated with edges, accompanied by respective derived atoms $\texttt{ws}(r)$ for $r \in P \setminus \{r_9\}$. Legal combinations include $\texttt{dep}(p, q)$ or both $\texttt{dep}(p, s)$ and $\texttt{dep}(p, t)$ to witness well-support for p via r_1 or r_2. Moreover, to avoid cycles, $\texttt{dep}(p, q)$ and $\texttt{dep}(q, p)$, $\texttt{dep}(p, s)$ and $\texttt{dep}(s, p)$, as well as $\texttt{dep}(p, t)$ and $\texttt{dep}(t, p)$ must remain mutually exclusive, so that $\texttt{dep}(q, y)$ or, alternatively, $\texttt{dep}(s, y)$ and $\texttt{dep}(t, y)$ are needed to express initial well-support via r_4 or both r_6 and r_8. Without going into further details, one can check that there are 19 stable models N of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ subject to $\langle \mathrm{At}(P), e\rangle$ for which $\langle \mathrm{At}(P), e(N)\rangle$ remains acyclic, while $\{\mathrm{head}(r) \mid \texttt{ws}(r) \in N\} = \{p, q, s, t, y\}$ holds for each such N.

In total, the five stable models of P give rise to 44 stable models of its acyclicity translation $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ subject to $\langle \mathrm{At}(P), e\rangle$, obtained by selecting different atoms $\texttt{dep}(a, b)$ to express that $\langle a, b\rangle \in \mathrm{DG}^+(P)$ can be utilized within well-supporting rules. Due to the acyclicity constraint on corresponding subgraphs of $\mathrm{DG}^+(P)$, stable models coincide with supported models of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ subject to $\langle \mathrm{At}(P), e\rangle$, as stated in Proposition 3.12. Notably, the acyclicity requirement is crucial for this correspondence, and supported models not matching stable models could be obtained otherwise. For instance, supported models augmenting $\{p, q, s, t, x\}$ with $\texttt{dep}(q, p)$, $\texttt{dep}(s, p)$, and $\texttt{dep}(t, p)$ as well as $\texttt{dep}(p, q)$ or both $\texttt{dep}(p, s)$ and $\texttt{dep}(p, t)$ become eligible when dropping the acyclicity extension $\langle \mathrm{At}(P), e\rangle$, where inherent cycles reflect that $\{p, q, s, t, x\}$ is not a stable model of P. ■

As witnessed by Theorem 3.11 and Proposition 3.12, the translation $\mathrm{Tr}_{\mathrm{ACYC}}$ provides means to capture stability in terms of the acyclicity constraint. However, the computational efficiency of the translation can be improved when additional constraints governing $\texttt{dep}(a, b)$ atoms are introduced. The purpose of these constraints is to falsify dependencies in settings where they are not truly needed. We first concentrate on choice programs and will then extend the consideration to weight rules below. The following definition adopts the cases from [12] but reformulates them in terms of rules rather than clauses.

Definition 3.14. Let P be a choice program. The strong acyclicity translation of P, denoted by $\mathrm{Tr}_{\mathrm{ACYC+}}(P)$, extends $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ as follows.

1. For each $\langle a, b\rangle \in \mathrm{DG}^+(P)$, the constraint:

$$\leftarrow \texttt{dep}(a, b), \texttt{not}\ a. \tag{15}$$

2. For each $\langle a, b\rangle \in \mathrm{DG}^+(P)$ and $r \in \mathrm{Def}_P(a)$ such that $b \notin \mathrm{B}(r)^+$, the constraint:

$$\leftarrow \texttt{dep}(a, b), \texttt{ws}(r). \tag{16}$$

Intuitively, dependencies from a in (15) are not needed if a is false. Similarly, a particular dependency in (16) may safely be omitted if the well-support for a is provided by a rule r not involving this dependency. That is, the constraints introduced in Definition 3.14 suppress dependencies only if they are not needed to establish well-support for a particular stable model.

Example 3.15. The strong acyclicity translation $\mathrm{Tr}_{\mathrm{ACYC+}}(P)$ of P from Example 3.7 augments $\mathrm{Tr}_{\mathrm{ACYC}}(P)$, which adds the rules given in Example 3.13 to P, with the following constraints:

$\leftarrow \mathtt{dep}(p,q),\ \mathtt{not}\ p.$	$\leftarrow \mathtt{dep}(q,p),\ \mathtt{not}\ q.$	$\leftarrow \mathtt{dep}(s,p),\ \mathtt{not}\ s.$	$\leftarrow \mathtt{dep}(t,p),\ \mathtt{not}\ t.$
$\leftarrow \mathtt{dep}(p,s),\ \mathtt{not}\ p.$	$\leftarrow \mathtt{dep}(q,y),\ \mathtt{not}\ q.$	$\leftarrow \mathtt{dep}(s,y),\ \mathtt{not}\ s.$	$\leftarrow \mathtt{dep}(t,y),\ \mathtt{not}\ t.$
$\leftarrow \mathtt{dep}(p,t),\ \mathtt{not}\ p.$			
$\leftarrow \mathtt{dep}(p,q),\ \mathtt{ws}(r_2).$	$\leftarrow \mathtt{dep}(q,p),\ \mathtt{ws}(r_4).$	$\leftarrow \mathtt{dep}(s,p),\ \mathtt{ws}(r_6).$	$\leftarrow \mathtt{dep}(t,p),\ \mathtt{ws}(r_8).$
$\leftarrow \mathtt{dep}(p,s),\ \mathtt{ws}(r_1).$	$\leftarrow \mathtt{dep}(q,y),\ \mathtt{ws}(r_3).$	$\leftarrow \mathtt{dep}(s,y),\ \mathtt{ws}(r_5).$	$\leftarrow \mathtt{dep}(t,y),\ \mathtt{ws}(r_7).$
$\leftarrow \mathtt{dep}(p,t),\ \mathtt{ws}(r_1).$			

The addition of these constraints to $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ reduces the number of stable as well as supported models subject to $\langle \mathrm{At}(P), e\rangle$, corresponding to the five stable models of P, from 44 to 10. In particular, additional models extending the minimal options $\{y\} \cup \{\mathtt{ws}(r_{10})\}$, $\{s,y\} \cup \{\mathtt{dep}(s,y), \mathtt{ws}(r_6), \mathtt{ws}(r_{10})\}$, and $\{t,y\} \cup \{\mathtt{dep}(t,y), \mathtt{ws}(r_8), \mathtt{ws}(r_{10})\}$, described in Items 2–4 of Example 3.13, are eliminated by constraints of the form (15), requiring atoms $a \in \mathrm{At}(P)$ such that $\mathtt{dep}(a,b)$ is included in a model to be true as well. Beyond that, constraints of the form (16) suppress redundant edges for which some rule is identified as well-supporting without them. This eliminates options, mentioned in Item 5 of Example 3.13, such that both $\mathtt{dep}(p,q)$ and $\mathtt{dep}(p,s)$ or $\mathtt{dep}(p,t)$, $\mathtt{dep}(q,p)$ and $\mathtt{dep}(q,y)$, $\mathtt{dep}(s,p)$ and $\mathtt{dep}(s,y)$, or $\mathtt{dep}(t,p)$ and $\mathtt{dep}(t,y)$ hold in a model. The remaining alternatives to express well-support for $\{p,q,s,t,y\}$ distinguish whether r_1 or r_2 is used for well-supporting the atom p. Moreover, r_5 and r_6 as well as r_7 and r_8 provide mutually exclusive ways to derive s or t, respectively, when $\mathtt{dep}(p,q)$ indicates well-support for p via r_1, while r_3 and r_4 offer two distinct derivations of q when $\mathtt{dep}(p,s)$ and $\mathtt{dep}(p,t)$ yield that r_2 is used to derive p. ■

We now extend the strong acyclicity translation to weight rules by including additional subprograms.

Definition 3.16. Let P be a weight constraint program, and $r \in P$ a weight rule of the form (3) such that $\mathrm{head}(r) = a$, $|\{b_1, \ldots, b_n\}| = n$, and the weights $w_1, \ldots, w_n$ are ordered according to $w_{i-1} \leq w_i$ for each $1 < i \leq n$. The strong acyclicity translation $\mathrm{Tr}_{\mathrm{ACYC+}}(P)$ of P is fortified as follows.

1. For $1 < i \leq n$, the rules:

$$\mathtt{nxt}(r,i) \leftarrow \mathtt{dep}(a, b_{i-1}). \tag{17}$$

$$\mathtt{nxt}(r,i+1) \leftarrow \mathtt{nxt}(r,i),\ i < n. \tag{18}$$

$$\mathtt{chk}(r,i) \leftarrow \mathtt{nxt}(r,i),\ \mathtt{dep}(a, b_i). \tag{19}$$

2. The weight rule:

$$\mathtt{red}(r) \leftarrow k \leq [\mathtt{chk}(r,2) = w_2, \ldots, \mathtt{chk}(r,n) = w_n, \mathtt{not}\ c_1 = w_{n+1}, \ldots, \mathtt{not}\ c_m = w_{n+m}]. \tag{20}$$

3. For each $b \in \mathrm{B}(r)^+$, the constraint:

$$\leftarrow \mathtt{dep}(a,b),\ \mathtt{red}(r). \tag{21}$$

The idea is to cancel dependencies $\langle a,b\rangle \in \mathrm{DG}^+(P)$ by the constraint (21) when the well-support obtained through r can be deemed redundant by the rule (20). To this end, the rules of the forms (17) and

(18) identify an atom b_i among $b_1, \ldots, b_n$ of smallest weight having an active dependency from a, i.e., $\texttt{dep}(a, b_i)$ is true, provided that such an i exists. By the rules of the form (19), any further dependencies from a to $b_{i+1}, \ldots, b_n$ are determined, and the rule (20) checks whether the weights associated with the positive literals $b_{i+1}, \ldots, b_n$ having an active dependency from a, together with the weights of satisfied negative literals, are sufficient to reach the bound k. If so, all dependencies from a to $\mathrm{B}(r)^+$ are viewed as redundant and denied by the constraint (21). In particular, note that this check covers cases where, e.g., negative literals suffice to satisfy the body of a weight rule and positive dependencies play no role.

Example 3.17. The program P from Example 3.7 and $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ given in Example 3.13 can be modified to yield equivalent weight constraint programs. To this end, assume that the rules with heads p (i.e., r_1 and r_2), $\texttt{ws}(r_1)$, and $\texttt{ws}(r_2)$ as well as the integrity constraint including p are replaced by:

$$\begin{aligned}
r_0\colon\quad & p \leftarrow 2 \leq [s = 1,\ t = 1,\ q = 2].\\
\texttt{ws}(r_0) & \leftarrow 2 \leq [\texttt{dep}(p, s) = 1,\ \texttt{dep}(p, t) = 1,\ \texttt{dep}(p, q) = 2].\\
& \leftarrow p,\ \texttt{not}\ \texttt{ws}(r_0).
\end{aligned}$$

This translation $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ still yields 44 stable as well as supported models subject to $\langle \mathrm{At}(P), e\rangle$, representing the five stable models of the modified program P. Its strong version $\mathrm{Tr}_{\mathrm{ACYC+}}(P)$ is obtained by replacing integrity constraints, given in Example 3.15, that mention either $\texttt{ws}(r_1)$ or $\texttt{ws}(r_2)$ by:

$$\begin{array}{lll}
\texttt{nxt}(r_0, 2) \leftarrow \texttt{dep}(p, s). & & \texttt{chk}(r_0, 2) \leftarrow \texttt{nxt}(r_0, 2),\ \texttt{dep}(p, t).\\
\texttt{nxt}(r_0, 3) \leftarrow \texttt{dep}(p, t). & \texttt{nxt}(r_0, 3) \leftarrow \texttt{nxt}(r_0, 2). & \texttt{chk}(r_0, 3) \leftarrow \texttt{nxt}(r_0, 3),\ \texttt{dep}(p, q).\\
\multicolumn{3}{l}{\texttt{red}(r_0) \leftarrow 2 \leq [\texttt{chk}(r_0, 2) = 1,\ \texttt{chk}(r_0, 3) = 2].}\\
\leftarrow \texttt{dep}(p, s),\ \texttt{red}(r_0). & \leftarrow \texttt{dep}(p, t),\ \texttt{red}(r_0). & \leftarrow \texttt{dep}(p, q),\ \texttt{red}(r_0).
\end{array}$$

As in Example 3.15, the addition of such rules to $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ reduces the number of stable as well as supported models to 10. In particular, redundant models of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ subject to $\langle \mathrm{At}(P), e\rangle$ based on $\texttt{dep}(p, q)$ along with $\texttt{dep}(p, s)$ or $\texttt{dep}(p, t)$ yield $\texttt{red}(r_0)$ via the rule of form (20), so that these atoms are in turn denied by constraints of the form (21). Hence, well-supports for p relying on $\texttt{dep}(p, q)$ or both $\texttt{dep}(p, s)$ and $\texttt{dep}(p, t)$ remain mutually exclusive, while selections between r_3 and r_4, r_5 and r_6, or r_7 and r_8 to provide well-support for the atoms q, s, and t reproduce the six alternatives sketched in Example 3.15 to represent the stable model $\{p, q, s, t, y\}$ of P. ■

The following result extends Theorem 3.11 to the strong acyclicity translation $\mathrm{Tr}_{\mathrm{ACYC+}}$.

Theorem 3.18. Let P be a weight constraint program, $\mathrm{Tr}_{\mathrm{ACYC+}}(P)$ its strong acyclicity translation with an acyclicity extension $\langle \mathrm{At}(P), e\rangle$, and assume that each weight rule $r \in P$ of the form (3) is such that $|\{b_1, \ldots, b_n\}| = n$, i.e., the atoms of positive weighted literals in the body of r are distinct.

1. If M is a stable model of P and, for each $a \in M$, some $D_a \in \mathcal{D}_P^M(a)$ is fixed, then $\mathrm{Tr}_{\mathrm{ACYC+}}(P)$ subject to $\langle \mathrm{At}(P), e\rangle$ has a supported model N such that

$$\begin{aligned}
N \cap \mathrm{At}(\mathrm{Tr}_{\mathrm{ACYC}}(P)) = M & \cup \{\texttt{ws}(r) \mid a \in \mathrm{At}(P), r \in \mathrm{WS}_P^M(a, \emptyset)\}\\
& \cup \{\texttt{ws}(r) \mid a \in M, r \in \mathrm{WS}_P^M(a, D_a)\}\\
& \cup \{\texttt{dep}(a, b) \mid a \in M, b \in D_a\}.
\end{aligned}$$

2. If N is a supported model of $\mathrm{Tr}_{\mathrm{ACYC+}}(P)$ subject to $\langle \mathrm{At}(P), e\rangle$, then $M = N \cap \mathrm{At}(P)$ is a stable model of P that is well-supported by $R = \{r \mid \mathtt{ws}(r) \in N, \mathrm{head}(r) \in M\}$.

Proof:
(1.) Suppose that M is a stable model of P. By Theorem 3.11, $N \cap \mathrm{At}(\mathrm{Tr}_{\mathrm{ACYC}}(P))$ is a supported model of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ subject to $\langle \mathrm{At}(P), e\rangle$. We will consider instances of the constraints (15), (16), and (21) and show by contradiction that each of them is satisfied by a supported model N of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ augmented with rules of the forms (17), (18), (19), and (20) in $\mathrm{Tr}_{\mathrm{ACYC+}}(P)$.

First, observe that, if a constraint of the form (15) is not satisfied, then $\mathtt{dep}(a, b) \in N$ and $a \notin M$. However, the definition of N guarantees that $a \in M$ if $\mathtt{dep}(a, b) \in N$, a contradiction.

Next, let us assume that an instance of (16) is not satisfied by N. Such a constraint concerns a rule $r \in \mathrm{Def}_P(a)$ such that $b \notin \mathrm{B}(r)^+$. By the assumption, $\mathtt{ws}(r) \in N$, and the definition of N implies that $a \in M$, $b \in D_a$, and $r \in \mathrm{WS}_P^M(a, D_a)$. Since $D_a \models \mathrm{B}(r)^M$ and $b \notin \mathrm{B}(r)^+$, however, $D_a \setminus \{b\} \models \mathrm{B}(r)^M$, a contradiction to the minimality of D_a.

Finally, assume that an instance of (21) is not satisfied by any extension of $N \cap \mathrm{At}(\mathrm{Tr}_{\mathrm{ACYC}}(P))$ to a supported model N of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ augmented with rules of the forms (17), (18), (19), and (20). The constraint concerns a weight rule r of the form (3) such that $\mathrm{head}(r) = a$, $|\{b_1, \ldots, b_n\}| = n$, and the weights $w_1, \ldots, w_n$ are ordered according to $w_{i-1} \leq w_i$ for each $1 < i \leq n$. Given the definition of $\mathtt{nxt}(\cdot, \cdot)$ by rules of the forms (17) and (18), by induction on $1 < i \leq n$, it follows that $\mathtt{nxt}(r, i) \in N$ is required only if there is some $j < i$ for which $\mathtt{dep}(a, b_j) \in N$, and the same applies to $\mathtt{chk}(r, i)$ defined by rules of the form (19), provided that $\mathtt{dep}(a, b_i) \in N$. Now, let r' denote the defining weight rule of $\mathtt{red}(r)$ of the form (20). By the assumption that the constraint in question of the form (21) is not satisfied by N, we have that $N \models \mathrm{B}(r')^M$ and $D_a \setminus \{b_j \mid j = \min\{i \mid b_i \in D_a\}\} \models \mathrm{B}(r)^M$. Thus, again the support for a could be obtained via r by a smaller set than D_a, a contradiction to the minimality of D_a.

We have thus established that there is a supported model N of $\mathrm{Tr}_{\mathrm{ACYC+}}(P)$ subject to $\langle \mathrm{At}(P), e\rangle$ such that $N \cap \mathrm{At}(\mathrm{Tr}_{\mathrm{ACYC}}(P))$ is a supported model of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ subject to $\langle \mathrm{At}(P), e\rangle$.
(2.) Let N be a supported model of $\mathrm{Tr}_{\mathrm{ACYC+}}(P)$ subject to $\langle \mathrm{At}(P), e\rangle$. Since $\mathrm{Tr}_{\mathrm{ACYC+}}(P)$ only extends $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ by further constraints (15), (16), and (21) and (non-recursive) definitions of new atoms by rules of the forms (17), (18), (19), and (20), we have that $N \cap \mathrm{At}(\mathrm{Tr}_{\mathrm{ACYC}}(P))$ is a supported model of $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ subject to $\langle \mathrm{At}(P), e\rangle$. Theorem 3.11 thus yields that $M = N \cap \mathrm{At}(P)$ is well-supported by $R = \{r \mid \mathtt{ws}(r) \in N, \mathrm{head}(r) \in M\} = \{r \mid \mathtt{ws}(r) \in N \cap \mathrm{At}(\mathrm{Tr}_{\mathrm{ACYC}}(P)), \mathrm{head}(r) \in M\}$, so that M is a stable model of P. □

Given that $\mathrm{head}(\mathrm{Tr}_{\mathrm{ACYC+}}(P) \setminus \mathrm{Tr}_{\mathrm{ACYC}}(P)) \cap \mathrm{At}(\mathrm{Tr}_{\mathrm{ACYC}}(P)) = \emptyset$ along with the fact that the subprogram $\mathrm{Tr}_{\mathrm{ACYC+}}(P) \setminus \mathrm{Tr}_{\mathrm{ACYC}}(P)$ is tight, the splitting set theorem yields a direct extension of Proposition 3.12 to $\mathrm{Tr}_{\mathrm{ACYC+}}$.

Corollary 3.19. Let P be a weight constraint program, $\mathrm{Tr}_{\mathrm{ACYC+}}(P)$ its strong acyclicity translation with an acyclicity extension $\langle \mathrm{At}(P), e\rangle$, and $M \subseteq \mathrm{At}(\mathrm{Tr}_{\mathrm{ACYC+}}(P))$ an interpretation. Then, M is a supported model of $\mathrm{Tr}_{\mathrm{ACYC+}}(P)$ subject to $\langle \mathrm{At}(P), e\rangle$ iff M is a stable model of $\mathrm{Tr}_{\mathrm{ACYC+}}(P)$ subject to $\langle \mathrm{At}(P), e\rangle$.

The translations $\mathrm{Tr}_{\mathrm{ACYC}}$ and $\mathrm{Tr}_{\mathrm{ACYC+}}$ can be adjusted to take SCCs into account and, in practice, their *component-aware* versions are implemented. Definitions 3.8, 3.14, and 3.16 require the following revisions to incorporate SCCs. Given an atom $a \in \mathrm{At}(P)$ and the component $\mathrm{SCC}_P(a)$, the atoms

$\mathtt{dep}(a, b_i)$ in the rules (12) and (13) are replaced by b_i if $b_i \notin \mathrm{SCC}_P(a)$. Moreover, rules of the forms (11), (15), (16), and (21) are only needed if $b \in \mathrm{SCC}_P(a)$. In Definition 3.16, the condition for ordering the weight rule is refined such that, for some $0 \leq j \leq n$, $\{b_1, \ldots, b_j\} \subseteq \mathrm{SCC}_P(a)$, $\{b_{j+1}, \ldots, b_n\} \cap \mathrm{SCC}_P(a) = \emptyset$, and $w_{i-1} \leq w_i$ for each $1 < i \leq j$. Then, the rules (17) and (19) are restricted to $1 < i \leq j$, and the rule (18) to $1 < i < j$. Finally, the atoms $\mathtt{chk}(r, i)$ in the rule (20) are replaced by b_i for $j < i \leq n$. In view of the module theorem, the relationships among models established in Theorems 3.11 and 3.18, Proposition 3.12, and Corollary 3.19 remain valid for the component-aware versions of $\mathrm{Tr}_{\mathrm{ACYC}}$ and $\mathrm{Tr}_{\mathrm{ACYC+}}$, which restrict $\mathtt{dep}(a, b)$ atoms to $\langle a, b\rangle \in \mathrm{DG}^+(P)$ such that $\mathrm{SCC}_P(a) = \mathrm{SCC}_P(b)$.

Example 3.20. For P from Example 3.7 and $a \in \{p, q, s, t\}$, we have that $\mathrm{SCC}_P(a) = \{p, q, s, t\}$. Hence, the component-aware version of $\mathrm{Tr}_{\mathrm{ACYC+}}(P)$ replaces the atoms $\mathtt{dep}(q, y)$, $\mathtt{dep}(s, y)$, and $\mathtt{dep}(t, y)$ in the rules defining the heads $\mathtt{ws}(r_4)$, $\mathtt{ws}(r_6)$, and $\mathtt{ws}(r_8)$, given in Example 3.13, by y, and also drops choice rules for such atoms $\mathtt{dep}(a, y)$, so that the corresponding edges no longer contribute to the acyclicity extension $\langle \mathrm{At}(P), e\rangle$ determined by $e(\mathtt{dep}(a, b)) = \langle a, b\rangle$. Moreover, among the constraints shown in Example 3.15, those mentioning some of the three obsolete $\mathtt{dep}(a, y)$ atoms are simply dropped. Then, all stable as well as supported models extending $\{p, q, s, t, y\}$ for the component-aware version of $\mathrm{Tr}_{\mathrm{ACYC+}}(P)$ include $\mathtt{ws}(r_4)$, $\mathtt{ws}(r_6)$, and $\mathtt{ws}(r_8)$, given that $\mathtt{dep}(q, y)$, $\mathtt{dep}(s, y)$, and $\mathtt{dep}(t, y)$ are not needed as prerequisites anymore. As a consequence, the constraints $\leftarrow \mathtt{dep}(a, p), \mathtt{ws}(r)$ for $a \in \{q, s, t\}$ along with a corresponding rule $r \in \{r_4, r_6, r_8\}$ suppress edges associated with $\mathtt{dep}(a, p)$. Since the latter were still admissible in Example 3.15, the component-aware translation further reduces the number of stable as well as supported models extending $\{p, q, s, t, y\}$ from six to two. The two remaining options differ in whether r_1 or r_2 is used as well-support for p, as reflected by the atoms $\mathtt{dep}(p, q)$ and $\mathtt{ws}(r_1)$ or $\mathtt{dep}(p, s)$, $\mathtt{dep}(p, t)$, and $\mathtt{ws}(r_2)$, respectively. Similar cases for $\mathtt{dep}(p, b)$ atoms with $b \in \{q, s, t\}$ are obtained when r_1 and r_2 are replaced by r_0 according to Example 3.17, where both possibilities agree on $\mathtt{ws}(r_0)$. Either way, along with unique extensions of $\{x\}$, $\{y\}$, $\{s, y\}$, and $\{t, y\}$, the component-aware version of $\mathrm{Tr}_{\mathrm{ACYC+}}(P)$ yields six stable as well as supported models subject to $\langle \mathrm{At}(P), e\rangle$, capturing the five stable models of P given in Example 3.7. ■

4. Experiments

Acyclicity programs are implemented in the development version 3.2.0-R47179 of CLASP [17], using a propagator for acyclicity reasoning similar to the one introduced in the SAT modulo acyclicity solver ACYCGLUCOSE [10],[2] and will be supported from the forthcoming release 3.2.0 on. On the one hand, a program may define the dedicated predicate $\mathtt{\_edge}(v, u)$ to declare the edges $e(\mathtt{\_edge}(v, u)) = \langle v, u\rangle$ in an associated acyclicity extension. For instance, the rule

$$\mathtt{\_edge}(v, u) \leftarrow \mathtt{hc}(v, u),\ v \neq v_0,\ u \neq v_0.$$

specifies the acyclicity extension for the Hamiltonian Cycle encoding consisting of (4) and (5) in Example 3.3. On the other hand, the tool LP2ACYC [12] implements $\mathrm{Tr}_{\mathrm{ACYC}}$ as well as $\mathrm{Tr}_{\mathrm{ACYC+}}$, and thus allows for capturing stable models by supported models subject to an acyclicity extension. Hence,

[2]The maintenance of ACYCMINISAT has been discontinued, as its performance is usually dominated in view of the more recent base SAT solver of ACYCGLUCOSE.

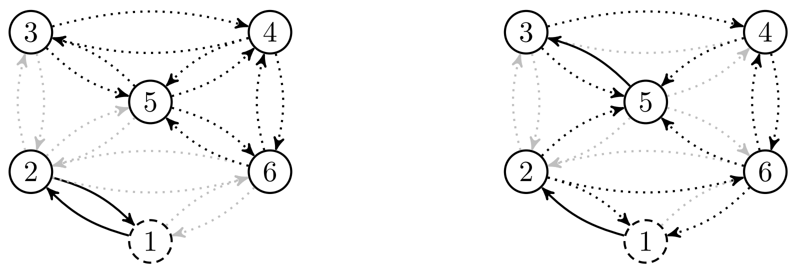


Figure 1. Example graph for Hamiltonian Cycle, where solid edges indicate respective atoms assigned to true.

program completion [27], performed by LP2SAT [23], is sufficient to map acyclicity programs (without weight rules) obtained by either translation to SAT modulo acyclicity.[3] We make use of this for comparing CLASP to ACYCGLUCOSE, version R845.[4]

We consider three benchmark classes, Hamiltonian Cycle, Labyrinth, and Sokoban, with instances stemming from substantial collections utilized in the literature [28, 29, 30]. These benchmarks involve crucial reachability conditions, which can be expressed in terms of acyclicity. For Hamiltonian Cycle, we use a linear number of *normal* rules, corresponding to (4) and (5) in Example 3.3, for enabling a direct mapping to SAT modulo acyclicity without requiring an additional normalization step (cf. [31]). Such direct mappings via completion are also applicable to Labyrinth and Sokoban, two planning problems in which, at each point in time, some reached location must have a path to another location (the starting position for pushing a box or some grid tile to explore). In both benchmarks, the reached locations can be inferred at each time point and, assuming that they are provided by a predicate $\texttt{reached}(v)$, edge atoms $\texttt{\_edge}(v,u)$ are traced back to them by means of rules as follows:

$$\begin{aligned}
\texttt{succ}(u) &\leftarrow \texttt{\_edge}(v,u).\\
\texttt{pred}(v) &\leftarrow \texttt{\_edge}(v,u).\\
&\leftarrow \texttt{pred}(v), \texttt{not succ}(v), \texttt{not reached}(v).
\end{aligned}$$

In view of the acyclicity constraint on the graph induced by true $\texttt{\_edge}(v,u)$ atoms, which can be picked via choice rules, the integrity constraint enforces any path to start from a location given by $\texttt{reached}(v)$, so that reachability is guaranteed and continued progressively over time points.

Along with further problem-specific conditions (for Labyrinth and Sokoban), the encoding parts described so far yield *tight acyclicity programs* without weight rules. In order to contrast and combine acyclicity checking with traditional unfounded set checking (cf. [16]), we further augment the encodings with a *non-tight module implementing acyclicity* for the mapping $e(\texttt{\_edge}(v,u)) = \langle v,u\rangle$ in standard ASP according to Definition 3.4. Given that acyclicity and unfounded set checking can be selectively (de)activated in CLASP, this allows us to apply either or both kinds of propagation to common inputs. Moreover, the non-tight ASP module can be processed via the translations $\mathrm{Tr}_{\mathrm{ACYC}}$ and $\mathrm{Tr}_{\mathrm{ACYC+}}$ to obtain acyclicity programs whose completion can be passed to CLASP or ACYCGLUCOSE, thus enabling comparisons between both the available translations as well as SAT modulo acyclicity solvers.

[3]The translators are available at: `http://research.ics.aalto.fi/software/asp/lp2acyc/`
[4]Binaries and benchmarks are available at: `http://www.cs.uni-potsdam.de/clasp/?page=experiments`

On the one hand, the different solving approaches can be classified in terms of the technique used to deal with *unfounded sets*, where the options are exclusive and below abbreviated as follows: 'U' indicates traditional unfounded set checking; 'T' stands for the translation $\mathrm{Tr}_{\mathrm{ACYC}}$; 'T+' refers to its strong version $\mathrm{Tr}_{\mathrm{ACYC+}}$; and '_' expresses that neither of the former means is applied. In addition, the strength of *acyclicity propagation* can be varied: 'A' represents plain acyclicity checking, detecting a conflict whenever the graph given by true atoms associated with edges is cyclic; 'B' denotes acyclicity checking enhanced by "backward" inference of forbidden edges, i.e., potential edges that would close a cycle are identified in order to falsify the corresponding yet unassigned atoms; and '_' again means that acyclicity propagation is not performed at all. Given the options in these two orthogonal dimensions, the resulting solver variants are summarized in Table 1 and further discussed in the following.

First of all, note that the propagation mechanisms for unfounded sets or acyclicity, respectively, are indeed complementary. To see this, consider the task of finding a Hamiltonian cycle through the example graph in Figure 1, whose node 1 serves as initial node through which a cycle is admitted. The situation on the left visualizes a search state in which atoms associated with the edges $\langle 1, 2\rangle$ and $\langle 2, 1\rangle$ are assigned to true, so that atoms labeling other outgoing or incoming edges of the nodes 1 and 2 are falsified in view of mutual exclusions. Hence, the subgraph induced by the nodes 3, 4, 5, and 6 is separated from node 1, and unfounded set checking detects a contradiction to the reachability requirement for these nodes. The cycle through node 1 is, however, exempt from acyclicity checking, while the inherent necessity of some cycle among the nodes 3, 4, 5, and 6 follows by a counting argument that is not explored by acyclicity propagation, so that the search has to keep on guessing edges to successively discard individual cycles. Turning to the situation on the right, atoms labeling the edges $\langle 1, 2\rangle$ and $\langle 5, 3\rangle$ are assigned to true, and those of respective excluded edges falsified. Since all nodes remain potentially reachable from the initial node 1, unfounded set checking cannot propagate anything here, while backward inference allows for identifying forbidden edges, whose associated atoms have to be falsified to avoid cycles. In fact, this applies to the edge $\langle 3, 5\rangle$, so that $\langle 3, 4\rangle$ is left as the only outgoing edge of node 3 that can and must be on a Hamiltonian cycle. Iterating the falsification of forbidden edges by acyclicity propagation and in turn deriving necessary edges via basic (unit) propagation eventually yields the only Hamiltonian cycle including the edges $\langle 1, 2\rangle$ and $\langle 5, 3\rangle$: $\langle 1, 2, 5, 3, 4, 6, 1\rangle$. That is, acyclicity propagation with backward inference omits (possibly wrong) guesses that are otherwise needed with unfounded set checking only.

The target formalism, ASP or SAT modulo acyclicity, predetermines the applicable combinations of techniques along with respective solvers. While omitting both unfounded set checking and acyclicity propagation would be unsound, running CLASP as standard ASP solver constitutes a natural baseline. In the following, we refer to this solver variant by CLASP[U,_], given that it performs unfounded set checking but no acyclicity propagation. Possible extensions by either of the two kinds of acyclicity propagation are denoted by CLASP[U,A] and CLASP[U,B], distinguishing whether backward inference is applied or not. Interestingly, the non-tight module specified in Definition 3.4 yields a non-trivial unfounded set iff the graph induced by true atoms associated with edges includes a cycle. As a consequence, CLASP[U,_] and CLASP[U,A] detect exactly the same conflicts. We experimentally verified their identical behavior in terms of conflicts as well as comparable runtimes, and skip redundant results for CLASP[U,A] below. Unlike that, backward inference complements unfounded set checking in CLASP[U,B] and yields a strictly stronger propagation. This also applies to the solver variant CLASP[_,B] in relation to CLASP[_,A], both of which omit unfounded set checking and solve acyclicity programs by means of acyclicity propagation.

Mapping acyclicity programs to SAT modulo acyclicity allows for comparing the solvers CLASP and ACYCGLUCOSE. For both of them, the underlying translation as well as the kind of acyclicity propagation

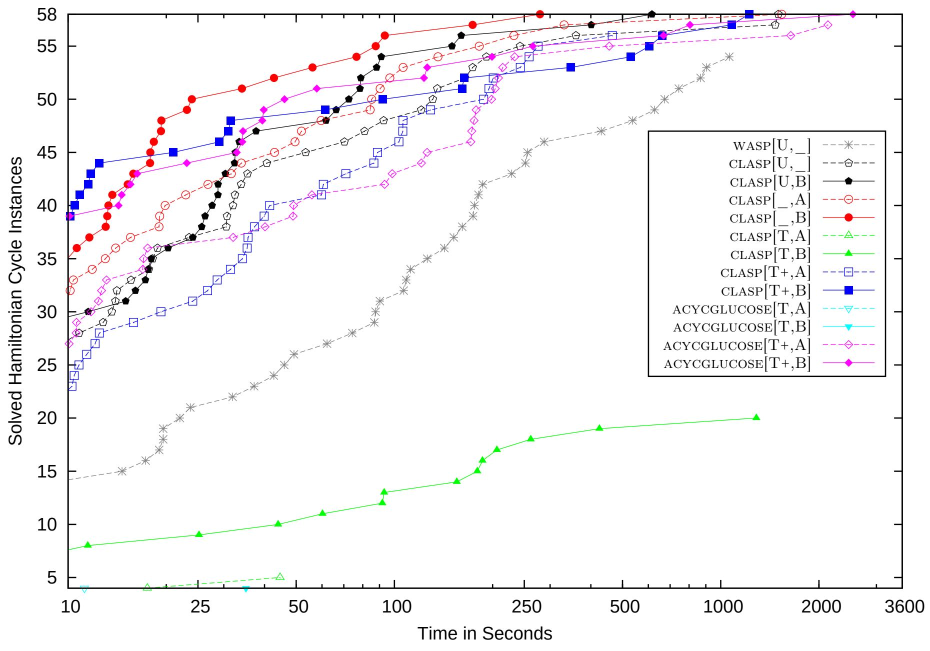
	Nothing ('_')	Acyclicity ('A')	Backward ('B')	
Nothing ('_')	—	CLASP[_,A]	CLASP[_,B]	ASP
Unfounded set ('U')	CLASP[U,_] WASP[U,_]	(CLASP[U,A])	CLASP[U,B]	ASP
$\mathrm{Tr}_{\mathrm{ACYC}}$ ('T')	—	CLASP[T,A] ACYCGLUCOSE[T,A]	CLASP[T,B] ACYCGLUCOSE[T,B]	SAT
$\mathrm{Tr}_{\mathrm{ACYC+}}$ ('T+')	—	CLASP[T+,A] ACYCGLUCOSE[T+,A]	CLASP[T+,B] ACYCGLUCOSE[T+,B]	SAT

Table 1. Overview of solver variants using different techniques for dealing with unfounded sets (listed vertically) as well as acyclicity conditions (listed horizontally). Missing entries, indicated by '—', refer to unsound combinations of techniques, for which no experimental results are reported below.

can be picked, yielding four variants per solver, indicated by the suffixes '[T,A]', '[T,B]', '[T+,A]', and '[T+,B]' below. We used the "trendy" configuration, which is CLASP's default for SAT (modulo acyclicity) inputs and likewise applicable to acyclicity programs, for all CLASP variants. For comparison, we also include the standard ASP solver WASP [32], version 2.0, denoted by WASP[U,_] below, as it performs unfounded set checking yet no acyclicity propagation. All solvers were run single-threaded on a cluster of Linux machines equipped with Intel Ten-Core Xeon E5-2680 2.80GHz processors, imposing a limit of 3,600 seconds (one hour) wall-clock time and 16GB memory limit per run.[5]

Figures 2–4 plot numbers of solved instances in terms of runtime in seconds as well as conflicts. On the 58 Hamiltonian Cycle instances in Figure 2, each satisfiable and successfully handled by some of the considered solver variants, it is apparent that the basic translation $\mathrm{Tr}_{\mathrm{ACYC}}$, applied for SAT modulo acyclicity solvers indicated by abbreviation 'T', leads to increased difficulty. The use of backward inference compensates this to some extent in CLASP[T,B], but it does not help ACYCGLUCOSE[T,B]. In fact, ACYCGLUCOSE[T,B] suffers from eleven memory outs here, while memory is not an issue for other solver variants or on other benchmarks. This phenomenon is due to the large size of graphs in some of the instances (up to about 700,000 potential edges), combinatorics of viable edges under $\mathrm{Tr}_{\mathrm{ACYC}}$, and implementation differences between CLASP and ACYCGLUCOSE: CLASP performs unit propagation as soon as backward inference yields some atom that must be false, but ACYCGLUCOSE extensively collects such atoms and corresponding clauses, which leads to memory pollution with ACYCGLUCOSE[T,B]. The strong translation $\mathrm{Tr}_{\mathrm{ACYC+}}$, however, results in much more robust performance of solver variants based on 'T+'. Regarding the benefits of backward inference, CLASP[T+,B] and ACYCGLUCOSE[T+,B] are both able to complete all 58 Hamiltonian Cycle instances in time, while CLASP[T+,A] and ACYCGLUCOSE[T+,A] yield outliers and fail in two cases or one, respectively. A similar observation applies to the ASP variant CLASP[_,B] in relation to its counterpart CLASP[_,A], and CLASP[U,B] tends to require less solving time than CLASP[U,_], which performs unfounded set checking only. In particular, the conflicts in the lower part of Figure 2 exhibit quite consistent search reductions due to backward inference on

[5]Running also the four other benchmark classes considered in preliminary experiments, reported in the workshop version of this paper [1], with common ASP modulo acyclicity encodings showed that the performance on these benchmarks is largely dominated by side constraints or the combinatorics of optimization, respectively, rather than the use of different techniques to deal with acyclicity conditions.



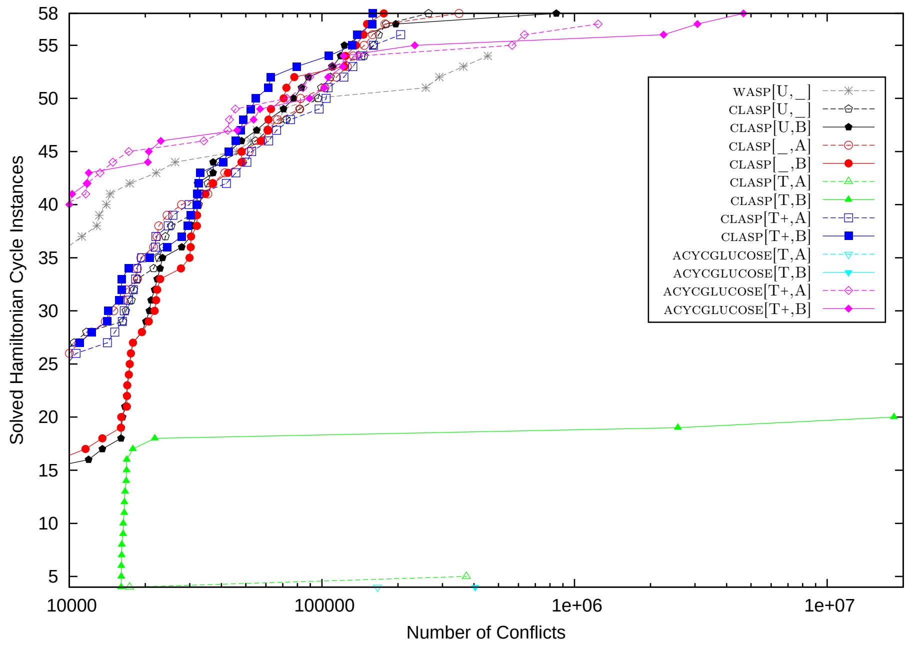
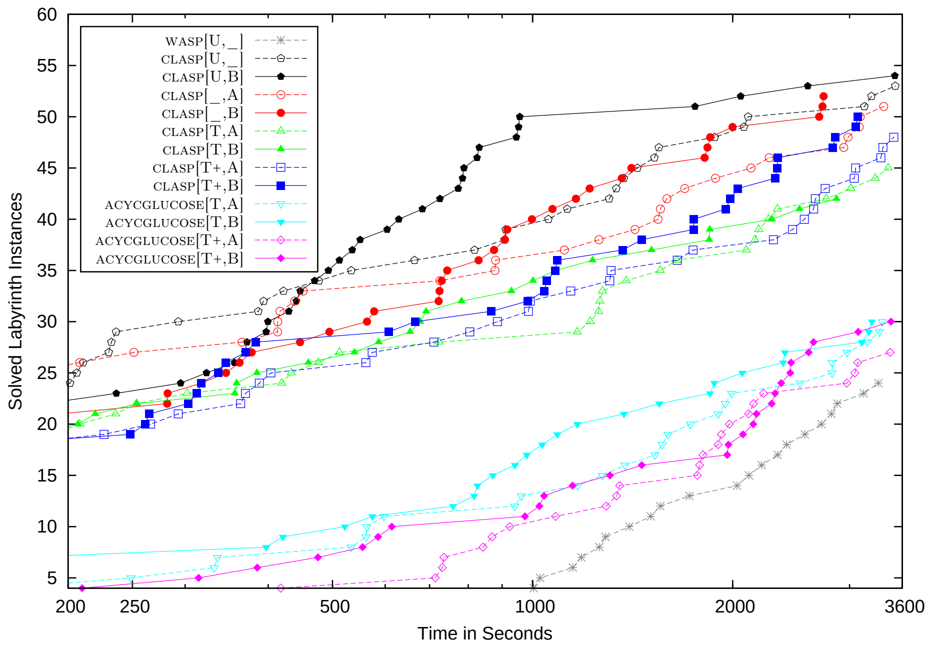


Figure 2. Comparison of CLASP and ACYCGLUCOSE on 58 Hamiltonian Cycle instances, varying the use of unfounded set checking ('U'), translation Tr_{ACYC} ('T'), Tr_{ACYC+} ('T+'), or neither of them ('_'), along with acyclicity checking disabled ('_'), enabled ('A'), or enhanced by backward inference ('B').

easy instances, where the number of conflicts stays within a few thousands. For more difficult instances, the interplay with search heuristics is less straightforward, e.g., the maximum number of conflicts encountered by CLASP[U,B] is greater than with CLASP[U,_]. Likewise, WASP[U,_] performs comparable to CLASP[U,_] in terms of conflicts, where it has some advantage on easy instances, yet this does not pay off in runtime, and four instances remain uncompleted by WASP[U,_] within the time limit. More globally, the proximity between the ASP variants of CLASP and SAT modulo acyclicity solvers based on 'T+' shows that avoiding redundant edges via $\mathrm{Tr}_{\mathrm{ACYC+}}$ can constitute an appropriate substitute for unfounded set checking.

Turning to the results on 60 satisfiable Labyrinth instances displayed in Figure 3, we observe a substantial gap between the variants of CLASP and ACYCGLUCOSE. In fact, each of the instances is solved in time by at least one CLASP variant (57 of them when considering the variants based on 'T' or 'T+'), while the four ACYCGLUCOSE variants taken together succeed on 37 instances only. Regarding the latter, backward inference turns out to be helpful for ACYCGLUCOSE[T,B] and ACYCGLUCOSE[T+,B] in comparison to their respective counterparts ACYCGLUCOSE[T,A] and ACYCGLUCOSE[T+,A]. Somewhat surprisingly, the use of translation $\mathrm{Tr}_{\mathrm{ACYC}}$ with ACYCGLUCOSE[T,B] or its strong version $\mathrm{Tr}_{\mathrm{ACYC+}}$ with ACYCGLUCOSE[T+,B] does not yield any major impact, and both lead to 30 instances solved in time. This is quite different from CLASP, whose variant CLASP[T+,B] solves 50 instances and thus eight more than CLASP[T,B]. In fact, CLASP[T,B] solves the fewest instances in time among all CLASP variants, and even CLASP[T,A], skipping backward inference with the same translation, is able to complete three instances more. However, backward inference brings about search reductions, especially regarding difficult instances leading to many conflicts, with CLASP[T+,B] as well as the ASP variants CLASP[_,B] and CLASP[U,B]. Notably, the most successful variants are based on traditional unfounded set checking, abbreviated by 'U', where backward inference in CLASP[U,B] improves on CLASP[U,_] in terms of both time and conflicts as well as one more solved instance. The decent performance of CLASP[U,_] does, however, not carry over to the other standard ASP solver, WASP[U,_], which (in default settings) completes only 24 instances in time. In fact, the lower part of Figure 3 exhibits that WASP[U,_] encounters more conflicts than each of the CLASP variants, yet still fewer than those of ACYCGLUCOSE, while the latter advantage does not amortize in runtime.

The 52 instances of Sokoban, with plots shown in Figure 4, are picked around minimum plan lengths such that exactly half of the instances are satisfiable. The most apparent observation is that, in contrast to Labyrinth before, the variants of ACYCGLUCOSE perform significantly more robustly than those of CLASP. While each instance is solved by some ACYCGLUCOSE variant, all CLASP variants fail on ten instances, four of which are satisfiable and six unsatisfiable. Also opposite to the performance on Labyrinth, the combination of unfounded set checking with backward inference in CLASP[U,B] turns out to be worst. Interestingly, omitting backward inference leads to slight improvements with CLASP[U,_], yet CLASP[_,B] benefits even more from abandoning unfounded set checking. Positive effects due to backward inference are also confirmed by CLASP[T+,B], which completes four instances more than CLASP[T+,A], skipping backward inference with the strong translation $\mathrm{Tr}_{\mathrm{ACYC+}}$. As already observed on Labyrinth, backward inference becomes ineffective when switching to $\mathrm{Tr}_{\mathrm{ACYC}}$ for CLASP variants based on 'T'. However, the basic translation $\mathrm{Tr}_{\mathrm{ACYC}}$ seems generally not a good choice for CLASP, while its strong version $\mathrm{Tr}_{\mathrm{ACYC+}}$ results in much more robust performance, especially together with backward inference. The behavior of ACYCGLUCOSE, which outperforms the CLASP variants on Sokoban, primarily benefits from backward inference, leading to substantial improvements and somewhat equalizing differences between the translations $\mathrm{Tr}_{\mathrm{ACYC}}$ and $\mathrm{Tr}_{\mathrm{ACYC+}}$. In terms of conflicts given in the lower part of Figure 4, WASP[U,_] is close to the ACYCGLUCOSE variants and ahead of CLASP[U,_] as well as other CLASP variants, but runtimes again do not reflect this. Rather, WASP[U,_] does not complete nine



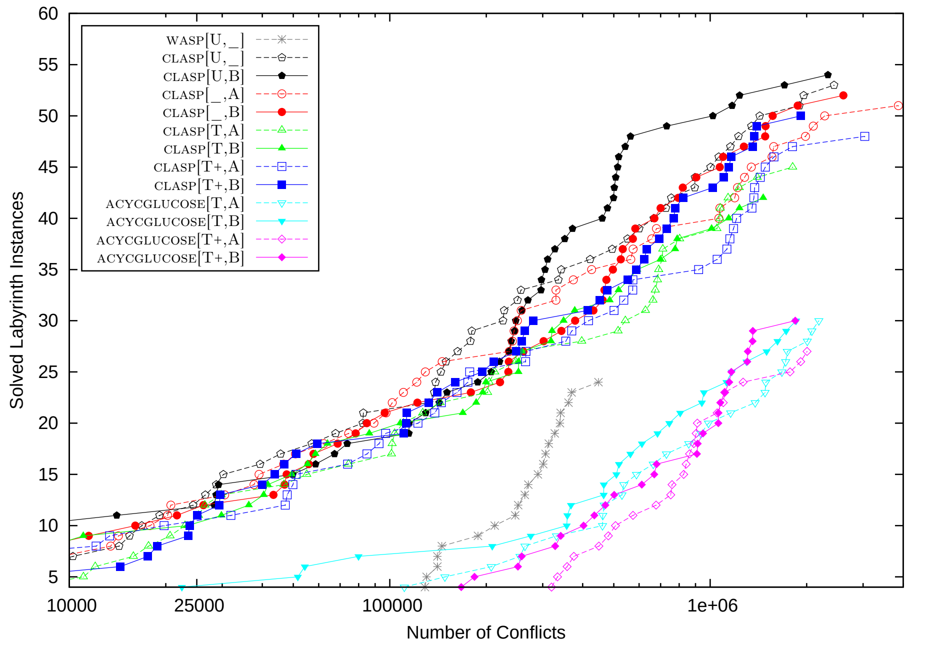
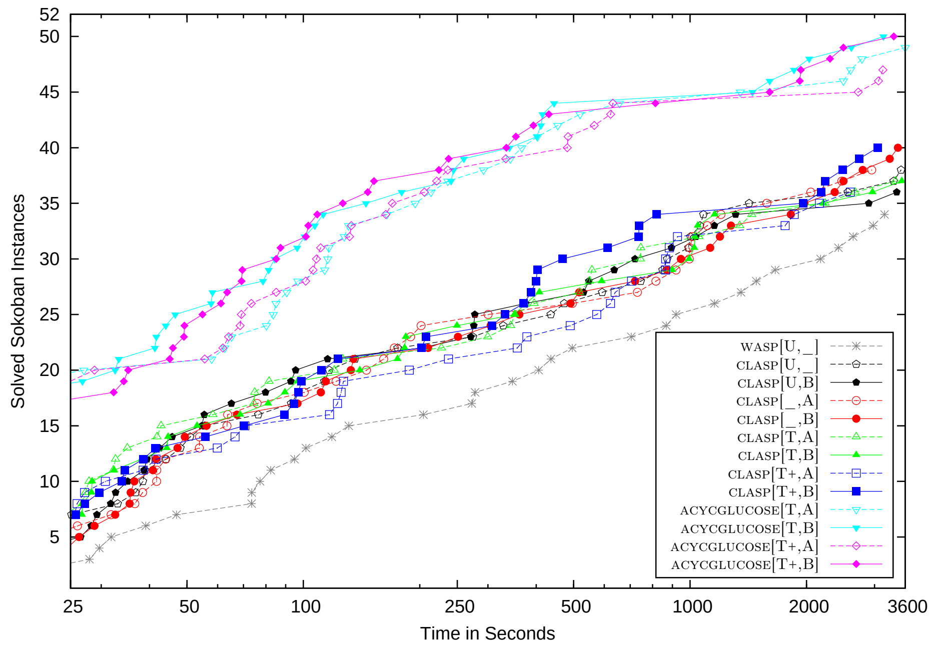


Figure 3. Comparison of CLASP and ACYCGLUCOSE variants as in Figure 2 on 60 Labyrinth instances.

satisfiable and nine unsatisfiable instances, including the ten on which all CLASP variants fail as well.

In summary, solving approaches including acyclicity propagation can be competitive to traditional unfounded set checking or complement it in an effective manner. Most importantly, backward inference to falsify atoms associated with edges appears to be a useful addition, given the lack of (efficient)



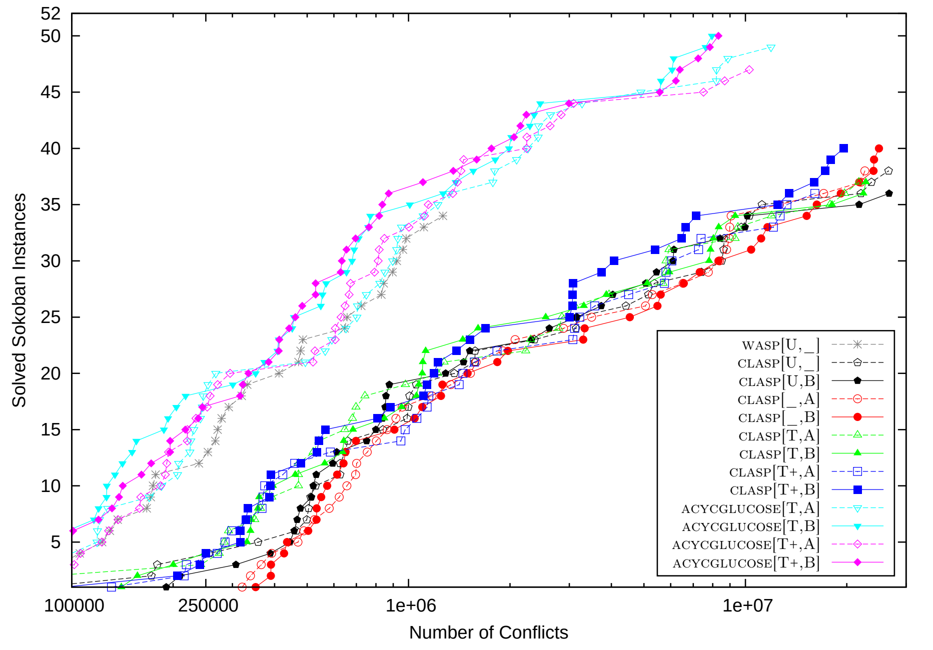


Figure 4. Comparison of CLASP and ACYCGLUCOSE variants as in Figure 2 on 52 Sokoban instances.

implementations of corresponding principles for unfounded sets. As witnessed by Sokoban, mapping logic programs (without weight constraints) to acyclicity programs via the strong translation $\mathrm{Tr}_{\mathrm{ACYC+}}$, along with completion, is a worthwhile approach to utilize solvers for SAT modulo acyclicity. That is, potential use cases of acyclicity programs are twofold. On the one hand, the native support for an acyclicity constraint enriches the spectrum of available modeling constructs, which allows for expressing particular problems, e.g., Hamiltonian Cycle, even more compactly than anyway in standard ASP. On the other hand, the computational mechanisms for acyclicity propagation provide complementary techniques for ASP solving, where implementations of ASP or SAT modulo acyclicity can be applied via likewise compact translations, up to extensions like weight rules or optimization statements.

5. Discussion

In this paper, we propose a novel SMT-style extension of ASP by explicit acyclicity constraints in analogy to SAT modulo acyclicity [10]. These kinds of constraints have not been directly addressed in previous SMT-style extensions of ASP [33, 34, 35]. The new extension, herein coined ASP modulo acyclicity, offers a unique set of primitives for applications involving DAGs or tree structures. While other extensions of ASP, such as DLVHEX [36] and CASP [37, 38], could be used to express acyclicity constraints as well, these approaches are technically so different that a performance comparison with systems having native acyclicity propagators would not make much sense. Moreover, the current work studies acyclicity propagators as an alternative to unfounded set checking, and turning off such checks in favor of other extensions does not seem straightforward either.

The fact that unfounded set checking can be captured through the embedding of ASP into itself (Definitions 3.8, 3.14, and 3.16 accompanied by Theorems 3.11 and 3.18) forms perhaps the most interesting application of our results. The notion of well-supporting rules utilized in this paper resembles *source pointers* [16], used in native ASP solvers to record rules justifying true atoms. Although mutual simulations between acyclicity and unfounded set checking are feasible, from release 3.2.0 onward, CLASP [17] will include the acyclicity propagator as a first-class citizen. Due to an orthogonal implementation, all other features of CLASP remain at users' disposal. For instance, it is possible to perform enumeration and optimization, not supported by ACYCMINISAT and ACYCGLUCOSE [10]. Upon enumeration, the potential replication of stable (and supported) models due to guesses made about edges representing dependencies, as introduced by our translations from ASP to itself, can be avoided by means of the projection capabilities of CLASP [39].

A distinguishing feature of acyclicity propagation is the availability of backward inference of forbidden edges, which is achieved by means of a light-weight extension to acyclicity checking, and its overhead has been reported to be uncritical regardless of the size of acyclicity extensions [10]. While a corresponding inference mechanism based on unfounded sets is also simple at the conceptual level [40], no linear implementation is known and existing approaches are either of quadratic time complexity [41] or incomplete [42]. Thus, the ease of utilizing backward inference can be considered a potential advantage of propagators for acyclicity, and in our experiments the additional inferences turned out to be helpful to make the underlying search procedure more robust. In fact, the possibility to infer truth values for yet unassigned atoms is based on a tight integration of acyclicity propagation into search. Such an integration goes beyond so-called CEGAR approaches that add constraints to suppress unintended total assignments. For instance, a respective method for the Hamiltonian Cycle problem [28] feeds a SAT solver with clauses denying cycles via the edges associated with labeling variables in a putative model.

In addition to the support for acyclicity as a complementary modeling construct in ASP, yet another contribution of this work is the implementation of translators performing $\mathrm{Tr}_{\mathrm{ACYC}}$ and $\mathrm{Tr}_{\mathrm{ACYC}+}$ as formally elaborated in Section 3. The version 1.29 of LP2ACYC implements these translations relative to the

SCCs of an input program. Given that the translations cover extended rule types, weight constraint programs output by the grounder GRINGO [21] can be readily processed. In fact, the translations into ASP modulo acyclicity serve as intermediate representations when compiling logic programs into a variety of target formalisms [30, 43, 34]. The idea of *translation-based* ASP builds on such compilations and the availability of solver technology from neighboring fields that can be harnessed to search for answer sets. For instance, under the assumption that a logic program at hand contains no weight rules, the final translation to SAT modulo acyclicity, first described in [12], amounts to simple program completion. Other target formalisms, such as bit-vector logic, mixed integer programming, and pseudo-Boolean constraints, are equipped with constructs providing an efficient representation of weight rules. Using ASP modulo acyclicity as an intermediate representation, the corresponding translations are feasible with back-end translators [12] that take the features of specific target formalisms into account. Due to analogies to traditional compilation, we use the term *cross-translation* for this methodology, where a target representation is decided at the very last translation step. It is even possible to convey further kinds of primitives in such an approach, as long as they are visible in the intermediate representation (cf. [34]).

Acknowledgments. This work was funded by the Academy of Finland (251170), DFG (SCHA 550/8 and 550/9), as well as DAAD and the Academy of Finland (57071677/279121). We also acknowledge the computational resources provided by the Aalto Science-IT project.

References

[1] Bomanson J, Gebser M, Janhunen T, Kaufmann B, Schaub T. Answer Set Programming Modulo Acyclicity. In: Inclezan D, Maratea M, editors. Proceedings of the Eighth Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'15); 2015. Available from: `https://sites.google.com/site/aspocp2015/ASPOCP2015paper5.pdf`.

[2] Bomanson J, Gebser M, Janhunen T, Kaufmann B, Schaub T. Answer Set Programming Modulo Acyclicity. In: Calimeri F, Ianni G, Truszczyński M, editors. Proceedings of the Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'15). Springer-Verlag; 2015. p. 143–150. doi:10.1007/978-3-319-23264-5_13.

[3] Cussens J. Bayesian Network Learning with Cutting Planes. In: Cozman F, Pfeffer A, editors. Proceedings of the Twenty-seventh International Conference on Uncertainty in Artificial Intelligence (UAI'11). AUAI Press; 2011. p. 153–160. Available from: `https://dslpitt.org/papers/11/p153-cussens.pdf`.

[4] Corander J, Janhunen T, Rintanen J, Nyman H, Pensar J. Learning Chordal Markov Networks by Constraint Satisfaction. In: Burges C, Bottou L, Ghahramani Z, Weinberger K, editors. Proceedings of the Twenty-seventh Annual Conference on Neural Information Processing Systems (NIPS'13). Neural Information Processing Systems Foundation; 2013. p. 1349–1357. Available from: `http://papers.nips.cc/paper/4960-learning-chordal-markov-networks-by-constraint-satisfaction.pdf`.

[5] Janhunen T, Gebser M, Rintanen J, Nyman H, Pensar J, Corander J. Learning Discrete Decomposable Graphical Models via Constraint Optimization. Statistics and Computing. 2015;advance access. doi:10.1007/s11222-015-9611-4.

[6] Erdem E, Lifschitz V, Wong M. Wire Routing and Satisfiability Planning. In: Lloyd J, Dahl V, Furbach U, Kerber M, Lau K, Palamidessi C, Pereira L, Sagiv Y, Stuckey P, editors. Proceedings of the First International Conference on Computational Logic (CL'00). Springer-Verlag; 2000. p. 822–836. doi:10.1007/3-540-44957-4_55.

[7] Koponen L, Oikarinen E, Janhunen T, Säilä L. Optimizing Phylogenetic Supertrees using Answer Set Programming. Theory and Practice of Logic Programming. 2015;15(4-5):604–619. doi:10.1017/S1471068415000265.

[8] Barrett C, Sebastiani R, Seshia S, Tinelli C. Satisfiability Modulo Theories. In: Biere A, Heule M, van Maaren H, Walsh T, editors. Handbook of Satisfiability. IOS Press; 2009. p. 825–885. doi:10.3233/978-1-58603-929-5-825.

[9] Biere A, Heule M, van Maaren H, Walsh T, editors. Handbook of Satisfiability. IOS Press; 2009. Available from: `http://ebooks.iospress.nl/volume/handbook-of-satisfiability`.

[10] Gebser M, Janhunen T, Rintanen J. SAT Modulo Graphs: Acyclicity. In: Fermé E, Leite J, editors. Proceedings of the Fourteenth European Conference on Logics in Artificial Intelligence (JELIA'14). Springer-Verlag; 2014. p. 137–151. doi:10.1007/978-3-319-11558-0_10.

[11] Bayless S, Bayless N, Hoos H, Hu A. SAT Modulo Monotonic Theories. In: Bonet B, Koenig S, editors. Proceedings of the Twenty-Ninth National Conference on Artificial Intelligence (AAAI'15). AAAI Press; 2015. p. 3702–3709. Available from: `http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9951`.

[12] Gebser M, Janhunen T, Rintanen J. Answer Set Programming as SAT Modulo Acyclicity. In: Schaub T, Friedrich G, O'Sullivan B, editors. Proceedings of the Twenty-first European Conference on Artificial Intelligence (ECAI'14). IOS Press; 2014. p. 351–356. doi:10.3233/978-1-61499-419-0-351.

[13] Eén N, Sörensson N. An Extensible SAT-solver. In: Giunchiglia E, Tacchella A, editors. Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03). Springer-Verlag; 2004. p. 502–518. doi:10.1007/978-3-540-24605-3_37.

[14] Audemard G, Simon L. Predicting Learnt Clauses Quality in Modern SAT Solvers. In: Boutilier C, editor. Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI'09). AAAI/MIT Press; 2009. p. 399–404. Available from: `http://ijcai.org/Proceedings/09/Papers/074.pdf`.

[15] Baral C. Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press; 2003. doi:10.1017/CBO9780511543357.

[16] Simons P, Niemelä I, Soininen T. Extending and Implementing the Stable Model Semantics. Artificial Intelligence. 2002;138(1-2):181–234. doi:10.1016/S0004-3702(02)00187-X.

[17] Gebser M, Kaufmann B, Schaub T. Conflict-Driven Answer Set Solving: From Theory to Practice. Artificial Intelligence. 2012;187-188:52–89. doi:10.1016/j.artint.2012.04.001.

[18] Van Gelder A, Ross K, Schlipf J. The Well-Founded Semantics for General Logic Programs. Journal of the ACM. 1991;38(3):620–650. doi:10.1145/116825.116838.

[19] Oikarinen E, Janhunen T. Achieving Compositionality of the Stable Model Semantics for smodels Programs. Theory and Practice of Logic Programming. 2008;8(5-6):717–761. doi:10.1017/S147106840800358X.

[20] Lifschitz V, Turner H. Splitting a Logic Program. In: Van Hentenryck P, editor. Proceedings of the Eleventh International Conference on Logic Programming (ICLP'94). MIT Press; 1994. p. 23–37. Available from: `https://www.d.umn.edu/~hudson/papers/iclp94a.pdf`.

[21] Gebser M, Kaminski R, Kaufmann B, Ostrowski M, Schaub T, Schneider M. Potassco: The Potsdam Answer Set Solving Collection. AI Communications. 2011;24(2):107–124. doi:10.3233/AIC-2011-0491.

[22] Gebser M, Janhunen T, Rintanen J. ASP Encodings of Acyclicity Properties. In: Baral C, De Giacomo G, Eiter T, editors. Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'14). AAAI Press; 2014. Available from: `http://www.aaai.org/ocs/index.php/KR/KR14/paper/view/8025`.

[23] Janhunen T. Some (In)translatability Results for Normal Logic Programs and Propositional Theories. Journal of Applied Non-Classical Logics. 2006;16(1-2):35–86. doi:10.3166/jancl.16.35-86.

[24] Niemelä I. Stable Models and Difference Logic. Annals of Mathematics and Artificial Intelligence. 2008;53(1-4):313–329. doi:10.1007/s10472-009-9118-9.

[25] Fages F. Consistency of Clark's Completion and the Existence of Stable Models. Journal of Methods of Logic in Computer Science. 1994;1:51–60. Available from: `http://lifeware.inria.fr/~fages/Papers/MLCS.ps.gz`.

[26] Erdem E, Lifschitz V. Tight Logic Programs. Theory and Practice of Logic Programming. 2003;3(4-5):499–518. doi:10.1017/S1471068403001765.

[27] Clark K. Negation as Failure. In: Gallaire H, Minker J, editors. Logic and Data Bases. Plenum Press; 1978. p. 293–322. doi:10.1007/978-1-4684-3384-5_11.

[28] Soh T, Le Berre D, Roussel S, Banbara M, Tamura N. Incremental SAT-Based Method with Native Boolean Cardinality Handling for the Hamiltonian Cycle Problem. In: Fermé E, Leite J, editors. Proceedings of the Fourteenth European Conference on Logics in Artificial Intelligence (JELIA'14). Springer-Verlag; 2014. p. 684–693. doi:10.1007/978-3-319-11558-0_52.

[29] Alviano M, Calimeri F, Charwat G, Dao-Tran M, Dodaro C, Ianni G, Krennwallner T, Kronegger M, Oetsch J, Pfandler A, Pührer J, Redl C, Ricca F, Schneider P, Schwengerer M, Spendier L, Wallner J, Xiao G. The Fourth Answer Set Programming Competition: Preliminary Report. In: Cabalar P, Son T, editors. Proceedings of the Twelfth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'13). Springer-Verlag; 2013. p. 42–53. doi:10.1007/978-3-642-40564-8_5.

[30] Janhunen T, Niemelä I, Sevalnev M. Computing Stable Models via Reductions to Difference Logic. In: Erdem E, Lin F, Schaub T, editors. Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09). Springer-Verlag; 2009. p. 142–154. doi:10.1007/978-3-642-04238-6_14.

[31] Bomanson J, Janhunen T. Normalizing Cardinality Rules using Merging and Sorting Constructions. In: Cabalar P, Son T, editors. Proceedings of the Twelfth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'13). Springer-Verlag; 2013. p. 187–199. doi:10.1007/978-3-642-40564-8_19.

[32] Alviano M, Dodaro C, Leone N, Ricca F. Advances in WASP. In: Calimeri F, Ianni G, Truszczyński M, editors. Proceedings of the Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'15). Springer-Verlag; 2015. p. 40–54. doi:10.1007/978-3-319-23264-5_5.

[33] Gebser M, Ostrowski M, Schaub T. Constraint Answer Set Solving. In: Hill P, Warren D, editors. Proceedings of the Twenty-fifth International Conference on Logic Programming (ICLP'09). Springer-Verlag; 2009. p. 235–249. doi:10.1007/978-3-642-02846-5_22.

[34] Liu G, Janhunen T, Niemelä I. Answer Set Programming via Mixed Integer Programming. In: Brewka G, Eiter T, McIlraith S, editors. Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'12). AAAI Press; 2012. p. 32–42. Available from: `http://www.aaai.org/ocs/index.php/KR/KR12/paper/view/4516`.

[35] Lee J, Meng Y. Answer Set Programming Modulo Theories and Reasoning about Continuous Changes. In: Rossi F, editor. Proceedings of the Twenty-third International Joint Conference on Artificial Intelligence (IJCAI'13). IJCAI/AAAI Press; 2013. p. 990–996. Available from: `http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6895`.

[36] Eiter T, Ianni G, Schindlauer R, Tompits H. DLVHEX: A Prover for Semantic-Web Reasoning under the Answer-Set Semantics. In: Proceedings of the International Conference on Web Intelligence (WI'06). IEEE Computer Society; 2006. p. 1073–1074. doi:10.1109/WI.2006.64.

[37] Mellarkod V, Gelfond M, Zhang Y. Integrating Answer Set Programming and Constraint Logic Programming. Annals of Mathematics and Artificial Intelligence. 2008;53(1-4):251–287. doi:10.1007/s10472-009-9116-y.

[38] Lierler Y. Relating Constraint Answer Set Programming Languages and Algorithms. Artificial Intelligence. 2014;207:1–22. doi:10.1016/j.artint.2013.10.004.

[39] Gebser M, Kaufmann B, Schaub T. Solution Enumeration for Projected Boolean Search Problems. In: van Hoeve W, Hooker J, editors. Proceedings of the Sixth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'09). Springer-Verlag; 2009. p. 71–86. doi:10.1007/978-3-642-01929-6_7.

[40] Gebser M, Schaub T. Tableau Calculi for Logic Programs under Answer Set Semantics. ACM Transactions on Computational Logic. 2013;14(2):15:1–15:40. doi:10.1145/2480759.2480767.

[41] Chen X, Ji J, Lin F. Computing Loops With at Most One External Support Rule. In: Brewka G, Lang J, editors. Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08). AAAI Press; 2008. p. 401–410. Available from: `http://www.aaai.org/Library/KR/2008/kr08-039.php`.

[42] Drescher C, Walsh T. Efficient Approximation of Well-Founded Justification and Well-Founded Domination. In: Cabalar P, Son T, editors. Proceedings of the Twelfth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'13). Springer-Verlag; 2013. p. 277–289. doi:10.1007/978-3-642-40564-8_28.

[43] Nguyen M, Janhunen T, Niemelä I. Translating Answer-Set Programs into Bit-Vector Logic. In: Tompits H, Abreu S, Oetsch J, Pührer J, Seipel D, Umeda M, Wolf A, editors. Proceedings of the Nineteenth International Conference on Applications of Declarative Programming and Knowledge Management (INAP'11) and the Twenty-fifth Workshop on Logic Programming (WLP'11). Springer-Verlag; 2013. p. 105–116. doi:10.1007/978-3-642-41524-1_6.