Answer Set Programming modulo Acyclicity *

Jori Bomanson[1], Martin Gebser[1,2], Tomi Janhunen[1],
Benjamin Kaufmann[2], and Torsten Schaub[2,3]**

[1] Aalto University, HIIT
[2] University of Potsdam
[3] INRIA Rennes

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. The implementation, obtained as an extension to the state-of-the-art answer set solver CLASP, provides a unique combination of traditional unfounded set checking with acyclicity propagation. The interplay of these orthogonal checks is experimentally evaluated by equipping logic programs with supplementary acyclicity constraints.

1 Introduction

Acyclic data structures such as DAGs and trees occur frequently in applications. For instance, Bayesian [9] and Markov [8] network learning as well as Circuit layout [13] 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 [4] for extending Boolean satisfiability in terms of graph-theoretic properties [19, 5]. The idea of *satisfiability modulo acyclicity* [17] is to view Boolean variables as conditionalized 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 contemporary CDCL-based *satisfiability* (SAT) solvers [6], MINISAT [11] and GLUCOSE [2], 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) [3], featuring a rule-based language for knowledge representation. While SAT solvers with explicit acyclicity constraints offer an alternative mechanism to implement ASP via appropriate translations into SAT modulo acyclicity [17], the goal of this paper is different: the idea is to incorporate acyclicity constraints into ASP, thus accounting for extended rule types [27] as well as reasoning tasks like enumeration and optimization. The resulting formalism, *ASP modulo acyclicity*, offers a rich set

* A short version of this paper will appear at LPNMR'15.
** Affiliated with Simon Fraser University, Canada, and IIIS Griffith University, Australia.

of primitives to express constraints related to recursive structures. The implementation, obtained as an extension to the state-of-the-art answer set solver CLASP [22], provides a unique combination of traditional unfounded set [29] checking and acyclicity propagation [19]. This novel combination is experimentally evaluated to assess the interplay of both reasoning mechanisms.

The rest of this paper is structured as follows. The basic notions of answer set programming are recalled in Section 2. The extension by explicit acyclicity constraints is worked out in Section 3, where we characterize relationships between ASP modulo acyclicity and standard ASP that pave the way for solving approaches implemented in the state-of-the-art ASP solver CLASP. The experimental evaluation of the new extension follows in Section 4. Finally, the results of the paper are discussed in Section 5.

2 Background

We consider logic programs built from rules and optimization statements of the forms:

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

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

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

$$\#\texttt{minimize}[b_1 = w_1, \ldots, b_n = w_n, \texttt{not}\ c_1 = w_{n+1}, \ldots, \texttt{not}\ c_m = w_{n+m}]. \quad (4)$$

Symbols $a, b_1, \ldots, b_n, c_1, \ldots, c_m$ stand for (propositional) *atoms*, $k, w_1, \ldots, w_{n+m}$ 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 or an *optimization statement* of the form (4), the respective sequence of *weighted literals* is denoted by $\mathrm{WL}(r)$, and its restrictions to positive or negative literals are denoted 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*, 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. A program P may be accompanied by an optimization statement declaring weighted literals to be minimized.

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 or optimization statement r evaluate to $\mathrm{v}_I(\mathrm{WL}(r)) = \sum_{1 \leq i \leq n, b_i \in I} w_i + \sum_{1 \leq i \leq m, c_i \notin I} w_{n+i}$; when r is a weight rule, 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 $\mathrm{head}(r) \in I$. 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 $\mathrm{head}(\mathrm{SR}_P(I)) = I$. Note that any positive program P possesses a unique *least model*, denoted by $\mathrm{LM}(P)$.

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 $\mathrm{B}(r)^I = (\max\{0, k - \mathrm{v}_I(\mathrm{WL}(r)^-)\} \leq \mathrm{WL}(r)^+)$ is the reduct of $\mathrm{B}(r)$ for a weight rule 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, I is a *stable model* of P iff $I \models P$ and $\mathrm{LM}(P^I) = I$. When P is accompanied by an optimization statement r, a stable (or supported) model I of P is *optimal* iff there is no stable (or supported) model I' of P such that $\mathrm{v}_{I'}(\mathrm{WL}(r)) < \mathrm{v}_I(\mathrm{WL}(r))$. 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 1. *A model I 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 $r_1, \ldots, r_n$ of R such that, for each $1 \leq i \leq n$, $\mathrm{head}(\{r_1, \ldots, r_{i-1}\}) \models \mathrm{B}(r_i)^I$.*

In fact, a (supported) model I of a program P is stable iff I is well-supported by some subset of $\mathrm{SR}_P(I)$, and several such subsets may 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 $\mathrm{SCC}_P(a)$.

3 Acyclicity Constraints

In [19], 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 2. *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 likely to contain cycles. If not, then 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 $E \subseteq V^2$ 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. *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 1. Consider a directed graph $\langle V, E\rangle$ and the task to find a Hamiltonian cycle through the graph, i.e., a cycle that visits each node of the graph exactly once. Let us encode the graph 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 similar to those in [20] express (ii):

$$1\{\mathtt{hc}(v, u) : \mathtt{edge}(v, u)\}1 \leftarrow \mathtt{node}(v). \tag{5}$$
$$1\{\mathtt{hc}(v, u) : \mathtt{edge}(v, u)\}1 \leftarrow \mathtt{node}(u). \tag{6}$$

To enforce (iii), we introduce an acyclicity extension $\langle V, e\rangle$, where e maps an atom $\mathtt{hc}(v, u)$ to an edge $\langle v, u\rangle$ whenever v and u are different from v_0. The mechanisms to implement acyclicity extensions in practice are described in Section 4. ■

Our next objective is to relate acyclicity programs to ordinary logic programs in terms of translations. It is well-known that logic programs subject to stable model semantics can express reachability in graphs, which implies that also acyclicity is expressible. To this end, we present a translation based on *elimination orderings* [18].

Definition 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:*

$$\mathtt{el}(v, u) \leftarrow \mathtt{not}\ a. \tag{7}$$
$$\mathtt{el}(v, u) \leftarrow \mathtt{el}(u). \tag{8}$$

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

$$\mathtt{el}(v) \leftarrow \mathtt{el}(v, u_1), \ldots, \mathtt{el}(v, u_k). \tag{9}$$
$$\leftarrow \mathtt{not}\ \mathtt{el}(v). \tag{10}$$

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 (7), 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 (8) is applicable once the end node u is eliminated. Then, the node v gets eliminated by the rule (9) if all edges starting from it are eliminated. Finally, the constraint (10) 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 $v \in V$.

Theorem 1. *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 $M' = M \cup \{\texttt{el}(v,u) \mid \langle v,u\rangle \in e(\mathrm{At}(P))\} \cup \{\texttt{el}(v) \mid v \in V\}$ is a stable model of $\mathrm{Tr}_{\mathrm{EL}}(P,V,e)$.*
2. *If M' is a stable model of $\mathrm{Tr}_{\mathrm{EL}}(P,V,e)$, then $M = M' \cap \mathrm{At}(P)$ is a stable model of P subject to $\langle V,e\rangle$.*

Example 2. Consider the following acyclicity program P:

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

$$
\begin{array}{llll}
\{x\}. & \{y\}. & \texttt{ws}(r_{11}). \quad \texttt{ws}(r_{12}). & \leftarrow x,\ \texttt{not ws}(r_{11}). \quad \leftarrow y,\ \texttt{not ws}(r_{12}).
\end{array}
$$

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

Let the acyclicity extension $\langle V,e\rangle$ of P consist of the nodes $V = \{p,q,r,s,x,y\}$, and atoms of the form $\texttt{dep}(v,u)$ indicate the mapping $e(\texttt{dep}(v,u)) = \langle v,u\rangle$ to edges. Observe that the nodes in V correspond to atoms in P and that $\texttt{dep}(v,u)$ atoms in the bodies of rules with $\texttt{ws}(r_i)$ in the head match (positive) dependencies in a corresponding rule r_i. In fact, P matches the outcome of applying the translation introduced in Definition 5 below to the subprogram P' consisting of the rules with elements of $V = \{p,q,r,s,x,y\}$ in their heads, and later on we also specify additional constraints pruning redundant stable models. Here we first use P with the acyclicity extension $\langle V,e\rangle$ to illustrate how it can be mapped to an ordinary logic program via $\mathrm{Tr}_{\mathrm{EL}}(P,V,e)$.

When interpreted in the standard way, P admits 620 stable models, out of which 68 remain in view of the acyclicity condition on the graph induced by true $\texttt{dep}(v,u)$

atoms. For instance, eight stable models M are such that $M \cap V = \{p, q, r, s, y\}$, and additional $\mathtt{dep}(v, u)$ atoms reflect well-supports, where $\mathtt{dep}(q, y)$ as well as $\mathtt{ws}(r_6)$ are true and express that the rule $q \leftarrow y$ must be applied in order to establish q. Then, the choice rules for r and s can be triggered, indicated by the atoms $\mathtt{dep}(v, q)$ and $\mathtt{dep}(v, y)$ for $v \in \{r, s\}$; finally, p is derived when at least two of the atoms $\mathtt{dep}(p, q)$, $\mathtt{dep}(p, r)$, and $\mathtt{dep}(p, s)$ hold, which yields four stable models containing $\mathtt{ws}(r_8)$ and $\mathtt{ws}(r_{10})$ along with $\mathtt{ws}(r_2)$, $\mathtt{ws}(r_3)$, and/or $\mathtt{ws}(r_4)$. The other four stable models include $\mathtt{dep}(v, q)$ and $\mathtt{dep}(v, y)$ as well as $\mathtt{ws}(r_8)$ or $\mathtt{ws}(r_{10})$, for either $v = r$ or $v = s$, along with $\mathtt{dep}(p, q)$, $\mathtt{dep}(p, v)$, and $\mathtt{ws}(r_2)$ or $\mathtt{ws}(r_3)$; the derivation of $u = s$, when $v = r$, or $u = r$, when $v = s$, is then based on $\mathtt{dep}(u, p)$, $\mathtt{dep}(u, q)$, and $\mathtt{ws}(r_7)$ or $\mathtt{ws}(r_9)$, where $\mathtt{dep}(u, y)$ and thus $\mathtt{ws}(r_8)$ or $\mathtt{ws}(r_{10})$ may hold in addition. In terms of P, these options to pick $\mathtt{dep}(v, u)$ atoms vary in whether the rule $p \leftarrow r,\ s$ or one among $r \leftarrow p,\ q$ and $s \leftarrow p,\ q$ is well-supporting, while the acyclicity condition on $\mathtt{dep}(p, r)$, $\mathtt{dep}(p, s)$, $\mathtt{dep}(r, p)$, and $\mathtt{dep}(s, p)$ prohibits using rules of both kinds simultaneously.

In order to capture the acyclicity condition by an ordinary program, P can be augmented with the following rules of the forms (7)–(10):

$$
\begin{array}{lll}
\mathtt{el}(p,q) \leftarrow \mathtt{not}\ \mathtt{dep}(p,q). & \mathtt{el}(p,q) \leftarrow \mathtt{el}(q). & \mathtt{el}(p) \leftarrow \mathtt{el}(p,q), \\
\mathtt{el}(p,r) \leftarrow \mathtt{not}\ \mathtt{dep}(p,r). & \mathtt{el}(p,r) \leftarrow \mathtt{el}(r). & \quad\ \mathtt{el}(p,r), \\
\mathtt{el}(p,s) \leftarrow \mathtt{not}\ \mathtt{dep}(p,s). & \mathtt{el}(p,s) \leftarrow \mathtt{el}(s). & \quad\ \mathtt{el}(p,s), \\
\mathtt{el}(p,x) \leftarrow \mathtt{not}\ \mathtt{dep}(p,x). & \mathtt{el}(p,x) \leftarrow \mathtt{el}(x). & \quad\ \mathtt{el}(p,x). \\
\mathtt{el}(q,r) \leftarrow \mathtt{not}\ \mathtt{dep}(q,r). & \mathtt{el}(q,r) \leftarrow \mathtt{el}(r). & \mathtt{el}(q) \leftarrow \mathtt{el}(q,r), \\
\mathtt{el}(q,s) \leftarrow \mathtt{not}\ \mathtt{dep}(q,s). & \mathtt{el}(q,s) \leftarrow \mathtt{el}(s). & \quad\ \mathtt{el}(q,s), \\
\mathtt{el}(q,y) \leftarrow \mathtt{not}\ \mathtt{dep}(q,y). & \mathtt{el}(q,y) \leftarrow \mathtt{el}(y). & \quad\ \mathtt{el}(q,y). \\
\mathtt{el}(r,p) \leftarrow \mathtt{not}\ \mathtt{dep}(r,p). & \mathtt{el}(r,p) \leftarrow \mathtt{el}(p). & \mathtt{el}(r) \leftarrow \mathtt{el}(r,p), \\
\mathtt{el}(r,q) \leftarrow \mathtt{not}\ \mathtt{dep}(r,q). & \mathtt{el}(r,q) \leftarrow \mathtt{el}(q). & \quad\ \mathtt{el}(r,q), \\
\mathtt{el}(r,y) \leftarrow \mathtt{not}\ \mathtt{dep}(r,y). & \mathtt{el}(r,y) \leftarrow \mathtt{el}(y). & \quad\ \mathtt{el}(r,y). \\
\mathtt{el}(s,p) \leftarrow \mathtt{not}\ \mathtt{dep}(s,p). & \mathtt{el}(s,p) \leftarrow \mathtt{el}(p). & \mathtt{el}(s) \leftarrow \mathtt{el}(s,p), \\
\mathtt{el}(s,q) \leftarrow \mathtt{not}\ \mathtt{dep}(s,q). & \mathtt{el}(s,q) \leftarrow \mathtt{el}(q). & \quad\ \mathtt{el}(s,q), \\
\mathtt{el}(s,y) \leftarrow \mathtt{not}\ \mathtt{dep}(s,y). & \mathtt{el}(s,y) \leftarrow \mathtt{el}(y). & \quad\ \mathtt{el}(s,y).
\end{array}
$$

$$
\begin{array}{llll}
\leftarrow \mathtt{not}\ \mathtt{el}(p). & \leftarrow \mathtt{not}\ \mathtt{el}(q). & \leftarrow \mathtt{not}\ \mathtt{el}(x). & \mathtt{el}(x). \\
\leftarrow \mathtt{not}\ \mathtt{el}(r). & \leftarrow \mathtt{not}\ \mathtt{el}(s). & \leftarrow \mathtt{not}\ \mathtt{el}(y). & \mathtt{el}(y).
\end{array}
$$

As stated in Theorem 1, the resulting ordinary logic program $\mathrm{Tr}_{\mathrm{EL}}(P, V, e)$ captures the 68 stable models of P subject to $\langle V, e\rangle$. Note that each stable model of $\mathrm{Tr}_{\mathrm{EL}}(P, V, e)$ includes all $\mathtt{el}(v, u)$ and $\mathtt{el}(v)$ atoms introduced in the above program part. ■

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 [17] provides a starting point for such a transformation, the target syntax is given by rules, including weight rules of the form (3), rather than clauses.

Definition 5. *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 $e(\mathtt{dep}(a, b)) = \langle a, b\rangle$ 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:*

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

2. *For each defining rule (1) or (2) of* a*, the rule:*

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

3. *For each defining rule (3) of* a*, the rule:*

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

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

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

The rules (12) and (13) specify when r provides well-support for a, i.e., the head atom a non-circularly depends on $\mathrm{B}(r)^+ = \{b_1, \ldots, b_n\}$. 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 the choice rules (11).

Theorem 2. *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*, then there is an ordering* $r_1, \ldots, r_n$ *of some* $R \subseteq \mathrm{SR}_P(M)$ *such that* $M' = M \cup \{\mathtt{ws}(r) \mid r \in R\} \cup \{\mathtt{dep}(\mathrm{head}(r_i), b) \mid 1 \leq i \leq n, b \in B_i\}$*, where* $B_i \subseteq \mathrm{B}(r_i)^+ \cap \mathrm{head}(\{r_1, \ldots, r_{i-1}\})$ *for each* $1 \leq i \leq n$*, is a supported model of* $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ *subject to* $\langle \mathrm{At}(P), e\rangle$*.*
2. *If* M' *is a supported model of* $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ *subject to* $\langle \mathrm{At}(P), e\rangle$*, then* $M = M' \cap \mathrm{At}(P)$ *is a stable model of* P *and* M *is well-supported by* $R = \{r \mid \mathtt{ws}(r) \in M'\}$*.*

Note that the first item would also hold when $B_i = \mathrm{B}(r_i)^+ \cap \mathrm{head}(\{r_1, \ldots, r_{i-1}\})$ enforces edges to all established atoms in a body. However, such strictness is inappropriate in the presence of additional constraints for weight rules, given in Definition 7 below.

It is well-known that supported and stable models coincide for *tight* logic programs [15, 12]. 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—either using unfounded set checking or the acyclicity constraint, or both.

Theorem 3. *Let* P *be a weight constraint program and* $\mathrm{Tr}_{\mathrm{ACYC}}(P)$ *its translation into an acyclicity program with an acyclicity extension* $\langle \mathrm{At}(P), e\rangle$*. 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* $\langle \mathrm{At}(P), e\rangle$*.*

Example 3. The acyclicity program P subject to $\langle V, e\rangle$ in Example 2 is the acyclicity translation $\mathrm{Tr}_{\mathrm{ACYC}}(P')$ of the subprogram P' given by the rules that do not include atoms of the form $\mathtt{dep}(v, u)$ or $\mathtt{ws}(r_i)$. The subprogram P' has five stable models: $\emptyset$, $\{x\}$, $\{q, y\}$, $\{p, q, r, s, y\}$, and $\{p, q, r, s, x, y\}$. They map to 68 supported models of $\mathrm{Tr}_{\mathrm{ACYC}}(P')$, all of which are stable. ■

As witnessed by Theorems 2 and 3, 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 $\mathtt{dep}(v, u)$ 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 [17] but reformulates them in terms of rules rather than clauses.

Definition 6. *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:*

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

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:*

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

Intuitively, dependencies from a are not needed if a is false (15). Quite similarly, a particular dependency may be safely prevented (16) if the well-support for a is provided by a rule r not involving this dependency.

Example 4. The strong acyclicity translation $\mathrm{Tr}_{\mathrm{ACYC+}}(P')$ of P' as in Example 3 augments $\mathrm{Tr}_{\mathrm{ACYC}}(P')$, given by P subject to $\langle V, e\rangle$ in Example 2, with the constraints:

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

The addition of these constraints to $\mathrm{Tr}_{\mathrm{ACYC}}(P')$ reduces the number of supported (and stable) models from 68 to 17. For instance, three of the eight supported models M extending $M \cap V = \{p, q, r, s, y\}$ are eliminated, namely, the ones containing $S_1 = \{\mathtt{dep}(p,q), \mathtt{dep}(p,r), \mathtt{dep}(p,s), \mathtt{ws}(r_2), \mathtt{ws}(r_3), \mathtt{ws}(r_4)\}$, $S_2 = \{\mathtt{dep}(r,p), \mathtt{dep}(r,q), \mathtt{dep}(r,y), \mathtt{ws}(r_7), \mathtt{ws}(r_8)\}$, or $S_3 = \{\mathtt{dep}(s,p), \mathtt{dep}(s,q), \mathtt{dep}(s,y), \mathtt{ws}(r_9), \mathtt{ws}(r_{10})\}$.

In fact, atoms of the form $\mathtt{dep}(v, u)$ in S_1, S_2, or S_3 lead to redundant well-supports, e.g., $\mathtt{ws}(r_4) \in S_1$ yields that $\mathtt{dep}(p, r)$ and $\mathtt{dep}(p, s)$ suffice to support p, so that $\mathtt{dep}(p, q)$ is unnecessary. However, five non-redundant supported models M such that $M \cap V = \{p, q, r, s, y\}$ still reflect different options to pick well-supporting rules. ■

The strong acyclicity translation for weight rules includes additional subprograms.

Definition 7. *Let P be a weight constraint program and $r \in P$ a weight rule of the form (3), where $\mathrm{head}(r) = a$, $|\{b_1, \ldots, b_n\}| = n$, and $w_1, \ldots, w_n$ are ordered such that $w_{i-1} \leq w_i$ 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:*

$$\mathtt{nxt}(r, i) \leftarrow \mathtt{dep}(a, b_{i-1}). \tag{17}$$
$$\mathtt{nxt}(r, i) \leftarrow \mathtt{nxt}(r, i-1). \tag{18}$$
$$\mathtt{chk}(r, i) \leftarrow \mathtt{nxt}(r, i), \mathtt{dep}(a, b_i). \tag{19}$$

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 $\langle a, b\rangle \in \mathrm{DG}^+(P)$ such that $b \in \mathrm{B}(r)^+$, 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 though r can be deemed redundant by the rule (20). To this end, the rules of the forms (17) and (18) identify an atom among $b_1, \ldots, b_n$ of smallest weight having an active dependency from a, i.e., $\mathtt{dep}(a, b_i)$ is true, provided such an i exists. By the rules of the form (19), any further dependencies are extracted, and (20) checks whether the remaining literals are sufficient to reach the bound k. If so, all dependencies from a are viewed as redundant. This check covers also cases where, e.g., negative literals suffice to satisfy the body and positive dependencies play no role.

Example 5. The subprogram P' and $P = \mathrm{Tr}_{\mathrm{ACYC}}(P')$ as in Example 3 can be modified to yield equivalent weight constraint programs. To this end, reconsider P and $\langle V, e\rangle$, given in Example 2, and assume that the rules with head p or $\mathtt{ws}(r_1), \ldots, \mathtt{ws}(r_4)$, respectively, as well as the integrity constraint including p, are replaced by:

$$p \leftarrow q,\ x. \qquad \mathtt{ws}(r_1) \leftarrow \mathtt{dep}(p, q), \mathtt{dep}(p, x).$$
$$p \leftarrow 3 \leq [q = 1,\ r = 2,\ s = 2]. \qquad \mathtt{ws}(r_2) \leftarrow 3 \leq [\mathtt{dep}(p, q) = 1,$$
$$\leftarrow p, \mathtt{not}\ \mathtt{ws}(r_1), \mathtt{not}\ \mathtt{ws}(r_2). \qquad \mathtt{dep}(p, r) = 2, \mathtt{dep}(p, s) = 2].$$

This translation $\mathrm{Tr}_{\mathrm{ACYC}}(P')$ still yields 68 supported models subject to $\langle V, e\rangle$. Its strong version $\mathrm{Tr}_{\mathrm{ACYC}+}(P')$ is obtained by replacing integrity constraints, given in Example 4, that contain atoms of the form $\mathtt{dep}(p, u)$ for $u \in \{q, r, s, x\}$ as follows:

$\leftarrow \mathtt{dep}(p,q),\ \mathtt{not}\ p.$

$\leftarrow \mathtt{dep}(p,r),\ \mathtt{not}\ p.$ $\quad \leftarrow \mathtt{dep}(p,s),\ \mathtt{not}\ p.$ $\quad \leftarrow \mathtt{dep}(p,x),\ \mathtt{not}\ p.$

$\leftarrow \mathtt{dep}(p,r),\ \mathtt{ws}(r_1).$ $\quad \leftarrow \mathtt{dep}(p,s),\ \mathtt{ws}(r_1).$ $\quad \leftarrow \mathtt{dep}(p,x),\ \mathtt{ws}(r_2).$

$\mathtt{nxt}(r_2,2) \leftarrow \mathtt{dep}(p,q).$ $\quad \mathtt{nxt}(r_2,3) \leftarrow \mathtt{dep}(p,r).$

$\mathtt{nxt}(r_2,2) \leftarrow \mathtt{nxt}(r_2,1).$ $\quad \mathtt{nxt}(r_2,3) \leftarrow \mathtt{nxt}(r_2,2).$

$\mathtt{chk}(r_2,2) \leftarrow \mathtt{nxt}(r_2,2),\ \mathtt{dep}(p,r).$ $\quad \mathtt{chk}(r_2,3) \leftarrow \mathtt{nxt}(r_2,3),\ \mathtt{dep}(p,s).$

$\mathtt{red}(r_2) \leftarrow 3 \leq [\mathtt{chk}(r_2,2) = 2,\ \mathtt{chk}(r_2,3) = 2].$

$\leftarrow \mathtt{dep}(p,q),\ \mathtt{red}(r_2).$ $\quad \leftarrow \mathtt{dep}(p,r),\ \mathtt{red}(r_2).$ $\quad \leftarrow \mathtt{dep}(p,s),\ \mathtt{red}(r_2).$

As in Example 4, the addition of these constraints to $\mathrm{Tr}_{\mathrm{ACYC}}(P')$ reduces the number of supported models to 17. In particular, the redundant model M that extends $M \cap V = \{p,q,r,s,y\}$ and contains $\{\mathtt{dep}(p,q), \mathtt{dep}(p,r), \mathtt{dep}(p,s), \mathtt{ws}(r_2)\}$ is denied since $\mathtt{nxt}(r_2,2)$, $\mathtt{nxt}(r_2,3)$, $\mathtt{chk}(r_2,2)$, $\mathtt{chk}(r_2,3)$, and $\mathtt{red}(r_2)$ follow from $\mathtt{dep}(v,u)$ atoms. That is, $\mathtt{dep}(p,r)$ and $\mathtt{dep}(p,s)$ are sufficient for well-support through r_2, so that $\mathtt{dep}(p,q)$ is unnecessary. In turn, three non-redundant supported models can be obtained from M by dropping either $\mathtt{dep}(p,q)$, $\mathtt{dep}(p,r)$, or $\mathtt{dep}(p,s)$. ■

The transformations $\mathrm{Tr}_{\mathrm{ACYC}}$ and $\mathrm{Tr}_{\mathrm{ACYC+}}$ can be adjusted to take SCCs into account and, in practice, only their component-aware versions have been implemented. Definitions 5 and 6 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 $b_i \notin \mathrm{SCC}_P(a)$. Rules of the forms (11), (15), (16), and (21) are only needed if $b \in \mathrm{SCC}_P(a)$. As regards Definition 7, the condition for ordering the weight rule is refined so that, for some $1 \leq k \leq n$, $\{b_1,\ldots,b_k\} \subseteq \mathrm{SCC}_P(a)$, $\{b_{k+1},\ldots,b_n\} \cap \mathrm{SCC}_P(a) = \emptyset$, and $w_{i-1} \leq w_i$ for each $1 < i \leq k$. Then, the rules (17)–(19) are restricted to $1 < i \leq k$. Finally, the atoms $\mathtt{chk}(r,i)$ in the rule (20) are replaced by b_i for $k < i \leq n$. Note that the relationships among models established in Theorems 2 and 3 remain valid for the strong translation $\mathrm{Tr}_{\mathrm{ACYC+}}$ as well as component-aware versions of $\mathrm{Tr}_{\mathrm{ACYC}}$ and $\mathrm{Tr}_{\mathrm{ACYC+}}$. As a consequence, the correspondence between stable models and supported models subject to acyclicity carries forward to optimal models in the presence of optimization statements.

Example 6. For P' as in Example 3 and $a \in \{p,q,r,s\}$, we have that $\mathrm{SCC}_{P'}(a) = \{p,q,r,s\}$. Hence, the component-aware version of $\mathrm{Tr}_{\mathrm{ACYC+}}(P')$ replaces the atoms $\mathtt{dep}(p,x)$, $\mathtt{dep}(q,y)$, $\mathtt{dep}(r,y)$, and $\mathtt{dep}(s,y)$ in P, given in Example 2, by x or y, respectively, so that the corresponding edges no longer contribute to the acyclicity extension induced by $e(\mathtt{dep}(v,u)) = \langle v,u\rangle$. Moreover, among the constraints shown in Example 4, those mentioning some of the four obsolete $\mathtt{dep}(v,u)$ atoms are simply dropped. Note that all supported models extending $M \cap V = \{p,q,r,s,y\}$ include $\mathtt{dep}(r,q)$ and $\mathtt{dep}(s,q)$, leading to $\mathtt{ws}(r_8)$ and $\mathtt{ws}(r_{10})$ because $\mathtt{dep}(r,y)$ and $\mathtt{dep}(s,y)$ are not needed as prerequisites anymore. Hence, the constraints $\leftarrow \mathtt{dep}(r,p),\ \mathtt{ws}(r_8)$ and $\leftarrow \mathtt{dep}(s,p),\ \mathtt{ws}(r_{10})$ suppress edges associated with $\mathtt{dep}(r,p)$ and $\mathtt{dep}(s,p)$. Since the latter were still admissible in Example 4, the component-aware translation further reduces the number of non-redundant supported models such that $M \cap V = \{p,q,r,s,y\}$ from five to three. In total, we obtain eight non-redundant supported models capturing the five stable models given in Example 3. ■

4 Experiments

Acyclicity programs are implemented in the development version 3.2.0-R45720 of CLASP[4] [22], using a propagator for acyclicity reasoning similar to the one introduced in [19], and will be supported from the forthcoming release 3.2.0 on. On the one hand, a program may define the dedicated predicate $\texttt{\_edge}(v,u)$ to declare $e(\texttt{\_edge}(v,u)) = \langle v,u\rangle$. For instance, the rule $\texttt{\_edge}(v,u) \leftarrow \texttt{hc}(v,u),\ v \neq n_0,\ u \neq n_0$ specifies the acyclicity extension for the Hamiltonian cycle encoding consisting of (5) and (6) in Example 1. On the other hand, the tool LP2ACYC[5] [17], version 1.29, 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. In fact, we used ordinary ASP encodings and LP2ACYC to compare the performance of CLASP performing usual *unfounded set* or *acyclicity checking*, below abbreviated by UFS or ACYC, respectively. The ACYC mode allows for "backward" inference of forbidden edges, i.e., (unassigned) atoms standing for edges that would close a cycle are falsified, and we abbreviate the use of such propagation by BCYC. Moreover, we denote the application of ACYC or BCYC relative to $\mathrm{Tr}_{\mathrm{ACYC+}}$ by ACYC+ and BCYC+. Given that stable and supported models coincide for acyclicity programs obtained by translation, each of the acyclicity checking variants can be combined with UFS, and we refer to such a combination by appending /UFS to the respective mode, e.g., ACYC+/UFS denotes acyclicity along with unfounded set checking relative to $\mathrm{Tr}_{\mathrm{ACYC+}}$. Note that all modes but UFS, which matches standard ASP solving, are based on translation into acyclicity programs.

Our benchmark set[6] consists of four classes of *decision* problems, Hamiltonian Cycle (*Cycle*) [28], Labyrinth (*Laby*) [1], Sokoban (*Soko*) [24], and Wire Routing (*Route*) [10], as well as three *optimization* problems, Bayesian Network Structure Learning (*Bayes*) [9], Chordal Markov Network Learning (*Markov*) [8], and Incremental Scheduling (*Sched*) [7]. The instances of each problem are *non-tight* [15, 12], so that unfounded set and/or acyclicity checking is required for solving them. All experiments were run single-threaded with CLASP's *trendy* configuration, which turned out to boost the search performance (8 timeouts less than the default configuration in baseline UFS mode), on a Linux machine equipped with Intel Quad-Core Xeon E5520 2.27GHz processors, imposing a limit of 600 seconds wall-clock time and no (effective) memory limit per run, where a timeout is counted as 600 seconds within average runtimes.

Table 1 provides average runtimes in seconds and numbers of timeouts for each CLASP mode, highlighting the minimum values per benchmark class in boldface, where the number of instances per class is given at the top. In terms of timeouts, we observe that regular unfounded set checking, i.e., UFS, is ahead or equal to acyclicity checking modes. However, recall that acyclicity checking is applied to $\mathrm{Tr}_{\mathrm{ACYC}}$ or $\mathrm{Tr}_{\mathrm{ACYC+}}$ translations, thus not using native encodings in terms of acyclicity programs, which may still boost the performance on a problem-specific level. Nevertheless, for some classes, different acyclicity checking modes yield the same number of timeouts and smaller average runtimes than UFS. This applies to ACYC, using plain acyclicity checking, on

[4] http://sourceforge.net/projects/potassco/

[5] http://research.ics.aalto.fi/software/asp/lp2acyc/

[6] http://www.cs.uni-potsdam.de/clasp/?page=experiments

Mode	*Cycle* #60		*Laby* #20		*Soko* #30		*Route* #23		*Bayes* #30		*Markov* #21		*Sched* #18	
UFS	**36.0**	**0**	255.3	**4**	182.6	**2**	**5.8**	**0**	116.8	**0**	100.7	**0**	**281.2**	**7**
ACYC	373.6	37	261.0	6	350.7	10	134.5	4	**66.3**	**0**	120.3	1	320.9	8
BCYC	266.3	26	286.7	7	256.2	7	111.5	2	84.6	**0**	54.1	**0**	324.2	**7**
ACYC/UFS	209.4	18	279.2	**4**	174.6	3	11.4	**0**	103.1	1	170.2	3	348.2	9
BCYC/UFS	209.2	19	314.3	6	179.7	4	10.0	**0**	104.3	1	72.5	**0**	340.3	9
ACYC+	118.0	7	366.7	7	336.7	10	137.2	4	106.2	1	61.5	**0**	340.9	9
BCYC+	85.3	5	279.6	5	230.4	5	138.6	4	102.2	2	**39.9**	**0**	341.1	9
ACYC+/UFS	115.9	8	311.8	5	176.6	4	15.4	**0**	110.3	1	171.4	3	367.5	9
BCYC+/UFS	91.9	6	**212.7**	**4**	**170.2**	3	12.3	**0**	122.5	2	111.5	1	360.6	9

Table 1. Benchmark results comparing unfounded set and acyclicity checking

the optimization problem *Bayes*. Its strengthening by additional constraints in $\mathrm{Tr}_{\mathrm{ACYC+}}$ along with backward propagation, BCYC+, yields the best performance for the *Markov* class, and the respective combination with unfounded set checking, BCYC+/UFS, leads to the shortest average runtime (and fewest timeouts) on the decision problem *Laby*, where BCYC+/UFS and UFS each solve an instance on which the other mode times out. In contrast, *Cycle* constitutes a negative example in which translation and acyclicity checking deteriorate performance, even if combined with UFS.

Comparing the different acyclicity checking modes to each other, using backward propagation may significantly improve performance, as observed on the gaps between ACYC and BCYC as well as ACYC+ and BCYC+ for the *Cycle* and *Soko* classes. However, these are also the problems where UFS has advantages (in terms of timeouts), and combining acyclicity with unfounded set checking largely outweighs backward propagation here. On the other hand, the *Markov* class, where combinations with UFS deteriorate performance, shows that backward propagation does not become obsolete when unfounded set checking is used in addition. The latter turns out to be particularly helpful for the *Route* class, although no combination catches up to pure UFS. Regarding the effect of $\mathrm{Tr}_{\mathrm{ACYC+}}$, the *Cycle* class yields significant advantages of the ACYC+ and BCYC+ modes compared to their ACYC and BCYC counterparts. In turn, on the optimization problem *Bayes*, omitting the additional constraints of $\mathrm{Tr}_{\mathrm{ACYC+}}$ saves some overhead.

In summary, the benchmark results in Table 1 show that the use of $\mathrm{Tr}_{\mathrm{ACYC}}$ or $\mathrm{Tr}_{\mathrm{ACYC+}}$ and likewise of backward propagation on the solver side matter, although there is no winning strategy that would dominate for all classes of problems. Similarly, combining acyclicity with unfounded set checking can be advantageous, especially in cases where acyclicity checking has difficulties on its own, but it may also lead to overhead instead of gains. In fact, when using $\mathrm{Tr}_{\mathrm{ACYC}}$ or $\mathrm{Tr}_{\mathrm{ACYC+}}$, acyclicity and unfounded set checking both deal with the same concern of guaranteeing that a supported model is also well-supported. Hence, applying different mechanisms serving the same purpose is partially redundant. In direct encodings, which are beyond the scope of this paper, a user has the option to customize whether to apply acyclicity or unfounded set checking for particular purposes. Thus, acyclicity programs enrich the available modeling constructs.

5 Discussion

In this paper, we propose a novel SMT-style extension of ASP by explicit acyclicity constraints in analogy to [19]. These kinds of constraints have not been directly addressed in previous SMT-style extensions of ASP [23, 26, 25]. The new extension, herein coined ASP modulo acyclicity, offers a unique set of primitives for applications involving DAGs or tree structures. One interesting application is the embedding of ASP itself, given that unfounded set checking can be captured (Theorem 2). The utilized notion of well-supporting rules resembles *source pointers* [27], used in native answer set solvers to record rules justifying true atoms. In fact, a major contribution of this work is the implementation of new translations and principles in tools. Firstly, version 1.29 of LP2ACYC implements $\mathrm{Tr}_{\mathrm{ACYC}}$ and $\mathrm{Tr}_{\mathrm{ACYC+}}$ relative to SCCs of an input program. Given that the implementation covers extended rule types, weight constraint programs output by the grounder GRINGO [20] can be processed. Secondly, from release 3.2.0 onward, CLASP [22] will have the acyclicity propagator built in. 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 [19], which turned out to be competitive on translations of logic programs (without extended rules) into SAT modulo acyclicity [17]. Upon enumeration, a replication of supported (and stable) models under translations can be avoided by using the projection capabilities of CLASP [21]. Last but not least, acyclicity programs enrich the variety of modeling primitives available to users.

Acknowledgments. This work was funded by AoF (251170), DFG (SCHA 550/8 and 550/9), as well as DAAD and AoF (57071677/279121).

References

1. M. Alviano, F. Calimeri, G. Charwat, M. Dao-Tran, C. Dodaro, G. Ianni, T. Krennwallner, M. Kronegger, J. Oetsch, A. Pfandler, J. Pührer, C. Redl, F. Ricca, P. Schneider, M. Schwengerer, L. Spendier, J. Wallner, and G. Xiao. The fourth answer set programming competition: Preliminary report. In P. Cabalar and T. Son, editors, *Proceedings of the Twelfth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'13)*, volume 8148 of *Lecture Notes in Artificial Intelligence*, pages 42–53. Springer-Verlag, 2013.
2. G. Audemard and L. Simon. Predicting learnt clauses quality in modern SAT solvers. In C. Boutilier, editor, *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 399–404. AAAI/MIT Press, 2009.
3. C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.
4. C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. Satisfiability modulo theories. In Biere et al. [6], pages 825–885.
5. S. Bayless, N. Bayless, H. Hoos, and A. Hu. SAT modulo monotonic theories. In B. Bonet and S. Koenig, editors, *Proceedings of the Twenty-ninth National Conference on Artificial Intelligence (AAAI'15)*, pages 3702–3709. AAAI Press, 2015.
6. A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

7. F. Calimeri, M. Gebser, M. Maratea, and F. Ricca. The design of the fifth answer set programming competition. In M. Leuschel and T. Schrijvers, editors, *Technical Communications of the Thirtieth International Conference on Logic Programming (ICLP'14). Theory and Practice of Logic Programming*, 14(4-5):Online Supplement, 2014.
8. J. Corander, T. Janhunen, J. Rintanen, H. Nyman, and J. Pensar. Learning chordal Markov networks by constraint satisfaction. In C. Burges, L. Bottou, Z. Ghahramani, and K. Weinberger, editors, *Proceedings of the Twenty-seventh Annual Conference on Neural Information Processing Systems (NIPS'13)*, pages 1349–1357. NIPS Foundation, 2013.
9. James Cussens. Bayesian network learning with cutting planes. In F. Cozman and A. Pfeffer, editors, *Proceedings of the Twenty-seventh International Conference on Uncertainty in Artificial Intelligence (UAI'11)*, pages 153–160. AUAI Press, 2011.
10. M. Denecker, J. Vennekens, S. Bond, M. Gebser, and M. Truszczyński. The second answer set programming competition. In Erdem et al. [14], pages 637–654.
11. N. Eén and N. Sörensson. An extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer-Verlag, 2004.
12. E. Erdem and V. Lifschitz. Tight logic programs. *Theory and Practice of Logic Programming*, 3(4-5):499–518, 2003.
13. E. Erdem, V. Lifschitz, and M. Wong. Wire routing and satisfiability planning. In J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K. Lau, C. Palamidessi, L. Pereira, Y. Sagiv, and P. Stuckey, editors, *Proceedings of the First International Conference on Computational Logic (CL'00)*, volume 1861 of *Lecture Notes in Computer Science*, pages 822–836. Springer-Verlag, 2000.
14. E. Erdem, F. Lin, and T. Schaub, editors. *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, volume 5753 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2009.
15. F. Fages. Consistency of Clark's completion and the existence of stable models. *Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.
16. E. Fermé and J. Leite, editors. *Proceedings of the Fourteenth European Conference on Logics in Artificial Intelligence (JELIA'14)*, volume 8761 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2014.
17. M. Gebser, T. Janhunen, and J. Rintanen. Answer set programming as SAT modulo acyclicity. In T. Schaub, G. Friedrich, and B. O'Sullivan, editors, *Proceedings of the Twenty-first European Conference on Artificial Intelligence (ECAI'14)*, pages 351–356. IOS Press, 2014.
18. M. Gebser, T. Janhunen, and J. Rintanen. ASP encodings of acyclicity properties. In C. Baral, G. De Giacomo, and T. Eiter, editors, *Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'14)*. AAAI Press, 2014.
19. M. Gebser, T. Janhunen, and J. Rintanen. SAT modulo graphs: Acyclicity. In Fermé and Leite [16], pages 137–151.
20. M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and M. Schneider. Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):107–124, 2011.
21. M. Gebser, B. Kaufmann, and T. Schaub. Solution enumeration for projected Boolean search problems. In W. van Hoeve and J. Hooker, editors, *Proceedings of the Sixth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'09)*, volume 5547 of *Lecture Notes in Computer Science*, pages 71–86. Springer-Verlag, 2009.
22. M. Gebser, B. Kaufmann, and T. Schaub. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 187-188:52–89, 2012.

23. M. Gebser, M. Ostrowski, and T. Schaub. Constraint answer set solving. In P. Hill and D. Warren, editors, *Proceedings of the Twenty-fifth International Conference on Logic Programming (ICLP'09)*, volume 5649 of *Lecture Notes in Computer Science*, pages 235–249. Springer-Verlag, 2009.
24. T. Janhunen, I. Niemelä, and M. Sevalnev. Computing stable models via reductions to difference logic. In Erdem et al. [14], pages 142–154.
25. J. Lee and Y. Meng. Answer set programming modulo theories and reasoning about continuous changes. In F. Rossi, editor, *Proceedings of the Twenty-third International Joint Conference on Artificial Intelligence (IJCAI'13)*, pages 990–996. IJCAI/AAAI Press, 2013.
26. G. Liu, T. Janhunen, and I. Niemelä. Answer set programming via mixed integer programming. In G. Brewka, T. Eiter, and S. McIlraith, editors, *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'12)*, pages 32–42. AAAI Press, 2012.
27. P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.
28. T. Soh, D. Le Berre, S. Roussel, M. Banbara, and N. Tamura. Incremental SAT-based method with native Boolean cardinality handling for the Hamiltonian cycle problem. In Fermé and Leite [16], pages 684–693.
29. A. Van Gelder, K. Ross, and J. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.