Declarative Encodings of Acyclicity Properties*

Martin Gebser*, Tomi Janhunen, and Jussi Rintanen**

Helsinki Institute for Information Technology HIIT
Department of Computer Science
Aalto University, FI-00076 AALTO, FINLAND

Abstract. Many knowledge representation tasks involve trees or similar structures as abstract datatypes. However, devising compact and efficient declarative representations of such structural properties is non-obvious and can be challenging indeed. In this paper, we take a number of acyclicity properties into consideration and investigate various logic-based approaches to encode them. We use answer set programming as the primary representation language but also consider mappings to related formalisms, such as propositional logic, difference logic, and linear programming. We study the compactness of encodings and the resulting computational performance on benchmarks involving acyclic or tree structures.

1 Introduction

Numerous hard search tasks involve the construction of acyclic or tree structures, and constraint satisfaction or related methods are an important approach for solving many of these problems. For instance, Bayesian network structure learning, where directed acyclic graphs provide solution candidates, can be reduced to constraint optimization [15, 28, 16]. Constraint-based methods can also be used to infer phylogenetic trees [7, 5], describing the evolution of living organisms, languages, and other evolving systems. Furthermore, chordal Markov network learning amounts to the task of optimizing maximum weight spanning trees induced by chordal graphs [14]. Since acyclicity and the property of being a tree are no primitives in common constraint-based representation formalisms, the challenge of formulating such conditions in terms of more basic constraint expressions arises. Hence, in this work, we systematically investigate logic-based approaches to encode respective properties.

The basic problem to be solved with constraint-based formalisms is constructing an acyclic or a tree structure subject to further conditions. Thereby, a part of the constraints must enforce that the choice among underlying edge candidates actually results in some graph structure with the desired properties. Straightforward formulations, however, can become impractically large and inefficient. While exponential encodings of acyclicity [28, 16] or chordality [14] may be acceptable for small graphs, they are prohibitive

* The support from the Finnish Centre of Excellence in Computational Inference Research (COIN) funded by the Academy of Finland (under grant #251170) is gratefully acknowledged.

* Also affiliated with the University of Potsdam, Germany. Corresponding author, email address: Martin.Gebser@aalto.fi

** Also affiliated with Griffith University, Brisbane, Australia.

when the problem size increases. Such limitations also apply to polynomial formulations that lack compactness, even when the given edge candidates yield sparse graphs. For instance, acyclicity may be expressed in terms of transitive closure [7, 10, 15, 11, 6] and formulated in first-order logic as follows:

$$\begin{aligned}&[\forall xy : \mathit{Edge}(x,y) \rightarrow \mathit{Reach}(x,y)]\\ \wedge\ &[\forall xyz : \mathit{Edge}(x,y) \wedge \mathit{Reach}(y,z) \rightarrow \mathit{Reach}(x,z)]\\ \wedge\ &[\forall x : \neg \mathit{Reach}(x,x)]\end{aligned}$$

The main issue here is that an instantiation must in general include $\mathit{Reach}(u, v)$ for any pair u, v of vertices, thus requiring quadratic space. Unlike this, we develop encodings based on least fixpoints or well-foundedness, respectively, such that conditions for "deriving" a vertex remain local to edge candidates including the vertex. We formulate respective conditions characterizing acyclicity in the directed and undirected case as well as chordality. Further properties can be enforced by side constraints, e.g., in order to assert connectedness and distinguish trees. In fact, we provide compact encodings of the properties detailed in Section 2 that are proportional to graph size and remain linear in the number of vertices when the edge candidates per vertex are bound by a constant.

Although we rely on answer set programming (ASP) [6] as the primary representation language for encodings, respective formulations in related formalisms, such as propositional satisfiability (SAT) [2], difference logic (DL) [37], linear programming (LP) [17], and the recent SAT modulo graphs framework [24] can be obtained through automatic polynomial translations from ASP. Linear translations to SAT exist whenever ASP rules are tight [19], i.e., if there are no circular positive dependencies through rules' prerequisites on the ground level. In the non-tight case, level mappings [29] can be used to bridge the semantic gap between ASP and SAT in subquadratic space. Compact linear representations of level mappings can be achieved by means of difference constraints available in DL [36], numerical variables in LP [33], or acyclicity checking on top of SAT [21]. Direct encodings in these formalisms could be based on similar principles as ASP formulations subject to translations, yet we aim at investigating the general impact of encoding approaches on the the resulting computational performance rather than fine-tuning to any particular application or solving method.

The rest of this paper is organized as follows. After providing definitions of acyclicity and tree properties, we present encoding approaches and corresponding first-order ASP formulations in Section 3. In Section 4, we evaluate these encodings on synthetic, reachability-oriented, and application benchmarks, followed by conclusions in Section 5. Correctness proofs for a selection of the encodings developed in Section 3 are given in Appendix A. This paper extends a preliminary workshop publication [23] with additional encodings and experiments as well as proofs. Parts of the material on directed acyclic graphs were also presented in a short paper [22].

2 Acyclicity Properties

This section introduces the basic concepts of graphs, forests, and trees. We consider directed as well as undirected notions and provide alternative characterizations when appropriate.

2.1 Directed Graphs

As usual, a *directed graph* G is a pair $\langle V, E\rangle$, where V is a finite set of *vertices* and $E \subseteq V \times V$ is a set of directed *edges*. For some $v \in V$, we denote the number of incoming or outgoing edges, respectively, by $\deg^-(v) = |\{u \mid \langle u, v\rangle \in E\}|$ and $\deg^+(v) = |\{u \mid \langle v, u\rangle \in E\}|$; v is a *root* (or *leaf*) of G if $\deg^-(v) = 0$ (or $\deg^+(v) = 0$). A *path* of length $k-1 \geq 0$ in G is a non-empty sequence $v_1, \ldots, v_k$ of vertices from V such that $\langle v_i, v_{i+1}\rangle \in E$ and $v_i \neq v_j$ for all $1 \leq i < j \leq k$. A sequence $v_0, v_1, \ldots, v_k$ is a *cycle* of length $k \geq 1$ in G if $\langle v_0, v_1\rangle \in E$, $v_0 = v_k$, and $v_1, \ldots, v_k$ is a path in G. Note that v, v is a cycle of length 1 whenever $\langle v, v\rangle \in E$.

A *directed acyclic graph* is a directed graph G such that there is no cycle in G. A directed acyclic graph $G = \langle V, E\rangle$ is a *directed forest* if, for every $v_k \in V$, there is exactly one path $v_1, \ldots, v_k$ in G from a root v_1 of G to v_k. Given that any incoming edge $\langle v_{k-1}, v_k\rangle$ can be extended to a path $v_1, \ldots, v_{k-1}, v_k$ from a root v_1, the former condition is equivalent to requiring $\deg^-(v) \leq 1$ for all vertices $v \in V$. Furthermore, a *directed tree* is a directed forest $G = \langle V, E\rangle$ with a unique root, i.e., $\deg^-(v) = 0$ holds for at most one $v \in V$ (for exactly one $v \in V$ when $V \neq \emptyset$). Some directed example graphs illustrating the introduced acyclicity properties are depicted in Figure 1.

2.2 Undirected Graphs

An *undirected graph* $G = \langle V, E\rangle$ consists of a finite set V of *vertices* and a set E of undirected *edges* $\{u, v\}$ such that $u, v \in V$ and $u \neq v$; note that excluding singletons from E follows the standard concept of an undirected edge (cf. [18]). For some $v \in V$, $\deg(v) = |\{u \mid \{u, v\} \in E\}|$ denotes the number of edges including v; v is a *leaf* of G if $\deg(v) \leq 1$. A *path* $v_1, \ldots, v_k$ of length $k-1 \geq 0$ in G is defined as in the directed case except for replacing the requirement $\langle v_i, v_{i+1}\rangle \in E$ by $\{v_i, v_{i+1}\} \in E$ for all $1 \leq i < k$. A sequence $v_0, v_1, \ldots, v_k$ is a *cycle* of length $k \geq 3$ in G if $\{v_0, v_1\} \in E$, $v_0 = v_k$, and $v_1, \ldots, v_k$ is a path in G. Requiring the length to be at least 3 avoids that the definition becomes trivial, given that every undirected graph with some edge were cyclic otherwise.

An *undirected forest* is an undirected graph G such that there is no cycle in G. An *undirected tree* is an undirected forest G such that, for every pair $v_1, v_k \in V$, there is some path from v_1 to v_k in G. Moreover, *chordal graphs* form an interesting generalization of forests allowing for triangles as basic building blocks. More precisely, an undirected graph $G = \langle V, E\rangle$ is *chordal* if, for every cycle $v_0, \ldots, v_k$ of length $k > 3$ in G, there is some edge $\{v_i, v_j\} \in E$ such that $1 \leq i < j \leq k$ and $2 \leq j - i \leq k - 2$. For example, when $k = 4$, then $\{v_1, v_3\} \in E$ or $\{v_2, v_4\} \in E$ is required.

3 Encodings

In this section, we gradually develop encodings of the graph-theoretic concepts introduced in the previous section. We illustrate different encodings by means of first-order specifications in the input language of the ASP grounder GRINGO [26], using a subset of the language such that deciding the existence of a solution is NP-complete [38]. The underlying principles, however, are of general applicability, and we outline particularities of respective SAT, DL, and LP formulations.

(a) Cyclic (b) Acyclic (c) Forest (d) Tree

Fig. 1: Directed example graphs with five vertices each

```
1  #const n=5.
2  node(1..n).
3  pair(X,Y) :- node(X;Y), X != Y.
4  { edge(X,Y) } :- pair(X,Y).
```

Fig. 2: Encoding part for generating directed graphs (tight; size $\mathcal{O}(|E|) = \mathcal{O}(|V|^2)$)

3.1 Directed Acyclic Graphs

To begin with, we consider directed graphs and their properties, focusing on the distinction of cyclic and acyclic graphs like the ones depicted in Figure 1(a) and 1(b).

Rules describing the (non-deterministic) generation of directed graphs are shown in Figure 2. The predicate `node`/1 provides the labels $1, \ldots, \mathbf{n}$ for vertices, where $\mathbf{n}$ is an integer constant standing for the number of vertices, and the symmetric predicate `pair`/2 represents the domain of directed edges given by all pairs of distinct vertices. Any subset of these pairs can be generated via the choice rule in line 4, permitting instances of `edge`/2 to hold without further preconditions.[1] For example, the atoms characterizing the directed graph in Figure 1(a) are `edge(1,2)`, `edge(1,3)`, `edge(2,4)`, `edge(3,2)`, `edge(4,5)`, and `edge(5,3)`.

Acyclicity Checking To make sure that a generated directed graph $\langle V, E \rangle$ is acyclic, we may check whether there is a strict partial order $<$ over vertices in V such that $u < v$ if $\langle u, v \rangle \in E$. The rules in Figure 3 encode this approach in an inductive fashion, where atoms of the form `order(`u`,`v`)` indicate the absence of cycles involving an edge candidate $\langle u, v \rangle$ and `order(`u`)` expresses the same for vertices u. Given $\mathbf{n}$=5, the following (simplified) ground instances are obtained when the label `3` is substituted for `X`:

```
order(3,1) :- not edge(3,1).
order(3,1) :- order(1).
order(3,2) :- not edge(3,2).
order(3,2) :- order(2).
order(3,4) :- not edge(3,4).
order(3,4) :- order(4).
order(3,5) :- not edge(3,5).
```

[1] When defining `pair`/2 such that atoms of the form `pair(`v`,`v`)` are included, picking `edge(`v`,`v`)` would immediately yield a cycle, and any of the acyclicity tests described below would discard such a choice.

```
5  order(X,Y) :- pair(X,Y), not edge(X,Y).
6  order(X,Y) :- pair(X,Y), order(Y).
7  order(X) :- node(X), order(X,Y) : pair(X,Y).

9  :- node(X), not order(X).
```

Fig. 3: Inductive bottom-up encoding of acyclicity test (non-tight; size $\mathcal{O}(|E|)$)

```
5  order(X,Y,1..n) :- pair(X,Y), not edge(X,Y).
6  order(X,Y,N-1)  :- pair(X,Y), order(Y,N).
7  order(X,N) :- node(X), order(X,Y,N) : pair(X,Y), N = 2..n.
8  order(X)   :- node(X), order(X,Y,1) : pair(X,Y).
```

Fig. 4: Unfolded bottom-up derivation of `order`/1 (tight; size $\mathcal{O}(|E|\times|V|)$)

```
order(3,5) :- order(5).
order(3)   :- order(3,1), order(3,2), order(3,4), order(3,5).
:- not order(3).
```

As expressed by `order(3,`v`)`, vertex `3` fulfills the condition of $<$ relative to a potential successor $v \in \{1,2,4,5\}$ if there is no edge from `3` to v or if v has no path to `3`. Then, `order(3)` is derived once the existence of a cycle through vertex `3` can be safely excluded for all potential successors v of `3`. In turn, the positive prerequisites of corresponding ground rules for other vertices along with well-foundedness of derivations, as required in ASP, prohibit `order(3)` to hold if `3` belongs to some cycle. Given this, the integrity constraint denying models such that `order(3)` is false rejects any directed graph with a cycle through `3`, and respective ground rules establish the same for other vertices as well.

Reconsidering the example graph in Figure 1(a), we have that `order(`u`,`v`)` can be derived for distinct vertices u and v such that $\langle u, v\rangle$ is not an edge. Since each vertex u has some successor v, there still is some atom `order(`u`,`v`)` for u that cannot be concluded in this way, e.g., `order(5,3)` for vertex `5`. Hence, `order(`u`)` remains underivable for all $u \in \{1,2,3,4,5\}$, and the cyclic graph in Figure 1(a) is rejected by means of the rules in Figure 3. Unlike this, vertex `5` has no successor in the acyclic graph shown in Figure 1(b), so that `order(5,`v`)` is derivable for $v \in \{1,2,3,4\}$. This in turn yields `order(5)`, `order(4,5)`, and `order(4)`, given that `5` is the only successor of vertex `4`. Similarly, the derivation of `order(2,4)` leads to `order(2)`, and the establishment of `order(3,2)`, `order(3,4)`, and `order(3,5)` for the successors of vertex `3` allows for deriving `order(3)`. Finally, `order(1)` can be concluded in view of `order(1,2)` and `order(1,3)`. As derivable atoms are compatible with (ground instances of) the integrity constraint in line 9 of Figure 3, the graph in Figure 1(b) passes the acyclicity test.

Encoding Variants The encoding in Figure 3 is *non-tight*, i.e., it relies on well-foundedness in the presence of circular positive dependencies on the ground level. Level mappings (cf. [29, 36, 33]) furnish an alternative mechanism to express well-founded derivations, where difference constraints or numerical variables allow for linear em-

beddings in DL and LP. A respective approach is taken by the *tight* ASP formulation of the predicate `order`/1 in Figure 4. In order to eliminate circular positive dependencies between rules' prerequisites and conclusions, the auxiliary predicates `order`/3 and `order`/2 include an additional argument for "step counting" that unfolds the derivation of a partial vertex order $<$ witnessing acyclicity. If such an order exists, its construction must be completed in at most **n** steps. For instance, the (total) order $1 < 3 < 2 < 4 < 5$ over vertices of the acyclic graph in Figure 1(b) is represented in terms of the atoms `order(5,2...5)`, `order(4,2...4)`, `order(2,2...3)`, and `order(3,2)` along with `order(`u`)` for $u \in \{1, 2, 3, 4, 5\}$. On the one hand, explicit step counting eliminates the potential of circular derivations, so that straightforward translations like completion [13] can be used to map the encoding in Figure 4 to SAT, DL, or LP. In fact, the tight ASP formulation of `order`/1 can be viewed as a directed counterpart of the SAT encoding for the undirected case in [14]. On the other hand, the step argument introduces an additional dimension increasing the number of atoms as well as constraints. For directed graphs $\langle V, E\rangle$, the size of ground instances grows from $\mathcal{O}(|E|)$ for the encoding in Figure 3 to $\mathcal{O}(|E|\times|V|)$, thus shifting from linear to quadratic space.

Variants of the encodings presented so far can be obtained by replacing the *unconditional* generation of edges in line 4 of Figure 2 by the following (non-deterministic) choice rule:

```
{ edge(X,Y) } :- pair(X,Y), order(Y).
```

The additional prerequisite `order(Y)` necessitates the absence of cycles through a vertex substituted for `Y` before admitting any edge from a predecessor substituted for `X`. Hence, the generation of edges with the above choice rule progresses successively from the leaves of a directed acyclic graph. Whether such *conditional* edge generation is advantageous or not is empirically investigated in Section 4. While the *leaf encodings* in Figure 3 and 4 describe "bottom-up" traversals starting from the leaves of a directed graph, acyclicity tests can likewise be performed "top-down" from roots, and a conditional choice rule similar to the one above can optionally be used in *root encodings* instead of line 4 in Figure 2. Without other side constraints, bottom-up and top-down traversals appear fully symmetric, but in the contexts of forests and trees, considered in the following subsections, the orientation of acyclicity tests may interact or interfere.

For a complement, we also consider acyclicity checking via the transitive closure of `edge`/2. In fact, the inductive formulation in Figure 5 is similar to encodings proposed in the literature [7, 10, 11, 6]. The rule in line 6 provides the base case for deriving the full transitive closure, represented in terms of the predicate `trans`/2, via the rule in line 7. Then, `order(`v`)` is derived in line 8 when there is no cycle connecting a vertex $v \in V$ to itself. Given this, the integrity constraint in line 9 of Figure 3 can be used as is for rejecting cyclic graphs. A tight formulation of the acyclicity test based on transitive closure is given in Figure 6. Observe that the integrity constraints in line 6 and 7 resemble respective inductive rules and require `trans(`v_0`,`v_k`)` to hold if there is some cycle $v_0, v_1, \ldots, v_k$. Since integrity constraints do not provide derivations for atoms, the choice rule in line 5 additionally expresses that instances of the predicate `trans`/2 may be generated unconditionally. As a consequence, a directed acyclic graph can have several corresponding (stable) models. For instance, 160 models describe the directed tree displayed in Figure 1(d). All of them entail `trans(1,`u`)` for

```
6  trans(X,Y) :- edge(X,Y).
7  trans(X,Z) :- edge(X,Y), trans(Y,Z).
8  order(X) :- node(X), not trans(X,X).
```

Fig. 5: Inductive transitivity encoding of acyclicity test (non-tight; size $\mathcal{O}(|E|\times|V|)$)

```
5  { trans(X,Y) } :- node(X), node(Y).
6  :- edge(X,Y), not trans(X,Y).
7  :- edge(X,Y), trans(Y,Z), not trans(X,Z).
8  order(X) :- node(X), not trans(X,X).
```

Fig. 6: Flattened transitivity encoding of acyclicity test (tight; size $\mathcal{O}(|E|\times|V|)$)

$u \in \{2,3,4,5\}$, `trans(3,4)`, and `trans(3,5)`, whereas different combinations of the atoms `trans(2,`v`)` and `trans(`v`,2)` for $v \in \{3,4,5\}$ as well as `trans(4,5)` and `trans(5,4)` are admissible in addition. Regarding space complexity, the *transitivity encodings* in Figure 5 and 6 yield the same as the unfolded bottom-up derivation of `order`/1 in Figure 4, i.e., $\mathcal{O}(|E|\times|V|)$. In particular, note that the non-tight ASP formulation in Figure 5 is less compact than the one in Figure 3.

Unlike the flattened transitivity encoding in Figure 6, the predicate definitions in Figure 3, 4, and 5 are *stratified* (cf. [1]). That is, when instances of the predicate `edge`/2 are fixed, there is at most one (well-founded) model that can be deterministically determined. In the absence of circular positive dependencies, as with ground instances of the rules in Figure 4, the unique model or unsatisfiability is also obtained by evaluating (via unit propagation) the rules' completion. Except for variants based on Figure 6 or the introduction of prerequisites for conditional edge generation, all encoding parts in the sequel are stratified, so that the choice of edges is the only source of non-determinism.

3.2 Directed Forests

In order to switch from directed acyclic graphs $\langle V, E\rangle$ to the more restrictive notion of directed forests, we have to make sure that $\deg^-(v) \leq 1$ holds for every vertex $v \in V$. For instance, the acyclic graph shown in Figure 1(b) is not a forest because $\deg^-(2) = \deg^-(4) = \deg^-(5) = 2$. The (sub)graph in Figure 1(c), however, is a forest with roots 1 and 3, given that $\deg^-(1) = \deg^-(3) = 0$ and $\deg^-(2) = \deg^-(4) = \deg^-(5) = 1$.

Five alternative approaches to test the additional requirement of at most one incoming edge per vertex are encoded by the rules in Figure 7(a)–(e). The integrity constraint shown in Figure 7(a) checks pairwise mutual exclusion by, for every vertex, enumerating distinct predecessor candidates and denying the joint occurrence of incoming edges for each pair of potential predecessors. While this approach is straightforward, its cubic space complexity, i.e., $\mathcal{O}(|E|\times|V|) = \mathcal{O}(|V|^3)$, is a major bottleneck for scalability. The usage of one cardinality constraint per vertex in Figure 7(b) allows for a more compact (quadratic) representation since pairs of predecessor candidates are not explicitly enumerated. However, cardinality constraints of arbitrary arity are not directly available in SAT and DL input languages, and the rules in Figure 7(c)–(e) encode *normalizations*

```
10  :- node(Z), edge(X;Y,Z), X < Y.
```

(a) Pairwise mutual exclusion (tight; size $\mathcal{O}(|E|\times|V|)$)

```
10  :- node(Y), 2 { edge(X,Y) }.
```

(b) Cardinality constraint (tight; size $\mathcal{O}(|E|)$)

```
10  unique(X,1,1)    :- node(X).
11  unique(X,n,-1)   :- node(X).
12  unique(X,Y+D,D) :- unique(X,Y,D), node(Y+D), not edge(Y,X).
13  unique(X) :- unique(X,Y,1;-1), n-2 < n*|X-Y|.
14  :- node(X), not unique(X).
```

(c) Bidirectional traversal (tight; size $\mathcal{O}(|E|)$)

```
10  unique(X,n,0)    :- node(X).
11  unique(X,n,1)    :- node(X), not edge(n,X).
12  unique(X,Y-1,0) :- unique(X,Y,1), node(Y-1), X != Y-1.
13  unique(X,Y-1,C) :- unique(X,Y,C), node(Y-1), not edge(Y-1,X).
14  :- node(X), not unique(X,1,0).
```

(d) Linear traversal (tight; size $\mathcal{O}(|E|)$)

```
10  unique(X,n,Y,0)      :- node(X;Y).
11  unique(X,n,Y,1)      :- node(X;Y), not edge(Y,X).
12  unique(X,I,I,C)      :- unique(X,J,J,C), I = (J+1)/2,
                            1 < J, J < 2*I.
13  unique(X,I,Y/2,C/2) :- unique(X,J,Y,C1), I = (J+1)/2, 0 < C,
                           unique(X,J,Y-1,C2), C = C1+C2, Y\2 == 0.
14  :- node(X), not unique(X,1,1,0).
```

(e) Tournament traversal (tight; size $\mathcal{O}(|E|)$)

Fig. 7: Forest property tests for directed acyclic graphs

(cf. [4]) for the purpose of checking mutual exclusion. To this end, it is sufficient to propagate the existence of some element with a certain property, here the property to be a predecessor, to other potential candidates. In fact, the schemes described in the following are rather simple, and the required space is directly proportional to graph size in terms of edge candidates, while generalizations [39, 20] of the encodings in Figure 7(d) and 7(e) are applicable to arbitrary cardinality constraints. On the other hand, the approach in Figure 7(c) addresses mutual exclusion testing in a dedicated fashion, and we are unaware of any corresponding proposal in the literature.

Bidirectional Traversal The basic idea of the encoding in Figure 7(c) is to traverse the potential predecessors of a vertex from both ends of an (arbitrary) order. For convenience, we here rely on the natural order given by vertex indexes along with the background information that any pair of distinct vertices may possibly be connected by an edge (cf. line 3 of Figure 2). In case of a more restrictive predecessor selection, the

candidates to be traversed and the order among them can of course be localized. However, the entry points for traversing potential predecessors of a vertex substituted for `X` are in line 10 and 11 of Figure 7(c) taken to be the vertices with the smallest and the greatest label, i.e., `1` and the integer constant represented by **n**. Starting from them, the rule in line 12 expresses that proceeding "upwards" or "downwards", respectively, to the next vertex in direction `D` is admissible if a current vertex substituted for `Y` does not have an edge to `X`. That is, a vertex substituted for `X` has at most one incoming edge if and only if some `Y` is encountered from both ends of the underlying order, as indicated by the atoms `unique(X,Y,1)` and `unique(X,Y,-1)`. Except for the corner case **n**=1 (in which `X`=`Y` is admitted by the condition $\mathbf{n} - 2 < \mathbf{n} * |\mathtt{X} - \mathtt{Y}|$), a respective `Y` can be assumed to be different from `X`, which per `X` eliminates one ground instance of the rule in line 13 for deriving `unique(X)`. Finally, the integrity constraint in line 14 requires ground instances of `unique(X)` for `X`=1, ..., `X`=**n** to hold, thus denying any graph with more than one incoming edge for some vertex.

The test based on bidirectional traversal filters the example graph in Figure 1(b) as follows. While `unique(1,`u`,`d`)` for $u \in \{1,2,3,4,5\}$ and $d \in \{1,-1\}$, `unique(3,1,1)`, `unique(3,1,-1)`, and thus `unique(1)` and `unique(3)` are derivable, such atoms remain underivable for other vertices. For instance, traversing the potential predecessors of vertex `2` yields `unique(2,1,1)` and `unique(2,`u`,-1)` for $u \in \{3,4,5\}$ only, so that `unique(2)` cannot be concluded from `unique(2,Y,1)` and `unique(2,Y,-1)` for any common ground substitution of `Y`. Unlike this, the test succeeds for the forest shown in Figure 1(c). In particular, `unique(2)` as well as `unique(`v`)` for $v \in \{4,5\}$ are derived in view of `unique(2,1,1)` and `unique(2,1,-1)` or `unique(`v`,3,1)` and `unique(`v`,3,-1)`, respectively.

Linear Traversal The approach encoded in Figure 7(d) inspects potential predecessors of a vertex substituted for `X` "downwards" from **n** while maintaining a counter flag in the third argument of the predicate `unique`/3. The latter is unconditionally initialized with `0` in line 10, and the rule in line 12 expresses that the flag can be decreased from `1` to `0` at any point (other than `X`) in traversing the indexes of vertices. Atoms of the form `unique(`v`,`u`,1)` indicate the absence of edges leading to v for the vertices with labels $u, \ldots, \mathbf{n}$, which is in line 11 checked for **n**, as well as when proceeding to any predecessor candidate with the next smaller label by applying the rule in line 13. The latter rule also forwards flag `0` to indicate the existence of at most one predecessor for a vertex v among $u, \ldots, \mathbf{n}$, and requiring `unique(`v`,1,0)` to hold by means of the integrity constraint in line 14 thus restricts the number of incoming edges per vertex to one or none. Note that the encoding in Figure 7(d) provides an ASP formulation of the well-known sequential counter approach from SAT [39] for the particular case of cardinality limited to at most one (cf. [34, 20]).

Tournament Traversal While linear traversal inspects vertices in (lexicographical) order, the encoding in Figure 7(e) aims at a symmetric partitioning of predecessor candidates. To this end, potential predecessors are viewed as leaves of a binary tree of depth $\lceil \log_2(\mathbf{n}) \rceil$, where `unique(X,`**n**`,Y,0)` and `unique(X,`**n**`,Y,1)` in line 10 and 11 provide base cases indicating whether the vertex substituted for `Y` may have an edge to `X`

or not, respectively. The rule in line 13 then combines two such atoms, inspecting an even label `Y` along with `Y-1` to derive `unique(X,(J+1)/2,Y/2,`c`)` for $c \in \{0,1\}$: the numbers `(J+1)/2` and `Y/2` denote the round of traversal as well as the position of an outcome, and c=`0` or c=`1` represents again that the investigated predecessor candidates may have some or no edge to `X`. If the number of positions in a round is odd, the rule in line 12 additionally forwards previous outcomes for the last position, lacking an even partner, to the next round. Considering for instance vertex `5` and its incoming edge from `3` in Figure 1(c), the atom `unique(5,3,2,0)` is derived from `unique(5,5,3,0)` and `unique(5,5,4,1)`. In addition, `unique(5,3,1,0)` and `unique(5,3,1,1)` jointly indicate that the vertices with labels `1` and `2` have no edge to `5`. Further combining successive outcomes yields `unique(5,2,1,0)` as well as `unique(5,2,2,1)` to eventually derive `unique(5,1,1,0)`, expressing that (at most) one edge to `5` has been encountered in traversing all candidates. In general, the integrity constraint in line 14 checks the respective condition for every vertex to filter directed acyclic graphs that are no forests. The presented tournament traversal approach to pairwisely compare elements as well as their aggregated outcomes resembles the so-called "commander encoding" from SAT [32], whose generalization [20] is applicable to arbitrary cardinality constraints.

Finally, note that (ground instances of) the encoding variants in Figure 7 do not involve circular positive dependencies, so that well-foundedness is not a bottleneck for corresponding SAT, DL, and LP formulations. However, SAT and DL input languages lack native support for cardinality constraints like the one in Figure 7(b),[2] and the scalability of explicit predecessor candidate pair enumeration, as in Figure 7(a), suffers from (cubic) space complexity. The normalizations in Figure 7(c)–(e) exemplify ways to encode cardinality limitations to at most one compactly, where the linear and tournament traversal formulations amount to ASP counterparts of respective approaches from SAT.

3.3 Directed Trees

Directed trees are directed forests $\langle V, E\rangle$ with (at most) one root. That is, all but one vertex in V must have some incoming edge, and since there may be at most one incoming edge per vertex, $|V|-1$ edges are required. In fact, either condition is sufficient to check that $\langle V, E\rangle$ is a tree. Moreover, $|V|-1$ edges are needed to connect the undirected version $\langle V, \{\{u,v\} \mid \langle u,v\rangle \in E\}\rangle$ of $\langle V, E\rangle$, so that connectedness provides a third criterion to distinguish trees. Figure 8 displays encodings of these equivalent conditions.

Edge Counting The cardinality constraint in Figure 8(a) holds if a directed forest $\langle V, E\rangle$ includes $|V|-1$ edges, and the (ground instance of the) integrity constraint in line 15 checks this condition. In view of $\mathbf{n}$=5, the three edges of the directed forest shown in Figure 1(c) are insufficient to pass the test, whereas it succeeds for the directed tree with four edges in Figure 1(d). Given that any pair of distinct vertices provides an edge candidate according to the declaration in line 3 of Figure 2, a ground instance of

[2] Cardinality constraints can be linearly translated to DL, e.g., by adopting the mapping in [35].

```
15  :- not n-1 { edge(X,Y) }.
```

(a) Edge counting (tight; size $\mathcal{O}(|E|)$)

```
15  child(Y) :- edge(X,Y).
16  :- 2 { not child(X) : node(X) }.
```

(b) Root counting (tight; size $\mathcal{O}(|E|)$)

```
15  reach(1) :- 0 < n.
16  reach(X) :- reach(Y), edge(X,Y).
17  reach(Y) :- reach(X), edge(X,Y).
18  :- node(X), not reach(X).
```

(c) Connectedness (non-tight; size $\mathcal{O}(|E|)$)

Fig. 8: Tree property tests for directed forests

the cardinality constraint over edges involves $|V|^2 - |V|$ atoms, and the bound $|V| - 1$ goes beyond mutual exclusion testing, as encoded by the normalizations in Figure 7(c)–(e). While generalizations of the linear and tournament traversal approaches allow for dealing with greater cardinalities too, growing in space proportionally to the cardinality bound (cf. [39, 20]), we omit them here for brevity. However, since the number of edges cannot exceed $|V| - 1$, the following integrity constraint is valid as well (if $V \neq \emptyset$):

```
:- n { edge(X,Y) }.
```

This can be viewed as a redundant formulation of an implied property of directed forests, and auxiliary atoms introduced in normalizations of the cardinality constraint in line 15 of Figure 8(a) may be reused to redundantly test the implied condition too. The effect of bounding the number of edges explicitly from above is assessed in Section 4.

Unique Root The second approach encoded in Figure 8(b) focuses on the uniqueness of a root. To this end, the predicate `child`/1 is derived from `edge`/2 in line 15 to indicate vertices targeted by some edge. The (ground instance of the) integrity constraint in line 16 then denies graphs in which several vertices have no incoming edge, thus limiting the number of roots to (at most) one. For instance, the atoms `child(1)` and `child(3)` are not derived from `edge(1,2)`, `edge(3,4)`, and `edge(3,5)`, representing the edges of the forest shown in Figure 1(c), so that it is rejected by means of

```
:- 2 { not child(1), not child(2),
       not child(3), not child(4), not child(5) }.
```

For the tree in Figure 1(d), `child(3)` is additionally derivable from `edge(1,3)`, and thus the respective test succeeds. Similarly to the cardinality constraint in Figure 7(b), limiting incoming edges to at most one per vertex, the requirement of a unique root addresses mutual exclusion. Hence, normalizations analogous to those in Figure 7(c)–(e) are applicable. Notably, the auxiliary atoms introduced in either of the latter normalizations can also be explored to replace the rule in line 15 by these:

```
child(X) :- node(X), not unique(X,X, 1).
child(X) :- node(X), not unique(X,X,-1).
```
(1)

```
child(X) :- node(X), not unique(X,1,1).
```
(2)

```
child(X) :- node(X), not unique(X,1,1,1).
```
(3)

In combination with the bidirectional traversal encoding in Figure 7(c), the rules in (1) conclude the existence of some edge to a vertex substituted for `X` from a predecessor that blocks proceeding "upwards" or "downwards" to `X` along the order of candidates. With the linear and tournament traversal approaches in Figure 7(d) and 7(e), vertex `X` must have some predecessor if the counter flag `1` remains underivable as an outcome of inspecting all predecessor candidates, which is checked in the prerequisite of the rule in (2) or (3), respectively. Either of the alternative derivations preserves the role of the predicate `child`/1 to indicate the vertices with some predecessor, so that the integrity constraint in line 16 of Figure 8(b) or corresponding normalizations can be used unmodified to test the uniqueness of a root.

Connectedness The encoding in Figure 8(c) checks that the undirected version of a (directed) graph $\langle V, E\rangle$ is connected. As a matter of fact, $|V|-1$ edges are necessary to connect all vertices in V, so that connectedness provides a criterion for testing whether a forest is a tree. In order to inductively explore connected vertices, the rule in line 15 picks the vertex with label `1` (if $V \neq \emptyset$) as an (arbitrary) starting point to proceed in either direction along edges by applying the rules in line 16 and 17. As a consequence, the predicate `reach`/1 provides the vertices connected to `1`, and (ground instances of) the integrity constraint in line 18 require all vertices of a graph to be connected. Reconsidering the forest in Figure 1(c), disconnectedness is revealed by deriving `reach(1)` and `reach(2)` but lacking `reach(`v`)` for $v \in \{3, 4, 5\}$. As the undirected versions of the other three graphs depicted in Figure 1 are connected, such atoms `reach(`v`)` are derivable in addition from the instances of `edge`/2 characterizing these graphs, so that they are compatible with the integrity constraint in line 18. Like the inductive derivation of `order`/1 in Figure 3, the connectedness encoding in Figure 8(c) induces circular positive dependencies, and a step argument ranging from `1` to **n**, similar to the one added in Figure 4, can be introduced for obtaining a tight variant. Finally, note that the orthogonal tests encoded in Figure 8 are not mutually exclusive but may be freely combined in order to inspect several complementary properties of a directed tree at once.

3.4 Undirected Forests and Trees

When switching from directed edges $\langle u, v\rangle$ to undirected edges $\{u, v\}$, several directed paths connecting two vertices yield an undirected cycle. Therefore, acyclic graphs coincide with forests in the undirected case. However, given that roots cannot (necessarily) be identified unambiguously when edges lack orientation, only leaves v satisfying $\deg(v) \leq 1$ can be distinguished. For instance, the undirected version (ignoring edge orientation) of the example graph in Figure 1(d) has the leaves 2, 4, and 5, whereas no unique root can be determined among the inner vertices 1 and 3. In what follows,

```
5  order(X,Y) :- pair(X,Y), not edge(X,Y).
6  order(X,Y) :- pair(X,Y), order(X).
7  order(X,Y) :- pair(X,Y), order(Y).
8  order(X) :- node(X), n-2 { order(X,Y) : pair(X,Y),
                              order(Y,X) : pair(Y,X) }.
9  :- node(X), not order(X).
```

Fig. 9: Inductive encoding of undirected acyclicity test (non-tight; size $\mathcal{O}(|E|)$)

we focus on the description of encoding modifications allowing for a transition from directed to undirected graphs.

As edge candidates represented by the atoms `pair(`u`,`v`)` and `pair(`v`,`u`)` refer to the same set $\{u, v\}$ of distinct vertices, first of all, a canonical edge representation is established by replacing the rule in line 3 of Figure 2 as follows:

```
3  pair(X,Y) :- node(X;Y), X < Y.
```

In this way, redundant representations are resolved via labels of vertices, e.g., the atom `edge(1,2)`, but not `edge(2,1)`, may be generated by applying the (non-deterministic) choice rule in line 4 of Figure 2. Beyond that, checking that an undirected graph is acyclic and thus a forest can be accomplished by rearranging the approach of the bottom-up acyclicity test in Figure 3 to handling leaves of an undirected graph. The basic idea of the encoding in Figure 9 is to successively derive `order(`v`)` for vertices v that do not belong to any cycle, as witnessed by `order(`u`)` for all but (at most) one vertex u in an edge $\{u, v\}$. Conversely, when each remaining vertex v has two or more edges $\{u, v\}$ such that `order(`u`)` does not hold, there must be some cycle through v.

Apart from line 8, the rules in Figure 9 are similar to those of the directed acyclicity test in Figure 3. The additional case in line 6 takes care of asymmetric edge representation in deriving an instance of `order(X,Y)` for an edge candidate $\{$`X,Y`$\}$ when `order(X)` or `order(Y)` indicates the absence of cycles involving $\{$`X,Y`$\}$. In view of `n-1` edge candidates per vertex substituted for `X`, the rule in line 8 applies once the existence of a cycle through `X` can be safely excluded. Its respective ground instance for `n`$=5$ and `X`$=$`3` is as follows:

```
order(3) :- 3 { order(1,3), order(2,3),
                order(3,4), order(3,5) }.
```

That is, when all but one among the edge candidates including vertex `3` do not belong to any cycle, then `3` cannot be part of a cycle either. Corresponding ground instances first allow for deriving `order(2)`, `order(4)`, and `order(5)` from instances of `edge`/2 characterizing the undirected version of the example graph in Figure 1(d). After concluding that the leaves along with edges including them do not contribute to any cycle, merely the atom `order(1,3)` remains open, so that `order(1)` and `order(3)` are derived in turn. In general, instances of `order`/1 are derivable for all vertices, as required in view of the integrity constraint in line 9, if and only if an undirected graph is acyclic and thus a forest.

Encoding Variants The cardinality constraint in line 8 of Figure 9 applies when all but one edge candidate including a vertex substituted for `X` cannot belong to any cycle. Given the uniqueness of a remaining edge candidate, normalizations analogous to those in Figure 7(c)–(e) can be used again. To this end, a rule indicating the absence of an edge for a candidate $\{\texttt{X}, \texttt{Y}\}$ is augmented with a second case based on `order(Y)`, expressing that there is no cycle through a vertex substituted for `Y`. In view of such positive prerequisites, neither the rule in line 8 of Figure 9 nor normalizations of this *non-tight* ASP formulation admit well-founded derivations of `order(`v_i`)` for vertices in a cycle $v_0, \ldots, v_k$, since each v_i is included in two (distinct) edges connecting it to its predecessor and successor along the cycle. However, an additional step argument like the one in Figure 4 allows for a *tight* encoding, simulating well-founded derivations without relying on circular positive dependencies, where $\lceil \mathtt{n}/2 \rceil$ steps are sufficient in the undirected case. In fact, in any but the final step of such a construction, the graph obtained by eliminating edges that include leaves from previous steps must yield at least two new leaves, or there certainly is some cycle otherwise. Moreover, the *unconditional* choice rule in line 4 of Figure 2 can likewise be replaced by the following two rules:

```
{ edge(X,Y) } :- pair(X,Y), order(Y).
{ edge(X,Y) } :- pair(X,Y), order(X).
```

The prerequisite `order(Y)` or `order(X)`, respectively, directs the generation of edges to proceed successively from leaves. The effect of such *conditional* edge generation is empirically assessed in Section 4.

Trees In order to distinguish undirected trees among forests, the approach to allow for one root only, available in the directed case, is no longer applicable. To see this, note that the undirected version of the tree depicted in Figure 1(d) has the inner vertices 1 and 3 along with the leaves 2, 4, and 5. However, when the edge $\{3, 4\}$ is removed, the graph becomes a disconnected forest with the same inner vertices and leaves. Hence, merely counting inner vertices or leaves does not provide sufficient information to identify trees or disconnected forests, respectively. Unlike this, the fact that $|V|-1$ edges are required to connect the vertices of a forest $\langle V, E\rangle$ applies both in the directed and the undirected case. Therefore, *edge counting* or checking *connectedness* are adequate approaches to distinguish undirected trees, and the respective encodings in Figure 8(a) and 8(c) as well as variants thereof can be used unmodified for this purpose.

3.5 Chordal Graphs

While cycles were not permitted in the directed and undirected graphs considered so far, chordal graphs may contain cycles, provided that they can be broken down into triangles. For instance, the undirected version of the example graph in Figure 1(a) is not chordal because the cycle $2, 4, 5, 3, 2$ of length 4 is accompanied by neither edge $\{2, 5\}$ nor $\{3, 4\}$. Since the latter is included in the undirected version of the graph in Figure 1(b), this graph is chordal. In fact, the other two cycles of length 4 or 5, i.e., $1, 2, 4, 3, 1$ and $1, 2, 4, 5, 3, 1$, are both broken by the edge $\{2, 3\}$.

As stated in [12], every chordal graph $\langle V, E\rangle$ is a clique, i.e., $E = \{\{u, v\} \mid u, v \in V, u \neq v\}$, or there are at least two vertices $v \in V$ such that $(V' \times V') \subseteq E$

```
5  scope(X,Y,Z) :- node(X;Y;Z), pair(Y,X) : not pair(X,Y),
                                Y < Z, pair(Z,X) : not pair(X,Z).
6  order(X,Y,Z) :- scope(X,Y,Z), not edge(X,Y), not edge(Y,X).
7  order(X,Y,Z) :- scope(X,Y,Z), not edge(X,Z), not edge(Z,X).
8  order(X,Y,Z) :- scope(X,Y,Z), edge(Y,Z).
9  order(X,Y,Z) :- scope(X,Y,Z), edge(Z,Y).
10 order(X,Y,Z) :- scope(X,Y,Z), order(Y).
11 order(X,Y,Z) :- scope(X,Y,Z), order(Z).
12 order(X) :- node(X), order(X,Y,Z) : scope(X,Y,Z).
13 :- node(X), not order(X).
```

Fig. 10: Inductive encoding of chordality test (non-tight; size $\mathcal{O}(|E|\times|V|)$)

for $V' = \{u \mid \{u, v\} \in E\}$, which means that the vertices u sharing some edge with v form a clique. Moreover, a graph is chordal if and only if the successive elimination of vertices "connected to a clique" leads to an empty graph. Reconsidering the undirected version of the graph in Figure 1(b), the neighbors 2 and 3 of 1 as well as the neighbors 3 and 4 of 5 form cliques, so that the vertices 1 and 5 can be eliminated. Since the remaining vertices 2, 3, and 4 induce a clique, they can be eliminated afterwards, thus certifying that the graph at hand is chordal. Unlike this, only vertex 1 can be eliminated when starting from the graph depicted in Figure 1(a), given that each of the other vertices has two remaining neighbors that do not share an edge.

The rules in Figure 10 encode the described approach to test chordality. For convenience, the auxiliary predicate `scope`/3 provides all pairs of distinct potential neighbors of a vertex substituted for `X` in line 5, i.e., lexicographically ordered pairs such that (asymmetric representations of) edges with `X` are among the candidates given by the predicate `pair`/2. For **n**=5 along with potential edges between any pair of distinct vertices, we thus obtain six instances of `scope(X,Y,Z)` per vertex `X`. In fact, the consideration of triples rather than isolated edges increases the required space to $\mathcal{O}(|E|\times|V|)$. However, note that the rule in line 5 adapts to the edge candidates represented by `pair`/2, so that the chordality test (as well as similar encoding parts presented above) can easily be narrowed in case of a more restrictive selection of edges. Like acyclicity tests in the previous sections, the rules from line 6 to 12 focus on the inductive derivation of `order`/1 to indicate vertices that become connected to a clique and can be eliminated successively. To this end, for any pair $\{\texttt{Y}, \texttt{Z}\}$ in the scope of a vertex `X`, some of the base cases encoded by the rules from line 6 to 9 applies if one of the edges $\{\texttt{X}, \texttt{Y}\}$ or $\{\texttt{X}, \texttt{Z}\}$ is excluded or if the edge $\{\texttt{Y}, \texttt{Z}\}$ is included in a graph.[3] Otherwise, the elimination of `Y` or `Z` is required as a prerequisite to derive a respective instance of `order(X,Y,Z)` via the rules in line 10 and 11. Given this, the rule in line 12 applies once `Y` or `Z` is eliminated for every pair $\{\texttt{Y}, \texttt{Z}\}$ such that the graph at hand includes both $\{\texttt{X}, \texttt{Y}\}$ and $\{\texttt{X}, \texttt{Z}\}$, whereas the edge $\{\texttt{Y}, \texttt{Z}\}$ is excluded. That is, the well-founded derivation of an instance of `order(X)` expresses that the vertex substituted for `X` becomes connected to a clique in the successive elimination of vertices

[3] The rule in line 9, provided for symmetry, could actually be skipped, given that `Y` and `Z` are lexicographically ordered in derivable instances of `scope(X,Y,Z)` as well as `edge(Y,Z)`.

fulfilling this condition. An incomplete elimination of vertices otherwise indicates that a graph is not chordal, in which case it is rejected in view of the integrity constraint in line 13. The *non-tight* encoding in Figure 10 induces circular positive dependencies on the ground level, which can again be eliminated by means of step counting in a *tight* variant, where $\lceil \mathtt{n}/2 \rceil$ steps are sufficient for any chordal graph with `n` vertices.

4 Evaluation

To test our encodings, we ran CLASP (3.1.0), both as an ASP and SAT solver, a prototype version of CLASP (3.2.0-R45682) implementing acyclicity checking [24, 3], the SAT solver LINGELING (ayv-86bf266-140429), the DL solver Z3 (4.3.2), and the LP solver CPLEX (12.6.0.0), all single-threaded and with default settings, on a cluster of Linux machines equipped with 2.6GHz AMD Opteron 2435 processors. (The imposed memory limit of 8GB has not been exceeded in any run.) We used GRINGO (3.0.5) to ground first-order ASP formulations and the translators LP2ACYC (1.13), LP2SAT (1.24), and ACYC2SOLVER (1.7), as utilized by translation-based systems that participated in the 2014 ASP competition [9], for converting ground instances to ASP with acyclicity checking [21], SAT [30], DL [31], or LP [33], respectively; grounding and translations were performed offline and are not included in the runtimes reported below. Note that the translations are based on completion [13] and straightforward for tight ASP formulations, while well-foundedness needs to be addressed in the non-tight case. The translations then introduce level mappings [29] to express inductive derivations, either by adding an encoding part of subquadratic size in SAT or in terms of linear representations relying on acyclicity checking on top of ASP or SAT, difference constraints in DL, or numerical variables in LP. That is, the automatic translations of non-tight ASP formulations are in most cases indeed more compact than their unfolded tight counterparts given in Section 3. Of course, it would be possible to formulate direct encodings in ASP with acyclicity checking, SAT, DL, or LP, yet we aim at investigating the general impact of encoding approaches (rather than fine-tuning) on the performance of solvers for ASP and related formalisms. Our experiments took synthetic benchmarks, reachability-oriented Hamiltonian cycle problems, and application-driven optimization tasks into account,[4] which are discussed in the following subsections.

4.1 Synthetic Benchmarks

In the first series of experiments, we perturbed the search space for graphs with desired properties by adding randomly generated XOR-constraints over edges, systematically varying both the length and the number of such constraints. The tables presented in the following provide average runtimes in seconds for 25 instances with $\mathtt{n}{=}25$ vertices each. Runs were limited to 1000 seconds, and timeouts are counted as 1000 seconds within average runtimes. An entry "—" marks that no run completed in time.

Table 1 gives average runtimes and numbers of solved instances in parentheses for deciding whether there is a solution subject to XOR-constraints, comparing the performance of the considered solvers on six different encodings of directed acyclic graphs,

[4] `http://research.ics.aalto.fi/software/asp/bench/`

directed acyclic graphs	leaf encoding		root encoding		transitivity encoding	
	non-tight	**tight**	**non-tight**	**tight**	**non-tight**	**tight**
CLASP	0.31 (25)	80.26 (23)	0.34 (25)	0.39 (25)	0.33 (25)	**0.28** (25)
CLASP/ACYC	0.25 (25)	40.21 (24)	0.23 (25)	40.24 (24)	1.51 (25)	**0.22** (25)
CLASP/SAT	1.03 (25)	0.86 (25)	2.38 (25)	1.37 (25)	6.36 (25)	**0.75** (25)
LINGELING	2.62 (25)	1.78 (25)	1.82 (25)	1.70 (25)	20.36 (25)	**0.50** (25)
Z3	**1.12** (25)	— (0)	1.48 (25)	— (0)	160.94 (25)	1.76 (25)
CPLEX	681.47 (11)	261.90 (20)	655.29 (13)	333.47 (20)	165.37 (22)	**13.73** (25)

Table 1: Average runtimes for directed acyclic graphs

where the shortest average runtime of each solver is highlighted in boldface. The non-tight and tight variants of the *leaf encoding* consist of the rules shown in Figure 2 along with the inductive or unfolded bottom-up acyclicity test, respectively, encoded in Figure 3 and 4. Their counterparts denoted by *root encoding* swap the orientation of acyclicity tests by traversing vertices top-down from roots instead of bottom-up from leaves. Moreover, the non-tight and the tight *transitivity encoding* utilize the rules in Figure 5 or 6, respectively, instead of bottom-up or top-down traversals. The runtimes of CLASP on ASP formulations in the first row exhibit no significant differences between the three non-tight encoding variants, and the performance on the tight root and the tight transitivity encoding is also comparable, whereas the average runtime on the tight leaf encoding is two orders of magnitude longer. Given that bottom-up and top-down acyclicity tests (without further side constraints) are symmetric, we attribute this performance gap to search heuristics of CLASP, also taking into account that the additional atoms for step counting in the tight case may mislead the search. Similar effects can be observed in the second row showing the runtimes of CLASP with acyclicity checking. The performance of this prototype version suffers likewise on the tight leaf and root encoding variants (one timeout each), where acyclicity checking is not utilized at all. On the corresponding non-tight instances, acyclicity checking turns out to be competitive to the standard mechanism of CLASP for guaranteeing well-founded derivations, while it produces some unnecessary overhead on the non-tight transitivity encoding. The latter is not incurred by the tight transitivity encoding, which again leads to robust performance. The runtimes of CLASP as SAT solver and LINGELING in the third and fourth row indicate the competitiveness of translations to SAT, where the tight ASP formulations have some advantage over the level mappings introduced when translating non-tight instances. In fact, investigating search parameters like conflicts and decisions yields a payoff between smaller size and more exposed problem structure, and the latter turns out to be beneficial. Moreover, the tight transitivity encoding is particularly advantageous here, but its non-tight counterpart that is more intricate to translate without being more compact is the worst option. A similar effect can be observed when comparing the average runtimes of Z3 and CPLEX on DL or LP translations, respectively, on the tight and the non-tight transitivity encoding. The performances of both systems, however, diverge on the encodings of bottom-up or top-down acyclicity tests, where Z3 copes much better with non-tight ASP formulations while CPLEX behaves the other way round. A likely explanation is that DL reasoning merely checks the existence of

directed forests	trees	**leaf encoding**		**root encoding**		**leaf encoding**		**root encoding**	
		non-tight	**tight**	**non-tight**	**tight**	**non-tight**	**tight**	**non-tight**	**tight**
pairwise		**5.93**	8.18	7.44	8.95	**2.29**	3.73	2.78	3.02
cardinality		**3.69**	6.41	5.30	5.11	**2.14**	4.09	2.77	5.38
bidirectional		**4.68**	6.44	6.14	6.94	5.46	6.47	**2.87**	3.99
linear		7.29	8.47	8.50	**7.08**	**8.42**	11.06	8.93	46.92
tournament		12.36	8.56	**8.41**	8.61	11.56	**5.09**	11.68	7.81
				counting		10.16	28.25	15.86	**7.82**
				counting/ub		12.81	16.43	**12.18**	38.47
				connectedness		**2.30**	5.85	2.72	46.07

Table 2: Average runtimes for directed forests and trees

numerical values such that all difference constraints can be satisfied, whereas numerical variables are indeed assigned in LP.

Average runtimes of CLASP on ASP formulations of directed forests are shown on the left-hand side of Table 2. We do not apply translators here to confine the amount of data and in view of their nonuniform treatment of cardinality constraints used in some encoding variants,[5] whereas CLASP handles cardinality constraints in ground instances of ASP rules natively. Moreover, we concentrate on bottom-up and top-down acyclicity tests, as the experiments in Section 4.2 demonstrate that encodings based on transitive closure, in particular, the non-tight ASP formulation, do not scale up well when the graph size increases. Given that CLASP solved all 25 instances in time with each encoding variant, we omit this number but only highlight the shortest average runtime per row in boldface. The first five columns provide runtime results on the leaf and root encoding variants of directed acyclic graphs augmented with either set of rules in Figure 7(a)–(e) for limiting predecessors to at most one per vertex. Although switching to the more restrictive notion of directed forests reduces the admissible outcomes and makes the search for a solution harder, CLASP performs well on all encoding variants. The average runtimes, however, indicate some advantage of the most compact formulation in Figure 7(b), relying on cardinality constraints for limiting incoming edges to at most one per vertex, while the overhead of normalizations is not dramatically large either. Similarly, non-tight and tight encoding variants are at eye level, and the latter sometimes outweigh the increased size of ground instances by search improvements in view of the unfolded structure, for instance, on the root encodings augmented with cardinality constraints. In general, the bottom-up and top-down acyclicity tests described by the leaf and root encoding variants have some impact on search performance, but there is no clear winner among them. Given that the effect of interactions or interferences between encoding parts and search heuristics is difficult to predict, systematic empirical investigation seems indispensable for fine-tuning encodings to specific applications.

The right-hand side of Table 2 provides average runtimes for the encoding approaches in Figure 8(a)–(c) to distinguish directed trees among forests. Since the cardinality constraint in Figure 7(b) turned out to be effective for ruling out directed acyclic graphs that are no forests, unless noted otherwise, we use it as basis for the encoding

[5] The LP2NORMAL2 tool [4] allows for normalizing cardinality constraints before translations.

variants investigated in the following. Although the approach of *counting* over all edge candidates, encoded in Figure 8(a), could be expected to yield poor performance, the average runtime of CLASP is surprisingly short on the tight root encoding. In that context, the addition of a second cardinality constraint "*ub*" to redundantly assert $|V|-1$ as upper bound on the number of edges of a directed tree $\langle V, E\rangle$ deteriorates performance, so that the tight root encoding takes the last place among all edge counting variants with additional upper bound. Checking the *uniqueness* of a root by means of the same encoding approaches as presented for forests in Figure 7 leads to more robust performance, which can be observed in the first three rows. While mutual exclusion via the enumeration of pairs or a cardinality constraint, respectively, rely on the rule in line 15 of Figure 8(b) to derive edge targets, we used its dedicated variants in (1), (2), and (3) for the bidirectional, linear, and tournament traversal approaches, applied to limit both incoming edges and roots to at most one. Interestingly, pairwise mutual exclusion turns out to be successful on tight instances, yet closely followed by uniqueness checking via cardinality constraints or bidirectional traversal, for which non-tight ASP formulations have some advantage over tight ones. The linear and tournament traversal approaches, however, yield some difficulties on tight or non-tight encoding variants, respectively.[6] Moreover, the alternative of checking the *connectedness* of a directed forest, as encoded in Figure 8(c), exhibits a robust performance of CLASP on non-tight ASP formulations. In summary, it appears promising to encode the properties of directed trees in terms of a limitation to (at most) one root and/or connectedness, where manifold combinations with underlying encoding parts aiming at forests can in principle be conceived.

Turning from the directed case to undirected graphs, the left-hand side of Table 3 shows average runtimes of CLASP (and numbers of solved instances in parentheses) on several variants of the encoding in Figure 9. Since there is no apparent way to perform an acyclicity test via the transitive closure of `edge`/2 and top-down from roots or inner vertices, respectively, encoding variants are, for one, obtained by replacing the cardinality constraint in line 8 of Figure 9 by normalizations analogous to those in Figure 7(c)–(e). For another, we focus on *conditional edge generation* via choice rules requiring the absence of cycles through a vertex as a prerequisite for the addition of edges including the vertex. The comparably long runtimes point at increased difficulty of search for a forest in which edges lack orientation, and non-tight as well as tight encoding variants lead to timeouts on 17 or more of the 25 instances. While the tournament traversal approach to check whether (at most) one edge including a vertex remains as yet unchecked to not belong to any cycle is competitive on non-tight instances, the use of cardinality constraints yields shortest average runtimes on tight ones, even though six solved instances in the tight case of unconditional edge generation are not optimal. Since conditional edge generation, where the prerequisites of choice rules formulate the acyclicity requirement in a redundant fashion, does not exhibit a clear performance trend, we do not further include respective runtime results for other graph structures.

The right-hand side of Table 3 provides average runtimes on encodings of undirected trees, combining the previously considered acyclicity test approaches with *edge counting* and/or *connectedness* checking, as encoded in Figure 8(a) and 8(c). Apart from

[6] The average runtimes of 46.92 and 46.07 seconds include one timeout each, while CLASP solved all 25 instances in time otherwise.

undirected		**unconditional**		**conditional**		**edge counting**		**connectedness**	
forests	trees	**non-tight**	**tight**	**non-tight**	**tight**	**non-tight**	**tight**	**non-tight**	**tight**
cardinality		800.00(5)	760.01(6)	**683.64**(8)	759.15(7)	823.12(5)	—(0)	**815.50**(5)	920.96(2)
bidirectional		**760.48**(6)	764.03(7)	800.01(5)	810.95(5)	**761.21**(6)	807.92(6)	829.11(5)	807.74(6)
linear		**762.12**(6)	780.55(6)	800.00(5)	796.08(6)	**766.78**(6)	820.36(5)	804.10(5)	815.98(6)
tournament		760.00(6)	847.32(5)	**699.04**(8)	920.02(2)	763.37(6)	902.37(3)	**720.01**(7)	879.30(4)
				cardinality/ub		802.87(5)	860.91(6)	**800.02**(5)	920.14(2)
				bidirectional/ub		**770.85**(6)	840.26(6)	788.37(6)	789.70(7)
				linear/ub		763.17(6)	813.14(5)	**760.02**(6)	793.65(6)
				tournament/ub		802.80(5)	846.59(5)	**761.63**(6)	790.82(6)

Table 3: Average runtimes for undirected forests and trees

the bidirectional and linear traversal approaches to indicate leaves along with connectedness checking, non-tight encoding variants turn out to be advantageous. Somewhat surprisingly, requiring the existence of $|V| - 1$ edges of a tree $\langle V, E\rangle$ is competitive to checking whether a forest is connected, even though the latter yields the best performance overall in combination with tournament traversal for handling leaves. The addition of a redundant cardinality constraint "*ub*" asserting $|V| - 1$ as an upper bound does not consistently improve the performance of CLASP on edge counting variants. In a similar fashion, hybrid variants incorporating cardinality constraints asserting the existence of $|V| - 1$ edges as well as an encoding part for checking connectedness in some cases yield performance gains in the lower right quarter of Table 3. However, crossing the non-tight connectedness encoding based on tournament traversal, which performs best overall, with an orthogonal condition turns out to be counterproductive.

Furthermore, we ran CLASP on the chordality test encoded in Figure 10 as well a tight variant thereof. Given that chordal graphs admit cycles, combinations with encoding parts building on acyclicity, in particular, choice rules relying on the absence of cycles through a vertex, are not applicable in the context of chordal graphs. However, since any undirected forest is chordal, the number of admissible outcomes subject to XOR-constraints increases, which facilitates the search for a solution. In fact, on the non-tight encoding of chordality, CLASP completed 18 runs with an average runtime of 373.97 seconds, and 14 tight instances were solved in 461.43 seconds on average.

4.2 Reachability Benchmarks

As second benchmark problem, we consider finding Hamiltonian cycles for directed as well as undirected planar graphs with 100 or 150 vertices. For either size, we generated 25 instances using the PLANAR tool,[7] which produces a symmetric edge representation including both $\langle u, v\rangle$ and $\langle v, u\rangle$ for neighboring vertices u and v. Directed Hamiltonian cycles can be expressed in terms of bottom-up or top-down traversals as in Figure 3 and 4 by providing `order(`u`)` as a fact for an arbitrary starting vertex u along with requiring the selection of exactly one predecessor and successor per vertex. In the undirected case, pairs $\langle u, v\rangle$ and $\langle v, u\rangle$ are combined into undirected edges $\{u, v\}$, thus

[7] `http://research.ics.aalto.fi/software/asp/misc/`

directed Hamiltonian cycles	leaf encoding		root encoding		transitivity encoding	
	non-tight	**tight**	**non-tight**	**tight**	**non-tight**	**tight**
CLASP	**0.69** (25)	123.53 (25)	0.76 (25)	131.65 (25)	922.74 (4)	39.08 (25)
CLASP/ACYC	0.58 (25)	105.39 (25)	**0.26** (25)	106.89 (25)	374.35 (22)	19.62 (25)
CLASP/SAT	311.58 (25)	17.00 (25)	333.68 (24)	16.17 (25)	219.90 (25)	**2.68** (25)
LINGELING	896.03 (5)	537.72 (18)	802.65 (10)	436.57 (22)	— (0)	**8.90** (25)
Z3	74.64 (25)	— (0)	**66.63** (25)	— (0)	704.92 (11)	74.70 (25)
CPLEX	**762.50** (6)	— (0)	841.34 (4)	— (0)	— (0)	978.70 (3)
CLASP	**16.06** (25)	987.73 (1)	47.95 (24)	— (0)	— (0)	879.85 (5)
CLASP/ACYC	10.67 (25)	965.05 (2)	**2.81** (25)	— (0)	974.28 (2)	721.07 (8)
CLASP/SAT	— (0)	905.88 (3)	— (0)	966.00 (3)	— (0)	**505.76** (18)
LINGELING	— (0)	— (0)	— (0)	— (0)	— (0)	**756.98** (8)
Z3	113.74 (24)	— (0)	**72.79** (25)	— (0)	— (0)	901.94 (6)
CPLEX	— (0)	— (0)	— (0)	— (0)	— (0)	— (0)

Table 4: Average runtimes for directed Hamiltonian cycles through planar graphs

dividing the number of edge candidates that can be included in a Hamiltonian cycle by two, while asserting the condition that two neighbors are selected per vertex by means of the choice rule in Figure 2 or variants thereof. Although there are fewer undirected than directed edge candidates, our encoding in the latter case disambiguates the predecessor and successor selection for the starting vertex u, so that symmetric directed graphs as well as their undirected counterparts admit the same number of Hamiltonian cycles. This allows us to below assess the impact of switching between a directed and an undirected graph representation. As before, runs were limited to 1000 seconds for finding one directed or undirected Hamiltonian cycle, where all instances are satisfiable.

Table 4 provides average runtimes on directed Hamiltonian cycle encodings for instances with 100 vertices in the upper part and those with 150 vertices in the lower part, investigating the same encoding approaches and solvers as in Table 1. As also observed on the synthetic benchmarks above, the non-tight encoding based on transitive closure is the worst option, leading to only timeouts on instances with 150 vertices apart from two runs completed by CLASP with acyclicity checking. Its tight variant, however, dominates corresponding tight formulations of bottom-up or top-down traversals described by the leaf and root encoding variants, which yield the same (quadratic) size complexity. The linear non-tight formulations of the latter, however, still exhibit performance improvements of CLASP (as ASP solver, using either standard well-foundedness or acyclicity checking), Z3, and CPLEX, whose input languages support compact representations of well-founded derivations. Differences between the leaf and root encoding variants appear to be specific to solvers (in their default settings); e.g., the two versions of CLASP as ASP solver behave complementary, and the translation-based approach relying on acyclicity checking further turns out to be highly competitive. The general advantages due to linear non-tight encodings become apparent when comparing the runtimes of CLASP versions and Z3 on instances with 150 vertices in the lower part, where solvers have difficulties on less compact ASP formulations or SAT translations, respectively.

undirected Hamiltonian cycles	unconditional non-tight	unconditional tight	conditional non-tight	conditional tight
cardinality	0.20 (25)	4.27 (25)	**0.19** (25)	5.87 (25)
bidirectional	0.75 (25)	— (0)	**0.64** (25)	— (0)
linear	0.67 (25)	— (0)	**0.56** (25)	— (0)
tournament	0.36 (25)	17.45 (25)	**0.34** (25)	16.86 (25)
cardinality	0.96 (25)	40.02 (25)	**0.41** (25)	76.09 (25)
bidirectional	5.15 (25)	— (0)	**4.56** (25)	— (0)
linear	2.99 (25)	— (0)	**2.78** (25)	— (0)
tournament	2.52 (25)	237.19 (25)	**1.52** (25)	219.35 (25)

Table 5: Average runtimes for undirected Hamiltonian cycles through planar graphs

Turning to undirected Hamiltonian cycles, Table 5 shows average runtimes of CLASP using the encoding variants also investigated for undirected forests on the left-hand side of Table 3, again for instances with 100 vertices in the upper part and those with 150 vertices in the lower part. As in the directed case, linear non-tight ASP formulations yield more robust performance than their unfolded tight counterparts. Conditional edge generation, progressing successively from leaves or a given starting vertex, respectively, appears to be helpful on non-tight encoding variants, whereas there is no clear trend on tight ones. Regarding different approaches to indicate leaves, the use of cardinality constraints turns out to be advantageous, especially on tight instances. Interestingly, the normalization based on tournament traversal of potential neighbors performs significantly better than the bidirectional and linear traversal approaches, whose tight variants lead to timeouts only. Most importantly, however, the undirected graph representation abolishes redundancy by disregarding edge orientation, given that directed edges of the planar graphs at hand are symmetric. This yields substantial performance improvements in comparison to the average runtimes of CLASP in Table 4 and highlights the utility of a canonical edge representation for avoiding (rather than breaking) symmetry.

4.3 Application Benchmarks

Our third series of experiments deals with the application problem of Bayesian network structure learning [15, 28, 16], which can be expressed in terms of optimization relative to ASP, SAT, or LP formulations. Given a set V of vertices, directed edge candidates C, and scores $s(v, P)$ for vertices $v \in V$ and parent sets $P \subseteq \{u \mid \langle u, v\rangle \in C\}$, the task consists of finding a directed acyclic graph $\langle V, E\rangle$ such that $E \subseteq C$ and $\sum_{v \in V, P=\{u \mid \langle u,v\rangle \in E\}} s(v, P)$ is maximal. Notably, the linear non-tight ASP formulations presented here are more compact than SAT or LP encodings suggested in the literature, which yield from quadratic to sometimes even exponential space complexity.

Among the considered solvers, the two versions of CLASP as ASP solver and CPLEX support optimization, and Table 6 shows their runtimes on eight Bayesian network structure learning instances along with compact non-tight and unfolded tight formulations of bottom-up or top-down acyclicity tests described by the leaf and root encoding

Bayesian net	asia-1	asia-10	hail	insure	mildew-1	mildew-10	water-1	water-10	
CLASP	**0.01**	**0.05**	**885.92**	**6.95**	0.29	**0.95**	10.26	**126.31**	**leaf encoding non-tight**
CLASP/ACYC	**0.01**	0.09	**871.90**	20.21	**0.15**	0.94	**8.31**	88.36	
CPLEX	1.04	**3.42**	**1.01**	81.15	**10.55**	0.39	569.07	—	
CLASP	0.02	0.11	2871.79	42.96	1.77	3.09	31.83	290.37	**leaf encoding tight**
CLASP/ACYC	0.02	0.08	1484.74	23.05	0.76	1.63	15.48	143.90	
CPLEX	10.28	23.49	—	—	—	3098.88	—	—	
CLASP	0.04	0.30	1737.08	19.58	**0.27**	0.96	**9.19**	167.64	**root encoding non-tight**
CLASP/ACYC	**0.01**	**0.05**	966.34	**7.84**	0.18	**0.76**	9.93	**70.16**	
CPLEX	**0.56**	3.54	1.45	**55.22**	35.60	**0.35**	**55.36**	**2922.53**	
CLASP	0.02	0.13	—	43.90	1.28	4.04	71.23	748.33	**root encoding tight**
CLASP/ACYC	0.03	0.09	—	34.86	1.10	3.78	51.25	568.77	
CPLEX	19.61	77.28	—	—	—	1511.86	—	—	

Table 6: Runtimes for Bayesian network structure learning

variants. In view of the increased hardness of performing optimization, we extended the time limit to 3600 seconds per run. The shortest runtime of each solver per instance in a column is highlighted in boldface. For all solvers, we observe significant advantages on non-tight encoding variants. Unlike with the synthetic benchmarks investigated in Section 4.1, CPLEX here benefits from the compact representation of level mappings in terms of numerical variables introduced by the translator ACYC2SOLVER. While CPLEX tends to perform better on the root than on the leaf encoding variants, the opposite applies to the standard version of CLASP, and the prototype with acyclicity checking shows no clear tendency on non-tight instances. These differences are application- and solver-specific in view of the symmetry of both approaches to test acyclicity. Although ASP formulations are automatically translated to LP and the two versions of the native ASP solver CLASP perform better in most cases, CPLEX has a clear advantage on the "*hail*" instance. This points out the potential utility of translational approaches and corresponding solver technology to solve ASP formulations.

5 Conclusions

Graphs satisfying acyclicity properties are frequent in knowledge representation tasks. As these properties do not appear as basic primitives in common constraint-based representation formalisms, formulating them compactly and efficiently is a recurring challenge. We investigated logic-based characterizations of acyclicity conditions and developed a systematic collection of encodings in the language of answer set programming. Corresponding encodings can also be expressed in related formalisms like SAT, DL, and LP, for which off-the-shelf tools provide automatic translations from ASP. Our experiments showed that linear non-tight formulations of acyclicity properties tend to be advantageous in formalisms supporting compact representations, as obtained through well-founded derivations in ASP, acyclicity checking on top of SAT (or ASP), difference constraints in DL, or numerical variables in LP. In the case of plain SAT, however, tight encodings based on transitive closure turned out to be the first choice. While such

an approach performs well for graphs of moderate size, our experiments also indicated that it does not scale anymore when the number of vertices goes into the hundreds. A second drawback is that transitivity relies on edge orientations and does not carry over to the undirected case in a natural way.

Interestingly, quadratic encodings based on the full transitive closure of edges appear frequently in the literature, e.g., for the purpose of Bayesian network structure learning [15]. In particular, several ASP-based approaches [7, 10, 11, 6] utilize respective non-tight formulations similar to the encoding in Figure 5. As this option was by far the worst in our experiments, future applications should rather take advantage of linear bottom-up or top-down acyclicity tests, as given in Figure 3, or switch to a tight formulation when transitive closure is not a space bottleneck. Furthermore, exponential encodings of acyclicity or chordality, respectively, have been proposed in the context of LP and SAT [28, 16, 14], whereas ours are proportional to graph size. By using compact non-tight formulations like those in Figure 3 and 10 instead, we achieved significant speed-ups in applications of our own that rely on acyclicity [16], chordality [14], or (taxonomy) trees [27], both when using ASP or translational approaches to perform optimization relative to SAT or LP, respectively. Hence, we believe that other applications dealing with acyclic or tree structures may benefit as well by incorporating encoding approaches presented here.

The encoding variants for mutual exclusion testing in Figure 7 resemble a variety of approaches from SAT to formulate cardinality constraints [39, 32, 34, 20]. Such conditions are particularly relevant for expressing acyclicity in the undirected case, which turned out to be an effective way to cancel symmetric edge orientations (cf. Section 4.2). In contrast, the SAT-based method for solving Hamiltonian cycle problems proposed in [40] uses a symmetric representation of undirected edges "for simple modeling". Similarly, edge orientations are a prerequisite for utilizing the recent SAT modulo graphs framework [24] to perform acyclicity tests. Such native support in principle abolishes the need of encoding acyclicity conditions explicitly, but it has also turned out to be highly competitive when applied to translations [21]. In view of reduced combinatorics, the experiments in Section 4.2 still exhibited more robust performance when disregarding edge orientation and using dedicated encodings for the undirected case. Such observations suggest that a native handling of undirected graphs based on an asymmetric edge representation may improve the search performance even further.

In summary, there are various options for formulating and combining encoding parts addressing acyclicity and tree properties in constraint-based representation formalisms. Experiments on combinatorial as well as application benchmarks indicate the relevance of representation approaches and constraint formulations, with particular emphasis on well-foundedness and cardinality constraints. While compact formulations are beneficial in most cases, the resulting computational performance is also affected by the specific formalism and solver under consideration. For instance, making the full transitive closure of edges explicit may be worthwhile in plain SAT, especially for graphs of moderate size, whereas compactly expressed fixpoint constructions relying on well-foundedness scale better otherwise. In view of such phenomena, diverse declarative means to represent acyclic or tree structures along with systematic empirical investigation are essential elements in the design of robust constraint-based solving methods.

References

1. Apt, K., Blair, H., Walker, A.: Towards a theory of declarative knowledge. In Minker, J., ed.: Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann Publishers (1988) 89–148
2. Biere, A., Heule, M., van Maaren, H., Walsh, T., eds.: Handbook of Satisfiability. Volume 185 of Frontiers in Artificial Intelligence and Applications. IOS Press (2009)
3. Bomanson, J., Gebser, M., Janhunen, T., Kaufmann, B., Schaub, T.: Answer set programming modulo acyclicity. In Calimeri, F., Ianni, G., Truszczyński, M., eds.: Proceedings of the Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'15). Volume 9345 of Lecture Notes in Artificial Intelligence. Springer (2015)
4. Bomanson, J., Janhunen, T.: Normalizing cardinality rules using merging and sorting constructions. In Cabalar, P., Son, T., eds.: Proceedings of the Twelfth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'13). Volume 8148 of Lecture Notes in Artificial Intelligence. Springer (2013) 187–199
5. Bonet, M., John, K.: Efficiently calculating evolutionary tree measures using SAT. In Kullmann, O., ed.: Proceedings of the Twelfth International Conference on Theory and Applications of Satisfiability Testing (SAT'09). Volume 5584 of Lecture Notes in Computer Science. Springer (2009) 4–17
6. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Communications of the ACM **54**(12) (2011) 92–103
7. Brooks, D., Erdem, E., Erdogan, S., Minett, J., Ringe, D.: Inferring phylogenetic trees using answer set programming. Journal of Automated Reasoning **39**(4) (2007) 471–511
8. Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F., Schaub, T.: ASP-Core-2: Input language format. (2012) Available at `https://www.mat.unical.it/aspcomp2013/ASPStandardization/`
9. Calimeri, F., Gebser, M., Maratea, M., Ricca, F.: The design of the fifth answer set programming competition. In Leuschel, M., Schrijvers, T., eds.: Technical Communications of the Thirtieth International Conference on Logic Programming (ICLP'14). (2014) Available at `http://arxiv.org/abs/1405.3710v4`
10. Çaylı, M., Karatop, A., Kavlak, E., Kaynar, H., Türe, F., Erdem, E.: Solving challenging grid puzzles with answer set programming. In Costantini, S., Watson, R., eds.: Proceedings of the Fourth International Workshop on Answer Set Programming (ASP'07). (2007) 175–190
11. Çelik, M., Erdoğan, H., Tahaoğlu, F., Uras, T., Erdem, E.: Comparing ASP and CP on four grid puzzles. In Gavanelli, M., Mancini, T., eds.: Proceedings of the Sixteenth RCRA International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (RCRA'09). CEUR Workshop Proceedings (2009)
12. Chandran, L., Ibarra, L., Ruskey, F., Sawada, J.: Enumerating and characterizing the perfect elimination orderings of a chordal graph. Theoretical Computer Science **307**(2) (2003) 303–317
13. Clark, K.: Negation as failure. In Gallaire, H., Minker, J., eds.: Logic and Data Bases. Plenum Press (1978) 293–322
14. 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., eds.: Proceedings of the Twenty-seventh Annual Conference on Neural Information Processing Systems (NIPS'13). Volume 26 of Advances in Neural Information Processing Systems. Neural Information Processing Systems Foundation (2013) 1349–1357
15. Cussens, J.: Bayesian network learning by compiling to weighted MAX-SAT. In McAllester, D., Myllymäki, P., eds.: Proceedings of the Twenty-fourth International Conference on Uncertainty in Artificial Intelligence (UAI'08). AUAI Press (2008) 105–112

16. Cussens, J.: Bayesian network learning with cutting planes. In Cozman, F., Pfeffer, A., eds.: Proceedings of the Twenty-seventh International Conference on Uncertainty in Artificial Intelligence (UAI'11). AUAI Press (2011) 153–160
17. Dantzig, G.: Linear Programming and Extensions. Princeton Landmarks in Mathematics and Physics. Princeton University Press (1963)
18. Diestel, R.: Graph Theory. Volume 173 of Graduate Texts in Mathematics. Springer (2010)
19. Erdem, E., Lifschitz, V.: Tight logic programs. Theory and Practice of Logic Programming **3**(4-5) (2003) 499–518
20. Frisch, A., Giannoros, P.: SAT encodings of the at-most-k constraint. In: Proceedings of the Ninth International Workshop on Constraint Modelling and Reformulation (ModRef'10). (2010)
21. Gebser, M., Janhunen, T., Rintanen, J.: Answer set programming as SAT modulo acyclicity. In Schaub, T., Friedrich, G., O'Sullivan, B., eds.: Proceedings of the Twenty-first European Conference on Artificial Intelligence (ECAI'14). IOS Press (2014) 351–356
22. Gebser, M., Janhunen, T., Rintanen, J.: ASP encodings of acyclicity properties. In Baral, C., De Giacomo, G., Eiter, T., eds.: Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'14). AAAI Press (2014)
23. Gebser, M., Janhunen, T., Rintanen, J.: Declarative encodings of acyclicity properties. In Inclezan, D., Maratea, M., eds.: Proceedings of the Seventh Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'14). (2014) Available at `https://sites.google.com/site/aspocp2014/`
24. Gebser, M., Janhunen, T., Rintanen, J.: SAT modulo graphs: Acyclicity. In Fermé, E., Leite, J., eds.: Proceedings of the Fourteenth European Conference on Logics in Artificial Intelligence (JELIA'14). Volume 8761 of Lecture Notes in Artificial Intelligence. Springer (2014) 137–151
25. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Thiele, S.: A user's guide to `gringo`, `clasp`, `clingo`, and `iclingo`. (2010) Available at `http://sourceforge.net/projects/potassco/files/potassco_guide/`
26. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers (2012)
27. Heikinheimo, H., Eronen, J., Sennikov, A., Preston, C., Oikarinen, E., Uotila, P., Mannila, H., Fortelius, M.: Convergence in the distribution patterns of Europe's plants and mammals is due to environmental forcing. Journal of Biogeography **39**(9) (2012) 1633–1644
28. Jaakkola, T., Sontag, D., Globerson, A., Meila, M.: Learning Bayesian network structure using LP relaxations. In Teh, Y., Titterington, D., eds.: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS'10). Volume 9 of JMLR Proceedings. JMLR.org (2010) 358–365
29. Janhunen, T.: Representing normal programs with clauses. In López de Mántaras, R., Saitta, L., eds.: Proceedings of the Sixteenth European Conference on Artificial Intelligence (ECAI'04). IOS Press (2004) 358–362
30. Janhunen, T., Niemelä, I.: Compact translations of non-disjunctive answer set programs to propositional clauses. In Balduccini, M., Son, T., eds.: Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday. Volume 6565 of Lecture Notes in Computer Science. Springer (2011) 111–130
31. Janhunen, T., Niemelä, I., Sevalnev, M.: Computing stable models via reductions to difference logic. In Erdem, E., Lin, F., Schaub, T., eds.: Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09). Volume 5753 of Lecture Notes in Artificial Intelligence. Springer (2009) 142–154

32. Klieber, W., Kwon, G.: Efficient CNF encoding for selecting 1 from n objects. In Velev, M., ed.: Proceedings of the Fourth International Workshop on Constraints in Formal Verification (CFV'07). (2007)
33. Liu, G., Janhunen, T., Niemelä, I.: Answer set programming via mixed integer programming. In Brewka, G., Eiter, T., McIlraith, S., eds.: Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'12). AAAI Press (2012) 32–42
34. Marques-Silva, J., Lynce, I.: Towards robust CNF encodings of cardinality constraints. In Bessiere, C., ed.: Proceedings of the Thirteenth International Conference on Principles and Practice of Constraint Programming (CP'07). Volume 4741 of Lecture Notes in Computer Science. Springer (2007) 483–497
35. 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., eds.: 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). Volume 7773 of Lecture Notes in Computer Science. Springer (2013) 105–116
36. Niemelä, I.: Stable models and difference logic. Annals of Mathematics and Artificial Intelligence **53**(1-4) (2008) 313–329
37. Nieuwenhuis, R., Oliveras, A.: DPLL(T) with exhaustive theory propagation and its application to difference logic. In Etessami, K., Rajamani, S., eds.: Proceedings of the Seventeenth International Conference on Computer Aided Verification (CAV'05). Volume 3576 of Lecture Notes in Computer Science. Springer (2005) 321–334
38. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138**(1-2) (2002) 181–234
39. Sinz, C.: Towards an optimal CNF encoding of Boolean cardinality constraints. In van Beek, P., ed.: Proceedings of the Eleventh International Conference on Principles and Practice of Constraint Programming (CP'05). Volume 3709 of Lecture Notes in Computer Science. Springer (2005) 827–831
40. 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., eds.: Proceedings of the Fourteenth European Conference on Logics in Artificial Intelligence (JELIA'14). Volume 8761 of Lecture Notes in Artificial Intelligence. Springer (2014) 684–693
41. Syrjänen, T.: Lparse 1.0 user's manual. (2001) Available at `http://www.tcs.hut.fi/Software/smodels/`

A Correctness Proofs

In what follows, we show the correctness of ground instances of the main first-order encodings given in Section 3; other variants and combinations can be proven analogously. A first-order encoding, possibly including arithmetic expressions, is a shorthand for all ground instances of rules obtainable by substituting ground terms for variables and evaluating arithmetic expressions in the standard way. For details on ground instantiation, we refer the interested reader to [8, 25, 41].

To begin with, we recall the formal syntax and semantics of ground logic programs with choice rules as well as cardinality and integrity constraints [38]. A *rule* r is an expression of the form

$$h \leftarrow a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n$$

where a_i, for $1 \leq i \leq n$, is an *atom* or a *cardinality constraint* of the form $l\,\{a_1, \ldots, a_j, \text{not } a_{j+1}, \ldots, \text{not } a_k\}\,u$ in which l, u are integers and $a_1, \ldots, a_k$ are atoms; either or both of l and u can be omitted if $l = 0$ or $u = k$, respectively. The head h of r is either an atom, the special symbol $\bot$, or a *choice* $\{a\}$ in which a is an atom; note that a choice is a shorthand for the cardinality constraint $0\,\{a\}\,1$. We denote the set of atoms occurring in h by $\mathrm{H}(h)$, i.e., $\mathrm{H}(h) = \{h\}$ if h is an atom, $\mathrm{H}(h) = \{a\}$ if $h = \{a\}$, and $\mathrm{H}(h) = \emptyset$ if $h = \bot$. A logic *program* R is a finite set of rules. The *positive dependency graph* of R is the directed graph consisting of atoms in R as vertices and edges $\langle a, a' \rangle$ for rules $r \in R$ such that $a \in \mathrm{H}(h)$ and, for $1 \leq i \leq m$, either $a_i = a'$ or $a_i = l\,\{a_1, \ldots, a_j, \text{not } a_{j+1}, \ldots, \text{not } a_k\}\,u$ and $a' \in \{a_1, \ldots, a_j\}$; R is *tight* [19] if its positive dependency graph is acyclic, and *non-tight* otherwise. Given a set X of atoms, the satisfaction relation $\models$ is inductively defined as follows:

- $X \models a$, if $a \in X$;
- $X \models l\,\{a_1, \ldots, a_j, \text{not } a_{j+1}, \ldots, \text{not } a_k\}\,u$, if $l \leq |\{a_1, \ldots, a_j\} \cap X| + |\{a_{j+1}, \ldots, a_k\} \setminus X| \leq u$;
- $X \models h \leftarrow a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n$, if $X \models a_1, \ldots, X \models a_m$ and $X \not\models a_{m+1}, \ldots, X \not\models a_n$ imply $X \models h$;
- $X \models R$, if $X \models r$ for all $r \in R$.

If $X \models R$, then X is a *model* of R. The *reduct* R^X of R relative to X is inductively defined as follows:

- $a^X = a$, for an atom a;
- $[l\,\{a_1, \ldots, a_j, \text{not } a_{j+1}, \ldots, \text{not } a_k\}\,u]^X = l'\{a_1, \ldots, a_j\}$, for $l' = l - |\{a_{j+1}, \ldots, a_k\} \setminus X|$;
- $R^X = \{a \leftarrow a_1^X, \ldots, a_m^X \mid [h \leftarrow a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n] \in R, a \in \mathrm{H}(h) \cap X, X \not\models a_{m+1}, \ldots, X \not\models a_n\}$.

Note that R^X has a unique least model, denoted by $\mathrm{LM}(R^X)$. Then, X is a *stable model* of R if $X \models R$ and $\mathrm{LM}(R^X) = X$. Deciding whether R has some stable model is NP-complete, given that $\mathrm{LM}(R^X)$ can be computed in linear time [38].

We below use the following abbreviations for ground instances of encoding parts in Section 3 obtained with $\mathtt{n}{=}n$ for some integer n:

G^n: Ground instances of the rules in Figure 2.
A^n: Ground instances of the rules in Figure 3.
F_a^n: Ground instances of the rules in Figure 7(a).
F_b^n: Ground instances of the rules in Figure 7(b).
F_c^n: Ground instances of the rules in Figure 7(c).
F_d^n: Ground instances of the rules in Figure 7(d).
F_e^n: Ground instances of the rules in Figure 7(e).
T_a^n: Ground instances of the rules in Figure 8(a).
T_b^n: Ground instances of the rules in Figure 8(b).
T_c^n: Ground instances of the rules in Figure 8(c).
G_u^n: Ground instances of the rules in Figure 2 with line 3 replaced by

```
pair(X,Y) :- node(X;Y), X < Y.
```

U^n: Ground instances of the rules in Figure 9.
C^n: Ground instances of the rules in Figure 10.

We proceed by showing the correctness of a selection of encodings of directed graphs that are acyclic, forests, or trees, respectively.

Proposition 1. *Given an integer n, let $V = \{1, \ldots, n\}$ and $R = G^n \cup A^n$. Then, we have that:*

1. *If $\langle V, E\rangle$ is a directed acyclic graph, then there is exactly one stable model X of R such that $\{\langle u, v\rangle \mid edge(u, v) \in X\} = E$.*
2. *If X is a stable model of R, then $\langle V, \{\langle u, v\rangle \mid edge(u, v) \in X\}\rangle$ is a directed acyclic graph.*

Proof. In view of line 2 and 3 in Figure 2, any stable model X of R must be such that $\{v \mid node(v) \in X\} = V$ and $\{\langle u, v\rangle \mid pair(u, v) \in X\} = (V \times V) \setminus \{\langle w, w\rangle \mid w \in V\}$. Along with line 7 and 9 in Figure 3, it must also hold that $\{v \mid order(v) \in X\} = V$ and $\{\langle u, v\rangle \mid order(u, v) \in X\} = (V \times V) \setminus \{\langle w, w\rangle \mid w \in V\}$, where establishing that $\{order(v) \mid v \in V\} \cup \{order(u, v) \mid \langle u, v\rangle \in (V \times V) \setminus \{\langle w, w\rangle \mid w \in V\}\} \subseteq \mathrm{LM}(R^X)$ is of interest. We further consider the items of the statement:

1. If $\langle V, E\rangle$ is a directed acyclic graph, let $X = \{node(v), order(v) \mid v \in V\} \cup \{pair(u, v), order(u, v) \mid \langle u, v\rangle \in (V \times V) \setminus \{\langle w, w\rangle \mid w \in V\}\} \cup \{edge(u, v) \mid \langle u, v\rangle \in E\}$, where the choice rule in line 4 of Figure 2 grants that $\{\langle u, v\rangle \mid edge(u, v) \in X\} = \{\langle u, v\rangle \mid edge(u, v) \in \mathrm{LM}(R^X)\} = E$. Given the above considerations, there cannot be any stable model $X' \neq X$ of R such that $\{\langle u, v\rangle \mid edge(u, v) \in X'\} = E$. Moreover, since $\langle V, E\rangle$ is acyclic, for any non-empty $U \subseteq V$, there is some vertex $u \in U$ such that $\{v \mid \langle u, v\rangle \in E\} \subseteq V \setminus U$, i.e., $\{edge(u, v) \in X \mid v \in U\} = \emptyset$. In view of line 5 to 7 in Figure 3, it follows by induction that $\{order(v) \mid v \in V\} \cup \{order(u, v) \mid \langle u, v\rangle \in (V \times V) \setminus \{\langle w, w\rangle \mid w \in V\}\} \subseteq \mathrm{LM}(R^X)$, so that $X = \mathrm{LM}(R^X)$ is a stable model of R.
2. If X is a stable model of R, then $X = \mathrm{LM}(R^X)$ implies that, for any non-empty $U \subseteq V$, there is some vertex $u \in U$ such that $\{edge(u, v) \in X \mid v \in U\} = \emptyset$, so that $\langle V, \{\langle u, v\rangle \mid edge(u, v) \in X\}\rangle$ is a directed acyclic graph. □

Proposition 2. *Given an integer n, let $V = \{1, \ldots, n\}$ and $R_f = G^n \cup A^n \cup F_f^n$ for $f \in \{a, b, c, d, e\}$. Then, we have that:*

1. *If $\langle V, E\rangle$ is a directed forest, then there is exactly one stable model X of R_f such that $\{\langle u, v\rangle \mid edge(u, v) \in X\} = E$.*
2. *If X is a stable model of R_f, then $\langle V, \{\langle u, v\rangle \mid edge(u, v) \in X\}\rangle$ is a directed forest.*

Proof. Since no head atom (if any) of F_f^n, for $f \in \{a, b, c, d, e\}$, occurs in $G^n \cup A^n$, any stable model X of R_f has the form $X = Y \cup Z$ for some stable model Y of $G^n \cup A^n$ such that $\{node(v), order(v) \mid v \in V\} \cup \{pair(u, v), order(u, v) \mid \langle u, v\rangle \in (V \times V) \setminus \{\langle w, w\rangle \mid w \in V\}\} \subseteq Y \subseteq \{node(v), order(v) \mid v \in V\} \cup \{pair(u, v), order(u, v), edge(u, v) \mid \langle u, v\rangle \in (V \times V) \setminus \{\langle w, w\rangle \mid w \in V\}\}$ and $Z \subseteq \bigcup_{[h \leftarrow a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n] \in F_f^n} \mathrm{H}(h)$. By Proposition 1, we have that Y uniquely represents the directed acyclic graph $G = \langle V, \{\langle u, v\rangle \mid edge(u, v) \in Y\}\rangle$. Below we consider particular cases for $f \in \{a, b, c, d, e\}$ and show that G is a directed forest iff X is the single stable model of R_f such that $X \setminus Z = Y$:

a. In view of line 10 in Figure 7(a), $X = Y$ is a stable model of R_a iff, for any $v \in V$, $|\{\langle u, v\rangle \mid edge(u, v) \in X\}| \leq 1$ iff G is a directed forest.

b. In view of line 10 in Figure 7(b), $X = Y$ is a stable model of R_b iff, for any $v \in V$, $|\{\langle u, v\rangle \mid edge(u, v) \in X\}| \leq 1$ iff G is a directed forest.

c. In view of line 14 in Figure 7(c), it must hold that $\{v \mid unique(v) \in Z\} = V$. Along with line 13, for any $v \in V$, there must be some $u \in V$ such that $\{unique(v, u, 1), unique(v, u, -1)\} \subseteq Z$, where $u \neq v$ if $n > 1$. On the other hand, the rules in line 10 to 12 yield that $\{u \mid unique(v, u, 1) \in \mathrm{LM}(R_c^X)\} = \{u \in V \mid \{w \mid edge(w, v) \in Y\} \subseteq \{u, \ldots, n\}\}$ and $\{u \mid unique(v, u, -1) \in \mathrm{LM}(R_c^X)\} = \{u \in V \mid \{w \mid edge(w, v) \in Y\} \subseteq \{1, \ldots, u\}\}$, so that $\{u \mid unique(v, u, 1) \in \mathrm{LM}(R_c^X), unique(v, u, -1) \in \mathrm{LM}(R_c^X)\} = \{u \in V \mid \{w \mid edge(w, v) \in Y\} \subseteq \{u\}\}$. It must thus hold that $\{unique(v) \mid v \in V\} \subseteq Z = \mathrm{LM}(R_c^X) \cap \bigcup_{[h \leftarrow a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n] \in F_c^n} \mathrm{H}(h) = \{unique(v) \mid v \in V, |\{u \mid edge(u, v) \in Y\}| \leq 1\} \cup \{unique(v, u, 1) \mid v, u \in V, \{w \mid edge(w, v) \in Y\} \subseteq \{u, \ldots, n\}\} \cup \{unique(v, u, -1) \mid v, u \in V, \{w \mid edge(w, v) \in Y\} \subseteq \{1, \ldots, u\}\}$. That is, $X = Y \cup Z$ is a stable model of R_c iff, for any $v \in V$, $|\{\langle u, v\rangle \mid edge(u, v) \in X\}| \leq 1$ iff G is a directed forest.

d. In view of line 14 in Figure 7(d), it must hold that $\{v \mid unique(v, 1, 0) \in Z\} = V$. For any $v \in V$, the rules in line 10 to 13 yield that $\{u \mid unique(v, u, 1) \in \mathrm{LM}(R_d^X)\} = \{u \in V \mid |\{w \geq u \mid edge(w, v) \in Y\}| = 0\}$ and $\{u \mid unique(v, u, 0) \in \mathrm{LM}(R_d^X)\} = \{u \in V \mid |\{w \geq u \mid edge(w, v) \in Y\}| \leq 1\}$. It must thus hold that $\{unique(v, 1, 0) \mid v \in V\} \subseteq Z = \mathrm{LM}(R_d^X) \cap \bigcup_{[h \leftarrow a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n] \in F_d^n} \mathrm{H}(h) = \{unique(v, u, 1) \mid v, u \in V, |\{w \geq u \mid edge(w, v) \in Y\}| = 0\} \cup \{unique(v, u, 0) \mid v, u \in V, |\{w \geq u \mid edge(w, v) \in Y\}| \leq 1\}$. That is, $X = Y \cup Z$ is a stable model of R_d iff, for any $v \in V$, $|\{\langle u, v\rangle \mid edge(u, v) \in X\}| \leq 1$ iff G is a directed forest.

e. In view of line 14 in Figure 7(e), it must hold that $\{v \mid unique(v, 1, 1, 0) \in Z\} = V$. For any $v \in V$, the rules in line 10 to 13 yield that $\{\langle i, j\rangle \mid unique(v, i, j, 1) \in$

$\mathrm{LM}(R_e^X)\} = \{\langle i,j\rangle \mid l \in \{0,\ldots,\lceil \log_2(n)\rceil\}, i = \lceil \frac{n}{2^l}\rceil, j \in \{1,\ldots,i\}, |\{2^l{*}(j{-}1) < u \leq 2^l{*}j \mid \mathit{edge}(u,v) \in Y\}| = 0\}$ and $\{\langle i,j\rangle \mid \mathit{unique}(v,i,j,0) \in \mathrm{LM}(R_e^X)\} = \{\langle i,j\rangle \mid l \in \{0,\ldots,\lceil \log_2(n)\rceil\}, i = \lceil \frac{n}{2^l}\rceil, j \in \{1,\ldots,i\}, |\{2^l * (j-1) < u \leq 2^l * j \mid \mathit{edge}(u,v) \in Y\}| \leq 1\}$. In particular, we have that $\mathit{unique}(v,1,1,0) \in \mathrm{LM}(R_e^X)$ iff $|\{0 < u \leq 2^{\lceil \log_2(n)\rceil} \mid \mathit{edge}(u,v) \in Y\}| = |\{u \mid \mathit{edge}(u,v) \in Y\}| \leq 1$. It must thus hold that $\{\mathit{unique}(v,1,1,0) \mid v \in V\} \subseteq Z = \mathrm{LM}(R_e^X) \cap \bigcup_{[h \leftarrow a_1,\ldots,a_m,\mathrm{not}\ a_{m+1},\ldots,\mathrm{not}\ a_n] \in F_e^n} \mathrm{H}(h) = \{\mathit{unique}(v,i,j,1) \mid v \in V, l \in \{0,\ldots,\lceil \log_2(n)\rceil\}, i = \lceil \frac{n}{2^l}\rceil, j \in \{1,\ldots,i\}, |\{2^l{*}(j{-}1) < u \leq 2^l{*}j \mid \mathit{edge}(u,v) \in Y\}| = 0\} \cup \{\mathit{unique}(v,i,j,0) \mid v \in V, l \in \{0,\ldots,\lceil \log_2(n)\rceil\}, i = \lceil \frac{n}{2^l}\rceil, j \in \{1,\ldots,i\}, |\{2^l * (j-1) < u \leq 2^l * j \mid \mathit{edge}(u,v) \in Y\}| \leq 1\}$. That is, $X = Y \cup Z$ is a stable model of R_e iff, for any $v \in V$, $|\{\langle u,v\rangle \mid \mathit{edge}(u,v) \in X\}| \leq 1$ iff G is a directed forest. □

Proposition 3. *Given an integer n, let $V = \{1,\ldots,n\}$ and $R_t = G^n \cup A^n \cup F_f^n \cup T_t^n$ for $f \in \{a,b,c,d,e\}$ and $t \in \{a,b,c\}$. Then, we have that:*

1. *If $\langle V,E\rangle$ is a directed tree, then there is exactly one stable model X of R_t such that $\{\langle u,v\rangle \mid \mathit{edge}(u,v) \in X\} = E$.*
2. *If X is a stable model of R_t, then $\langle V, \{\langle u,v\rangle \mid \mathit{edge}(u,v) \in X\}\rangle$ is a directed tree.*

Proof. Since no head atom (if any) of T_t^n, for $t \in \{a,b,c\}$, occurs in $G^n \cup A^n \cup F_f^n$ for $f \in \{a,b,c,d,e\}$, any stable model X of R_t has the form $X = Y \cup Z$ for some stable model Y of $G^n \cup A^n \cup F_f^n$ such that $\{\mathit{node}(v), \mathit{order}(v) \mid v \in V\} \cup \{\mathit{pair}(u,v), \mathit{order}(u,v) \mid \langle u,v\rangle \in (V \times V) \setminus \{\langle w,w\rangle \mid w \in V\}\} \subseteq Y \subseteq \{\mathit{node}(v), \mathit{order}(v) \mid v \in V\} \cup \{\mathit{pair}(u,v), \mathit{order}(u,v), \mathit{edge}(u,v) \mid \langle u,v\rangle \in (V \times V) \setminus \{\langle w,w\rangle \mid w \in V\}\} \cup \bigcup_{[h \leftarrow a_1,\ldots,a_m,\mathrm{not}\ a_{m+1},\ldots,\mathrm{not}\ a_n] \in F_f^n} \mathrm{H}(h)$ and $Z \subseteq \bigcup_{[h \leftarrow a_1,\ldots,a_m,\mathrm{not}\ a_{m+1},\ldots,\mathrm{not}\ a_n] \in T_t^n} \mathrm{H}(h)$. By Proposition 2, we have that Y uniquely represents the directed forest $G = \langle V, \{\langle u,v\rangle \mid \mathit{edge}(u,v) \in Y\}\rangle$. Below we consider particular cases for $t \in \{a,b,c\}$ and show that G is a directed tree iff X is the single stable model of R_t such that $X \setminus Z = Y$:

a. In view of line 15 in Figure 8(a), $X = Y$ is a stable model of R_a iff $|\{\langle u,v\rangle \mid \mathit{edge}(u,v) \in X\}| \geq |V| - 1$ iff G is a directed tree.

b. In view of line 15 in Figure 8(b), it must hold that $Z = \mathrm{LM}(R_b^X) \cap \bigcup_{[h \leftarrow a_1,\ldots,a_m,\mathrm{not}\ a_{m+1},\ldots,\mathrm{not}\ a_n] \in T_b^n} \mathrm{H}(h) = \{\mathit{child}(v) \mid \mathit{edge}(u,v) \in Y\}$. On the other hand, the integrity constraint in line 16 requires that $|\{v \in V \mid \mathit{child}(v) \notin Z\}| \leq 1$. That is, $X = Y \cup Z$ is a stable model of R_b iff $|\{v \in V \mid \{\langle u,v\rangle \mid \mathit{edge}(u,v) \in X\} = \emptyset\}| \leq 1$ iff G is a directed tree.

c. Let C denote the vertices in the connected component of $\langle V, \{\{u,v\} \mid \mathit{edge}(u,v) \in Y\}\rangle$ that contains vertex 1, or $C = \emptyset$ when $V = \emptyset$. In view of line 15 to 17 in Figure 8(c), it must hold that $Z = \mathrm{LM}(R_c^X) \cap \bigcup_{[h \leftarrow a_1,\ldots,a_m,\mathrm{not}\ a_{m+1},\ldots,\mathrm{not}\ a_n] \in T_c^n} \mathrm{H}(h) = \{\mathit{reach}(v) \mid v \in C\}$. On the other hand, the integrity constraint in line 18 requires that $\{v \in V \mid \mathit{reach}(v) \in Z\} = V$. That is, $X = Y \cup Z$ is a stable model of R_c iff $\langle V, \{\{u,v\} \mid \mathit{edge}(u,v) \in X\}\rangle$ is connected iff G is a directed tree. □

In what follows, we turn to undirected forests and chordal graphs. The considerations regarding undirected trees are similar to the directed case when adding the rules in T_a^n or T_c^n to an underlying encoding of undirected forests.

Proposition 4. *Given an integer n, let $V = \{1, \ldots, n\}$ and $R = G_u^n \cup U^n$. Then, we have that:*

1. *If $\langle V, E\rangle$ is an undirected forest, then there is exactly one stable model X of R such that $\{\{u, v\} \mid edge(u, v) \in X\} = E$.*
2. *If X is a stable model of R, then $\langle V, \{\{u, v\} \mid edge(u, v) \in X\}\rangle$ is an undirected forest.*

Proof. Recall that line 3 of Figure 2 is replaced by the following rule in G_u^n:

```
pair(X,Y) :- node(X;Y), X < Y.
```

Along with line 2 in Figure 2, any stable model X of R must be such that $\{v \mid node(v) \in X\} = V$ and $\{\langle u, v\rangle \mid pair(u, v) \in X\} = \{\langle u, v\rangle \mid u, v \in V, u < v\}$. In view of line 6, 7, and 9 in Figure 9, it must also hold that $\{v \mid order(v) \in X\} = V$ and $\{\langle u, v\rangle \mid order(u, v) \in X\} = \{\langle u, v\rangle \mid u, v \in V, u < v\}$, where establishing that $\{order(v) \mid v \in V\} \cup \{order(u, v) \mid u, v \in V, u < v\} \subseteq \mathrm{LM}(R^X)$ is of interest. We further consider the items of the statement:

1. If $\langle V, E\rangle$ is an undirected forest, let $X = \{node(v), order(v) \mid v \in V\} \cup \{pair(u, v), order(u, v) \mid u, v \in V, u < v\} \cup \{edge(u, v) \mid \{u, v\} \in E, u < v\}$, where the choice rule in line 4 of Figure 2 grants that $\{\langle u, v\rangle \mid edge(u, v) \in X\} = \{\langle u, v\rangle \mid edge(u, v) \in \mathrm{LM}(R^X)\} = \{\langle u, v\rangle \mid \{u, v\} \in E, u < v\}$. Given the above considerations, there cannot be any stable model $X' \neq X$ of R such that $\{\{u, v\} \mid edge(u, v) \in X'\} = E$. Moreover, since $\langle V, E\rangle$ is an undirected forest, for any non-empty $U \subseteq V$, there is some vertex $u \in U$ such that $|\{v \mid \{u, v\} \in E, v \in U\}| \leq 1$, i.e., $|\{edge(u, v) \in X \mid v \in U, u < v\} \cup \{edge(v, u) \in X \mid v \in U, v < u\}| \leq 1$. In view of line 5 to 8 in Figure 9, it follows by induction that $\{order(v) \mid v \in V\} \cup \{order(u, v) \mid u, v \in V, u < v\} \subseteq \mathrm{LM}(R^X)$, so that $X = \mathrm{LM}(R^X)$ is a stable model of R.
2. If X is a stable model of R, then $X = \mathrm{LM}(R^X)$ implies that, for any non-empty $U \subseteq V$, there is some vertex $u \in U$ such that $|\{edge(u, v) \in X \mid v \in U, u < v\} \cup \{edge(v, u) \in X \mid v \in U, v < u\}| \leq 1$, so that $\langle V, \{\{u, v\} \mid edge(u, v) \in X\}\rangle$ is an undirected forest. □

Proposition 5. *Given an integer n, let $V = \{1, \ldots, n\}$ and $R = G_u^n \cup C^n$. Then, we have that:*

1. *If $\langle V, E\rangle$ is a chordal graph, then there is exactly one stable model X of R such that $\{\{u, v\} \mid edge(u, v) \in X\} = E$.*
2. *If X is a stable model of R, then $\langle V, \{\{u, v\} \mid edge(u, v) \in X\}\rangle$ is a chordal graph.*

Proof. By the same argument as in the proof of Proposition 4 along with line 5 in Figure 10, any stable model X of R must be such that $\{v \mid node(v) \in X\} = V$, $\{\langle u, v\rangle \mid pair(u, v) \in X\} = \{\langle u, v\rangle \mid u, v \in V, u < v\}$, and $\{\langle u, v, w\rangle \mid$

$scope(u,v,w) \in X\} = \{\langle u,v,w\rangle \mid u,v,w \in V, u \neq v, u \neq w, v < w\}$. In view of line 12 and 13 in Figure 10, it must also hold that $\{v \mid order(v) \in X\} = V$ and $\{\langle u,v,w\rangle \mid order(u,v,w) \in X\} = \{\langle u,v,w\rangle \mid u,v,w \in V, u \neq v, u \neq w, v < w\}$, where establishing that $\{order(v) \mid v \in V\} \cup \{order(u,v,w) \mid u,v,w \in V, u \neq v, u \neq w, v < w\} \subseteq \mathrm{LM}(R^X)$ is of interest. We further consider the items of the statement:

1. If $\langle V,E\rangle$ is a chordal graph, let $X = \{node(v), order(v) \mid v \in V\} \cup \{pair(u,v) \mid u,v \in V, u < v\} \cup \{scope(u,v,w), order(u,v,w) \mid u,v,w \in V, u \neq v, u \neq w, v < w\} \cup \{edge(u,v) \mid \{u,v\} \in E, u < v\}$, where the choice rule in line 4 of Figure 2 grants that $\{\langle u,v\rangle \mid edge(u,v) \in X\} = \{\langle u,v\rangle \mid edge(u,v) \in \mathrm{LM}(R^X)\} = \{\langle u,v\rangle \mid \{u,v\} \in E, u < v\}$. Given the above considerations, there cannot be any stable model $X' \neq X$ of R such that $\{\{u,v\} \mid edge(u,v) \in X'\} = E$. Moreover, since $\langle V,E\rangle$ is a chordal graph, there is some "perfect elimination ordering" [12]. That is, for any non-empty $U \subseteq V$, there is some vertex $u \in U$ such that $\{v \mid \{u,v\} \in E, v \in U\}$ is a clique, i.e., $\{edge(v,w) \mid edge(u,v) \in X, edge(u,w) \in X, v,w \in U, v < w\} \cup \{edge(v,w) \mid edge(v,u) \in X, edge(u,w) \in X, v,w \in U\} \cup \{edge(v,w) \mid edge(v,u) \in X, edge(w,u) \in X, v,w \in U, v < w\} \subseteq X$. In view of line 6 to 12 in Figure 10, it follows by induction that $\{order(v) \mid v \in V\} \cup \{order(u,v,w) \mid u,v,w \in V, u \neq v, u \neq w, v < w\} \subseteq \mathrm{LM}(R^X)$, so that $X = \mathrm{LM}(R^X)$ is a stable model of R.
2. If X is a stable model of R, then $X = \mathrm{LM}(R^X)$ implies that, for any non-empty $U \subseteq V$, there is some vertex $u \in U$ such that $\{edge(v,w) \mid edge(u,v) \in X, edge(u,w) \in X, v,w \in U, v < w\} \cup \{edge(v,w) \mid edge(v,u) \in X, edge(u,w) \in X, v,w \in U\} \cup \{edge(v,w) \mid edge(v,u) \in X, edge(w,u) \in X, v,w \in U, v < w\} \subseteq X$, so that $G = \langle V, \{\{u,v\} \mid edge(u,v) \in X\}\rangle$ admits a "perfect elimination ordering" [12], which in turn means that G is chordal. □