Polynomial Constraints for Sets with Cardinality Bounds

Bruno Marnette[1], Viktor Kuncak[2], and Martin Rinard[2]

[1] ENS de Cachan, France
bruno@marnette.fr
[2] MIT CSAIL, Cambridge, USA
{vkuncak,rinard}@csail.mit.edu

Abstract. Logics that can reason about sets and their cardinality bounds are useful in program analysis, program verification, databases, and knowledge bases. This paper presents a class of constraints on sets and their cardinalities for which the satisfiability and the entailment problems are computable in polynomial time. Our class of constraints, based on tree-shaped formulas, is unique in being simultaneously tractable and able to express 1) that a set is a union of other sets, 2) that sets are disjoint, and 3) that a set has cardinality within a given range. As the main result we present a polynomial-time algorithm for checking entailment of our constraints.

1 Introduction

Hierarchical representations of sets of entities are ubiquitous in computer science, arising in programming languages, program analysis, software engineering and knowledge bases. When considering a class of constraints, we are interested in two main questions:
- **satisfiability:** is a set of constraints consistent (satisfiable)?
- **entailment:** does one set of constraints imply (entail) another set of constraints?

Note that a solution to the second problem is also a solution to the first problem: checking whether a set of constraints entails a fixed contradictory constraint solves the satisfiability problem.

In object-oriented programming and software modelling, set hierarchies model classification of entities into classes and are an important component of object models represented using notations such as UML [10] and Alloy [13]. The entailment problem for set hierarchies arises when checking, for example, that one UML diagram is a refinement of another diagram. Satisfiability checking can detect contradictory constraints that indicate an error in the model or system requirements.

Set hierarchies are also essential in knowledge representation [23]. Entailment checking allows one to check that the classification in a particular knowledge-base is a consequence of the classification in a more general ontology.

Recently, researchers have considered the (typestate) generalization of static class hierarchies in object-oriented languages to dynamically changing hierarchies of sets of objects [9, 16]. Using the ideas of [18], we can statically approximate dynamically changing set hierarchy at each program point by propagating constraints between sets of objects using a data-flow analysis. A modular approach to such analysis needs to check that 1) each procedure precondition is satisfied at each procedure call site, and

2) the postcondition holds at the end of each procedure. When the propagated information encodes a set hierarchy, these two checks require deciding the entailment of such hierarchies.

Sets with cardinality constraints. One often wishes to express constraints not only on sets but also on certain distinguished elements of these sets. A simple and unified way to reason about elements is to represent them as sets of cardinality one. Similarly, it is often desirable to state that a set is non-empty or, more generally, that the number of its elements is within given bounds. This motivates the use of cardinality constraints on sets that participate in hierarchies.

We have previously considered expressive logics that can express such constraints by combining the Boolean Algebra of sets with a cardinality operator and Presburger Arithmetic [17], [15, Chapter 7]. However, the NP-hardness of these constraints potentially limits their practical use, which motivated us to find constraints that have polynomial-time algorithms. The result is the class presented in this paper, for which we construct a polynomial-time algorithm for entailment (and therefore satisfiability). This class can express a combination of constraints that, to the best of our knowledge, cannot be represented using existing polynomial-time formalisms (see Section 6).

Our result. We call our notion of set hierarchy *itree*, standing for *inclusion tree*, because the edges in the hierarchy represent set inclusion $B \subseteq A$ and because the inclusion edges in an itree form an inverted tree. Moreover, an itree can specify that a set is covered by some of its subsets ($A = B \cup C \cup D$), or/and that these subsets are pairwise disjoint ($B \cap C = C \cap D = B \cap D = \emptyset$). An itree can also specify multiple orthogonal divisions of one set into subsets, such as $A = B \cup C \wedge A = D \cup E \wedge D \cap E = \emptyset$. Finally, an itree can specify constant cardinality constraints on sets, such as $1 \leq |A| \leq 10000$. Our algorithm checks entailment of conjunctions of such constraints.

The key idea of our polynomial-time algorithm is to define a notion of normal form where each tree node satisfies certain local constraints. We show that this normal form can be enforced in polynomial time using a set of rewrite rules. We then give polynomial-time conditions for checking whether a normalized itree implies a given constraint on variables. This yields an algorithm for checking whether an itree implies a conjunction of such constraints, and we show that an itree can always be represented as a conjunction of quantifier-free constraints. We therefore obtain a polynomial entailment test for itree constraints.

Contributions. The contributions of our paper include the following:

- We introduce itree constraints for expressing hierarchies of sets, and permitting a simple form of existential quantification over sets (Section 2.2).
- We show that generalizing the definition of itrees to permit acyclic graphs yields constraints whose satisfiability is NP-hard (Section 2.3).
- We give a polynomial-time algorithm for checking the satisfiability of itrees by proving sufficient conditions for the existence of their models (Section 3).
- We give a polynomial-time algorithm for checking whether an itree entails a given cardinality, inclusion or disjointness constraint (Section 4).
- We show that the quantifiers in an itree can be eliminated, which, with the previous result, gives polynomial-time entailment for itrees (Section 5).

A preliminary version of the current polynomial-time results (including proofs) appears in the technical report [20], using the same ideas but slightly different definitions. Due to space limitations we here present only proof outlines, describing the main ideas and revealing the underlying algorithms.

2 Constraints on sets and their graphical representation

The constraints that we consider in this paper are expressible using existentially quantified conjunctions of boolean algebra formulas whose variables range over sets of uninterpreted objects. We call these formulas Conjunctive constraints on Sets with Cardinalities and denote them CSC.

Definition 1. CSC *formulas are given by the following syntax:*

$$\begin{array}{l} \phi \ ::= \exists \nu_1, \ldots, \nu_n.\ P_1 \wedge \ldots \wedge P_m \\ P \ := \ S_1 \subseteq S_2 \mid S_1 \cap S_2 = \emptyset \ \mid \ |\nu| \leq k \ \mid \ |\nu| \geq k \\ S \ := \ s \mid \nu \mid S_1 \cup S_2 \end{array}$$

Variables in CSC formulas denote sets and can be free set variables (denoted s, s', s_i) or bound set variables (denoted ν, ν',ν_i). Sets in CSC formulas are denoted by variables or unions of variables. Cardinality constraints apply only to bound variables.

Lemma 1. *Satisfiability of* CSC *formulas is NP-hard.*

Lemma 1 holds because CSC can express boolean algebra constraints on subsets of a fixed set variable U. Namely, union together with disjointness from U can define set complement; union and complement then allow encoding arbitrary propositional operations.

2.1 Graph representation IGRAPH for CSC

As a first step towards identifying polynomial constraints, we introduce a representation of CSC by *igraphs* (standing for *inclusion graphs*). In the following definition of igraphs, the nodes VN are bound set variables ν and the edges $\rightsquigarrow$ represent the subset inclusion of sets. Nodes are tagged with cardinality constraints and with *mode symbols* establishing additional constraints between a node and its direct sons. If ν is tagged with the mode symbol $\bigcirc$, the sons of ν are pairwise disjoint. If ν is tagged with the mode symbol $\blacksquare$, the sons $\{\nu_1, \ldots, \nu_n\}$ of ν cover entirely ν, that is $\nu \subseteq \cup_i \nu_i$. If ν is tagged with the mode symbol $\blacklozenge$, then ν is *equal* to each of its sons. When a set ν participates in several atomic formulas, we can use the $\blacklozenge$ mode to introduce synonyms for ν. Finally, a mapping σ establishes equalities between free set variables $s \in \mathsf{SN}$ and bound variables $\nu \in \mathsf{VN}$. It also enables the encoding of set emptiness using a special symbol $\emptyset_I$.

Definition 2 (IGRAPH)**.** *An* igraph $G \in \mathsf{IGRAPH}$ *is either the* false igraph $\perp_I$ *or a tuple* $(\mathsf{SN}, \mathsf{VN}, \rightsquigarrow, \mathsf{CInf}, \mathsf{CSup}, \mathsf{M}, \sigma)$ *such that*

$$\begin{array}{ll} \mathsf{SN} \ \textit{and}\ \mathsf{VN} \ \textit{are two disjoint sets of set variables} & \\ (\mathsf{VN}, \rightsquigarrow) \ \textit{is a directed graph} & \\ \mathsf{CInf} : \mathsf{VN} \to \mathbb{N} & (\mathbb{N} = \{0, 1, 2, \ldots\}) \\ \mathsf{CSup} : \mathsf{VN} \to \mathbb{N} \cup \{\infty\} & (\forall k \in \mathbb{N}.\ k < \infty) \\ \mathsf{M} : \mathsf{VN} \to \mathcal{P}(\{\blacklozenge, \blacksquare, \bigcirc\}) & \\ \sigma : \mathsf{SN} \to \mathsf{VN} \cup \{\emptyset_I\} & \end{array}$$

The set SN corresponds to the free variables s of G. The elements of VN correspond to the bound variables ν and are also called *nodes* by graph analogy. $\mathcal{P}(\{\blacklozenge, \blacksquare, \bigcirc\})$ denotes the set of subsets of $\{\blacklozenge, \blacksquare, \bigcirc\}$. We write $\nu \rightsquigarrow \nu'$ when $(\nu, \nu') \in \rightsquigarrow$. We define the set of *sons* of $\nu \in \mathsf{VN}$ by $\mathsf{Sons}(\nu) = \{\nu' | \nu' \rightsquigarrow \nu\}$ and the *incoming degree* of ν by $\mathsf{d}(\nu) = |\mathsf{Sons}(\nu)|$.

Definition 3 **(IGRAPH semantics).** *The semantics* $\mathsf{Sem}(\perp_I)$ *of the* false igraph $\perp_I$ *is by definition the formula* false. *With each igraph* $G \neq \perp_I$ *we associate a quantifier-free* CSC *formula* $\mathsf{Sem}_0(G)$ *as follows:*

$$\mathsf{Sem}_0(G) \stackrel{def}{=} \bigwedge \begin{cases} \bigwedge\{\nu' \subseteq \nu \mid \nu, \nu' \in \mathsf{VN} \ \wedge \ \nu' \rightsquigarrow \nu\} \\ \bigwedge\{\nu' = \nu \mid \nu, \nu' \in \mathsf{VN} \ \wedge \ \nu' \rightsquigarrow \nu \wedge \ \blacklozenge \in \mathsf{M}(\nu)\} \\ \bigwedge\{\nu \subseteq \bigcup \mathsf{Sons}(\nu) \mid \nu \in \mathsf{VN} \ \wedge \ \blacksquare \in \mathsf{M}(\nu)\} \\ \bigwedge\{\nu' \cap \nu'' = \emptyset \mid \nu \in \mathsf{VN} \ \wedge \ \nu', \nu'' \in \mathsf{Sons}(\nu) \wedge \nu' \neq \nu'' \ \wedge \bigcirc \in \mathsf{M}(\nu)\} \\ \bigwedge\{\mathsf{CInf}(\nu) \leq |\nu| \leq \mathsf{CSup}(\nu) \mid \nu \in \mathsf{VN}\} \end{cases}$$

The semantics $\mathsf{Sem}(G)$ *of* G *is then:*

$$\mathsf{Sem}(G) \ \stackrel{def}{=} \ \exists \nu_1, \ldots, \nu_n. \ \mathsf{Sem}_0(G) \wedge \bigwedge_{s \in \mathsf{SN}} \begin{cases} s = \emptyset, \ \textit{if} \ \sigma(s) = \emptyset_I \\ s = \nu, \ \textit{if} \ \sigma(s) = \nu \end{cases}$$

Figure 1 gives an example of an igraph G (represented graphically) with its semantics $\mathsf{Sem}(G)$. Given two igraphs G and G' we write $G \models G'$ iff $\mathsf{Sem}(G)$ entails $\mathsf{Sem}(G')$ and we write $G \equiv G'$ when both $G \models G'$ and $G' \models G$. We say that G is satisfiable iff $\mathsf{Sem}(G)$ is satisfiable. We also use the symbols $\models$ and $\equiv$ to compare igraphs and CSC formulas, identifying igraphs G with their semantics $\mathsf{Sem}(G)$. To avoid confusion between the syntax and the semantics of formulas we use square brackets around formulas. Thus, in the following sections, $[\nu = \nu']$ denotes an equality between two sets while $\nu = \nu'$ only states that ν and ν' are the same variable symbol (or the same node).

By construction, the semantics of an igraph is expressible by a CSC formula. The following lemma shows that the converse holds as well.

Lemma 2. *For each* $\phi \in \mathsf{CSC}$ *we can compute in linear time an equivalent igraph.*

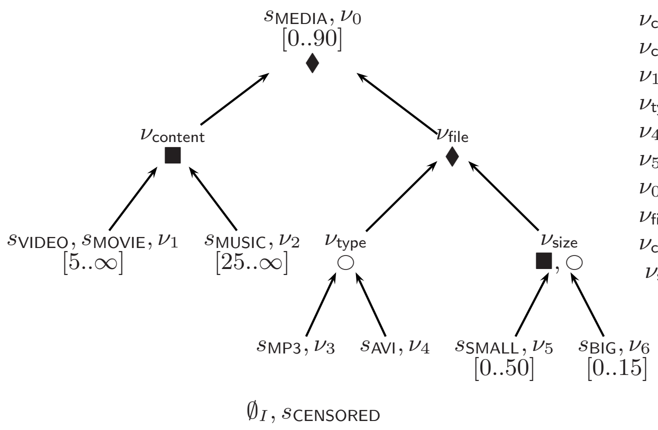
As a consequence, the satisfiability of igraphs is also NP-hard.

2.2 Definition of itrees

We can now define our subclass of tree-shaped igraphs. We call this subclass *itrees*. Polynomial-time algorithms for satisfiability and entailment of itrees are the subject of this paper.

Definition 4 **(ITREE).** *A* generalized itree *(gitree)* T *is either the false igraph* $\perp_I$ *or an igraph* $G \in \mathsf{IGRAPH}$ *such that* $(\mathsf{VN}, \rightsquigarrow)$ *is a tree, oriented from the leaves to the root. An* itree *is a generalized itree such that, for each* $\nu \in \mathsf{VN}$

$$\sigma^{-1}(\nu) = \emptyset \ \ \Rightarrow \ \ \mathsf{CInf}(\nu) = 0 \wedge \mathsf{CSup}(\nu) = \infty \qquad (\mathsf{QE})$$



$$
\begin{aligned}
&\mathsf{Sem}(G) = \\
&\exists \nu_1, \nu_2, \nu_3, \nu_4, \nu_5, \nu_6, \\
&\quad \nu_{\mathsf{content}}, \nu_{\mathsf{file}}, \nu_{\mathsf{type}}, \nu_{\mathsf{size}}. \\
&\quad \nu_{\mathsf{content}} \subseteq \nu_0 \wedge \nu_{\mathsf{file}} \subseteq \nu_0 \wedge \\
&\quad \nu_1 \subseteq \nu_{\mathsf{content}} \wedge \nu_2 \subseteq \nu_{\mathsf{content}} \wedge \\
&\quad \nu_{\mathsf{type}} \subseteq \nu_{\mathsf{file}} \wedge \nu_3 \subseteq \nu_{\mathsf{type}} \wedge \\
&\quad \nu_4 \subseteq \nu_{\mathsf{type}} \wedge \nu_{\mathsf{size}} \subseteq \nu_{\mathsf{file}} \wedge \\
&\quad \nu_5 \subseteq \nu_{\mathsf{size}} \wedge \nu_6 \subseteq \nu_{\mathsf{size}} \\
&\quad \nu_0 = \nu_{\mathsf{content}} = \nu_{\mathsf{file}} \wedge \\
&\quad \nu_{\mathsf{file}} = \nu_{\mathsf{type}} = \nu_{\mathsf{size}} \wedge \\
&\quad \nu_{\mathsf{content}} = \nu_1 \cup \nu_2 \wedge \\
&\quad \nu_{\mathsf{size}} = \nu_5 \cup \nu_6 \wedge \\
&\quad \nu_3 \cap \nu_4 = \emptyset \wedge \\
&\quad \nu_5 \cap \nu_6 = \emptyset \wedge \\
&\quad |\nu_0| \leq 90 \wedge |\nu_1| \geq 5 \wedge \\
&\quad |\nu_2| \geq 25 \wedge \\
&\quad |\nu_5| \leq 50 \wedge |\nu_6| \leq 15 \wedge \\
&\quad s_{\mathsf{MEDIA}} = \nu_0 \wedge s_{\mathsf{MOVIE}} = \nu_1 \wedge \\
&\quad s_{\mathsf{VIDEO}} = \nu_1 \wedge s_{\mathsf{MUSIC}} = \nu_2 \wedge \\
&\quad s_{\mathsf{MP3}} = \nu_3 \wedge s_{\mathsf{AVI}} = \nu_4 \wedge \\
&\quad s_{\mathsf{SMALL}} = \nu_5 \wedge s_{\mathsf{BIG}} = \nu_6 \wedge \\
&\quad s_{\mathsf{CENSORED}} = \emptyset
\end{aligned}
$$

Fig. 1. An example of itree (a particular case of igraph) and its semantics

Thanks to the tree-shape condition, itrees (and even generalized itrees) satisfy some important properties that are not true for general (or acyclic) igraphs. For example, it follows from Lemma 10 of Section 4.2 that the semantics ϕ of an itree always satisfies, for all set variables s_1, s_2, s_3, the following property:

$$\phi \models [s_1 \subseteq s_2 \wedge s_1 \subseteq s_3] \;\Rightarrow\; \big(\phi \models [s_1 = \emptyset] \;\vee\; \phi \models [s_2 \subseteq s_3] \;\vee\; \phi \models [s_3 \subseteq s_2]\big)$$

This property allows us to prove, for example, that the CSC formula $[A \subseteq B \wedge A \subseteq C]$ is not expressible as a (generalised) itree. Therefore, the class ITREE is a strict subclass of IGRAPH and is a good candidate for a more efficient fragment of IGRAPH.

The QE condition (standing for *quantifier elimination*) in the definition of ITREE ensures that the semantics of itrees can in fact be expressed using a quantifier-free CSC formula, as proved in Section 5. Note that a sufficient condition for QE is that $\sigma^{-1}(\nu) \neq \emptyset$ for each $\nu \in \mathsf{VN}$.

Because we can check whether a graph is a tree by depth-first traversal of the graph, we have the following result.

Lemma 3. *Deciding whether a given igraph $G \in \mathsf{IGRAPH}$ is an itree ($G \in \mathsf{ITREE}$) can be done in linear time.*

2.3 Hardness of acyclic igraphs

We have observed that satisfiability of igraphs is NP-hard. In contrast, we prove in the rest of this paper that itrees have polynomial-time satisfiability and entailment problems. A natural question to ask is whether we could obtain polynomial-time algorithms for igraphs where inclusions are acyclic but not tree-like. The following lemma (see also [20, Section 4, Lemma 4]) suggests a negative answer to this question.

Lemma 4. *Let* IDAG *denote the class of igraphs for which* $(\mathsf{VN}, \rightsquigarrow)$ *is a directed acyclic graph (DAG). For each igraph in* IGRAPH *we can compute in polynomial time an equivalent igraph in* IDAG*. Therefore, satisfiability in* IDAG *is NP-hard.*

The essence of the proof of Lemma 4 is that we can collapse (in polynomial time) cycles in an igraph to obtain an equivalent acyclic igraph. In addition to NP-hardness of the class of acyclic igraphs, we can prove NP-hardness for several subclasses of IDAG, using the construction in [20, Section 5, Theorem 2]. We therefore believe that considering tree-like restrictions on igraphs is a reasonable approach to identifying polynomial constraints.

3 Deciding satisfiability of generalized itrees in polynomial time

This section gives a linear-time algorithm for satisfiability of generalized itrees. This result is a first step to an algorithm for checking entailment, which we describe in Section 5, building on the results in this section. Moreover, the satisfiability algorithm is of interest in itself.

We proceed by first showing (Lemma 5) that the bottom-up propagation of constraints (rewriting rules $\mathcal{R}_1$ and $\mathcal{R}_2$) allows transforming in linear time any gitree T into an equivalent gitree $\mathcal{R}_2^{\downarrow}(T)$ such that either **a)** $\mathcal{R}_2^{\downarrow}(T) = \perp_I$, in which case T is clearly unsatisfiable, or **b)** $\mathcal{R}_2^{\downarrow}(T)$ satisfies two properties C_1 and C_2. We then show (Lemma 6) that any gitree for which C_1 and C_2 hold is satisfiable. As a result, we can decide in linear time whether T is satisfiable by first computing $\mathcal{R}_2^{\downarrow}(T)$ and then returning *satisfiable* if and only if $\mathcal{R}_2^{\downarrow}(T)$ is different from $\perp_I$.

Lemma 5. *For each gitree* T *we can compute in linear time an equivalent gitree* $\mathcal{R}_2^{\downarrow}(T)$ *such that either* $\mathcal{R}_2^{\downarrow}(T) = \perp_I$ *or* $\mathcal{R}_2^{\downarrow}(T)$ *satisfies (for each node* ν*) both:*

$$\mathsf{M}(\nu) \in \{\emptyset, \{\blacklozenge\}, \{\bigcirc\}, \{\blacksquare\}, \{\bigcirc, \blacksquare\}\} \qquad (C_1(\nu))$$

and

$$\mathsf{BUInf}(\nu) \leq \mathsf{CInf}(\nu) \leq \mathsf{CSup}(\nu) \leq \mathsf{BUSup}(\nu) \qquad (C_2(\nu))$$

where, for $\mathsf{Sons}(\nu) = \{\nu_1, \ldots, \nu_n\}$,

$$\mathsf{BUInf}(\nu) \overset{def}{=} \begin{cases} \sum_i \mathsf{CInf}(\nu_i), & \text{if } \bigcirc \in \mathsf{M}(\nu) \\ \max_i \mathsf{CInf}(\nu_i), & \text{otherwise} \end{cases}$$

$$\mathsf{BUSup}(\nu) \overset{def}{=} \begin{cases} \min_i \mathsf{CSup}(\nu_i), & \text{if } \blacklozenge \in \mathsf{M}(\nu) \\ \sum_i \mathsf{CSup}(\nu_i), & \text{if } \blacksquare \in \mathsf{M}(\nu) \\ \infty, & \text{otherwise} \end{cases}$$

Proof. Such a form $\mathcal{R}_2^{\downarrow}(T)$ can be obtained from T in two steps. The first steps consists in simplifying the mode combinations by applying the following rewriting rule R_1 to every node (in any order).

if	apply
$\mathsf{d}(\nu) = 0$	$\mathsf{M}(\nu) := (\mathsf{M}(\nu) - \{\blacklozenge\})$
$\mathsf{d}(\nu) \leq 1$	$\mathsf{M}(\nu) := (\mathsf{M}(\nu) - \{\bigcirc\})$
$\mathsf{d}(\nu) \geq 1$ $\blacklozenge \in \mathsf{M}(\nu)$	$\mathsf{M}(\nu) := (\mathsf{M}(\nu) - \{\blacksquare\})$
$\mathsf{d}(\nu) \geq 2$ $\{\blacklozenge, \bigcirc\} \subseteq \mathsf{M}(\nu)$	$\mathsf{M}(\nu) := (\mathsf{M}(\nu) - \{\bigcirc\})$ $\forall \nu' {\in} \mathsf{Sons}(\nu),\ \ \mathsf{CSup}(\nu') := 0$

$(\mathcal{R}_1(\nu))$

The second step consists in applying the rule $\mathcal{R}_2$ below to every node, proceeding *from the leaves towards the root*, in order to 1) refine the cardinality bounds and 2) recognize the contradictory bounds such that $\mathsf{CInf}(\nu) > \mathsf{CSup}(\nu)$.

$\mathsf{CInf}(\nu)$	$:=$	$\mathsf{Max}(\mathsf{CInf}(\nu), \mathsf{BUInf}(\nu))$
$\mathsf{CSup}(\nu)$	$:=$	$\mathsf{Min}(\mathsf{CSup}(\nu), \mathsf{BUSup}(\nu))$
If $\mathsf{CInf}(\nu) > \mathsf{CSup}(\nu)$ then $T := \perp_I$		

$(\mathcal{R}_2(\nu))$

□

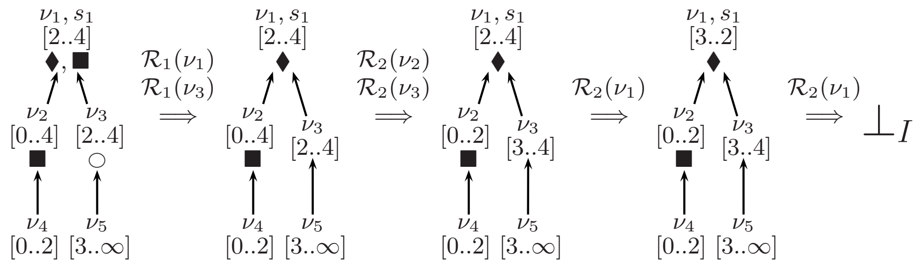


Fig. 2. Example of $\mathcal{R}_1$ and $\mathcal{R}_2$ derivation

We say that C_i *holds for* T (i.e. "T *satisfies* C_i") iff $C_i(\nu)$ holds for each node ν of T.

Lemma 6. *Every gitree $T \neq \perp_I$ for which both C_1 and C_2 hold is satisfiable.*

Proof. We first note that a gitree T such that $T \neq \perp_I$ is satisfiable if and only if there exists a model for $\mathsf{Sem}_0(T)$. Indeed, a model $(\Delta, \alpha : \mathsf{VN} \to \mathcal{P}(\Delta))$ for $\mathsf{Sem}_0(T)$ can be turned into a model $(\Delta, \alpha' : \mathsf{SN} \to \mathcal{P}(\Delta))$ for $\mathsf{Sem}(T)$ by taking $\alpha'(s) = \emptyset$ when $\sigma(s) = \emptyset_I$ and $\alpha'(s) = \alpha(\sigma(s))$ when $\sigma(s) \in \mathsf{VN}$.

When T satisfies both C_1 and C_2 we can build a model for $\mathsf{Sem}_0(T)$ in two steps. We first choose (in linear time) relevant cardinalities $\psi(\nu) \in \mathbb{N}$ for the nodes ν, proceeding *from the root to the leaves*. More precisely, we take $\psi(\nu) = \mathsf{CInf}(\nu)$ for the root ν of T and define recursively the values $\psi(\nu_i)$ for the sons ν_i of a node ν, for a chosen ordering of $\mathsf{Sons}(\nu) = \{\nu_1, \ldots, \nu_n\}$, and by induction on $i = 1..n$:

$$\psi(\nu_i) \stackrel{def}{=} \begin{cases} \mathsf{CInf}(\nu_i) & \text{if } \mathsf{M}(\nu) = \emptyset \\ \mathsf{CInf}(\nu_i) & \text{if } \mathsf{M}(\nu) = \{\bigcirc\} \\ \psi(\nu) & \text{if } \mathsf{M}(\nu) = \{\blacklozenge\} \\ \min(\mathsf{CSup}(\nu_i), \psi(\nu)) & \text{if } \mathsf{M}(\nu) = \{\blacksquare\} \\ \min(\mathsf{CSup}(\nu_i), \psi(\nu) - \sum\limits_{i'<i} \psi(\nu_{i'}) - \sum\limits_{i'>i} \mathsf{CInf}(\nu_{i'})) & \text{if } \mathsf{M}(\nu) = \{\bigcirc, \blacksquare\} \end{cases}$$

The conditions C_1 and C_2 guarantee that this cardinality choice satisfies the following property H^ψ for every node ν such that $\mathsf{Sons}(\nu) = \{\nu_1, \ldots, \nu_n\}$:

$$\left.\begin{array}{l} \mathsf{CInf}(\nu) \leq \psi(\nu) \leq \mathsf{CSup}(\nu) \\ \bigwedge_i \ \psi(\nu_i) \leq \psi(\nu) \\ \blacklozenge \in \mathsf{M}(\nu) \Rightarrow \ \bigwedge_i \psi(\nu_i) = \psi(\nu) \\ \bigcirc \in \mathsf{M}(\nu) \Rightarrow \ \sum_i \psi(\nu_i) \leq \psi(\nu) \\ \blacksquare \in \mathsf{M}(\nu) \Rightarrow \ \sum_i \psi(\nu_i) \geq \psi(\nu) \end{array}\right\} (H^\psi(\nu))$$

When a cardinality choice satisfying $H^\psi(\nu)$ for all nodes ν of T is chosen, the second step consists in building for every node ν, taken from the leaves to the root, a model for the formula $\mathsf{Sem}_0(T|_\nu)$ where $T|_\nu$ denotes the sub-itree $T|_\nu$ of T of root ν. The role played by $H^\psi(\nu)$ in this construction is the following: the property $\psi(\nu_i) \leq \psi(\nu)$ ensures that the son ν_i of ν is small enough to fit in ν; when $\mathsf{M}(\nu) = \{\blacklozenge\}$, the property $\psi(\nu_i) = \psi(\nu)$ ensures that the sons ν_i of ν have the right cardinality to be made equal to ν; when $\bigcirc \in \mathsf{M}(\nu)$ the property $\sum_i \psi(\nu_i) \leq \psi(\nu)$ ensures that the disjoint union of the sons of ν can fit in ν; when $\blacksquare \in \mathsf{M}(\nu)$, the property $\sum_i \ \psi(\nu_i) \geq \psi(\nu)$ ensures that the sons of ν contain enough elements to cover entirely ν; the property $\mathsf{CInf}(\nu) \leq \psi(\nu) \leq \mathsf{CSup}(\nu)$ ensures that the cardinality constraints are not violated. □

Corollary 1. *A gitree T is satisfiable iff $\mathcal{R}_2^{\downarrow}(T) \neq \perp_I$.*

Corollary 2. *We can decide the satisfiability of a gitree in linear time.*

4 Entailment of quantifier-free formulas

The goal of this section is to show that for every gitree T (and, in particular, for every itree $T \in \mathsf{ITREE}$), and every formula ϕ from a quantifier-free fragment QFCSC of CSC defined below, we can decide whether T entails ϕ in polynomial time.

Definition 5 (QFCSC).

$$\begin{array}{l} \phi \ ::= P_1 \wedge \ldots \wedge P_m \\ P := \ S_1 \subseteq S_2 \mid S_1 \cap S_2 = \emptyset \ \mid \ |s| \leq k \ \mid \ |s| \geq k \\ S := \ s \mid S_1 \cup S_2 \end{array}$$

Because QFCSC formulas are conjunctions of atomic formulas, we can decide whether T entails a formula $\Phi = P_1 \wedge \ldots \wedge P_n$ by checking whether $T \models P_i$ for all $i = 1..n$. For deciding $T \models P$ we start by applying additional rewriting rules that enforce stronger properties on gitrees than in the previous section. For each kind of atomic proposition P (cardinality constraint, inclusion, or disjointness) we then define conditions on normalized gitrees that 1) characterize the property $T \models P$, and 2) are computable in polynomial time.

4.1 Checking cardinality constraints

Analogously to the definition of BUInf and BUSup (in Lemma 5) we next define for every node ν of a gitree T a lower bound $\mathsf{TDInf}(\nu)$ and an upper bound $\mathsf{TDSup}(\nu)$ for the cardinality of ν, this time corresponding to top-down reasoning. Given a node ν such that $\nu = \mathsf{Root}(T)$ or $\nu \rightsquigarrow \nu'$ and $\mathsf{Sons}(\nu') = \{\nu, \nu_1, \ldots, \nu_n\}$ we define

$$\mathsf{TDInf}(\nu) \stackrel{def}{=} \begin{cases} 0, & \text{if } \nu = \mathsf{Root}(T) \\ 0, & \text{if } \mathsf{M}(\nu') \in \{\emptyset, \{\blacksquare\}\} \\ \mathsf{CInf}(\nu'), & \text{if } \mathsf{M}(\nu') = \{\blacklozenge\} \\ \mathsf{CInf}(\nu') - \sum_i \mathsf{CSup}(\nu_i), & \text{if } \mathsf{M}(\nu') \in \{\{\bigcirc\}, \{\bigcirc, \blacksquare\}\} \end{cases}$$

$$\mathsf{TDSup}(\nu) \stackrel{def}{=} \begin{cases} \infty, & \text{if } \nu = \mathsf{Root}(T) \\ \mathsf{CSup}(\nu'), & \text{if } \mathsf{M}(\nu') \in \{\emptyset, \{\blacklozenge\}, \{\bigcirc\}\} \\ \mathsf{CSup}(\nu') - \sum_i \mathsf{CInf}(\nu_i), & \text{if } \mathsf{M}(\nu') \in \{\{\blacksquare\}, \{\bigcirc, \blacksquare\}\} \end{cases}$$

Lemma 7. *For each satisfiable gitree T we can compute in linear time an equivalent gitree $\mathcal{R}_3^{\downarrow}(T)$ satisfying C_1, C_2, and, for each $\nu \in \mathsf{VN}$,*

$$\mathsf{TDInf}(\nu) \leq \mathsf{CInf}(\nu) \ \wedge\ \mathsf{CSup}(\nu) \leq \mathsf{TDSup}(\nu) \qquad (C_3(\nu))$$

Proof. Such a gitree $\mathcal{R}_3^{\downarrow}(T)$ can be obtained by applying the following rule $\mathcal{R}_3$ to $\mathcal{R}_2^{\downarrow}(T)$ using a top-down strategy

$$\boxed{\begin{array}{l} \mathsf{CInf}(\nu) := \mathsf{Max}(\mathsf{CInf}(\nu), \mathsf{TDInf}(\nu)) \\ \hline \mathsf{CSup}(\nu) := \mathsf{Min}(\mathsf{CSup}(\nu), \mathsf{TDSup}(\nu)) \end{array}} \qquad (\mathcal{R}_3(\nu))$$

□

Lemma 8 (Checking cardinality constraints). *For each gitree T satisfying C_1, C_2 and C_3, each $s \in \mathsf{SN}$, and each $a, b \in \mathbb{N}$ we have*

$$T \models [a \leq |s| \leq b] \iff \begin{cases} \textit{either } \sigma(s) = \emptyset_I \ \wedge\ a = 0 \\ \textit{or } a \leq \mathsf{CInf}(\sigma(s)) \leq \mathsf{CSup}(\sigma(s)) \leq b \end{cases}$$

We can therefore decide whether $T \models [a \leq |s| \leq b]$ in linear time.

Proof. For each $T \neq \perp_I$ satisfying C_1, C_2, C_3, for each $\nu \in \mathsf{VN}$ and for each $k \in [\mathsf{CInf}(\nu), \mathsf{CSup}(\nu)]$, the gitree $T_{|\nu| \leftarrow k}$ obtained from T by applying $\mathsf{CInf}(\nu) := k$ and $\mathsf{CSup}(\nu) := k$ satisfies $\mathcal{R}_2^{\downarrow}(T_{|\nu| \leftarrow k}) \neq \perp_I$. Therefore, by Corollary 1, there exists a model (Δ, α) of $\mathsf{Sem}_0(T)$ such that $|\alpha(\nu)| = k$.

When T satisfies C_1, C_2, C_3 we can then check that the cardinality bounds CInf and CSup are optimal. That is, for every node ν of such a gitree we have $\mathsf{CInf}(\nu) = \min\{|\alpha(\nu)|, (\Delta, \alpha) \models \mathsf{Sem}_0(T)\}$ and $\mathsf{CSup}(\nu) = \max\{|\alpha(\nu)|, (\Delta, \alpha) \models \mathsf{Sem}_0(T)\}$. The result follows directly from this observation. □

4.2 Checking inclusion and disjointness constraints

Now that we have optimal cardinality bounds, it is natural to look at the influence of cardinality constraints on other types of constraints. This approach allows us to enforce an additional property C_4 on gitrees using a rewriting system $\mathcal{R}_4$. Finally, we show how to take advantage of C_4 to decide which inclusion constraints (Lemma 10) or which pairwise disjointness (Lemma 11) hold in a gitree T.

Lemma 9. *For each satisfiable gitree T we can compute in linear time an equivalent gitree $\mathcal{R}_4^{\downarrow}(T)$ satisfying C_1, C_2, C_3, and, for each $\nu \in \mathsf{VN}$,*

$$\left.\begin{array}{l} \mathsf{CSup}(\nu) > 0 \\ \mathsf{d}(v) = 1 \Rightarrow \mathsf{M}(\nu) \in \{\{\blacklozenge\}, \{\bigcirc\}\} \\ \mathsf{M}(\nu) = \{\blacksquare\} \Rightarrow \mathsf{CInf}(\nu) < \sum\limits_{\nu' \rightsquigarrow \nu} \mathsf{CSup}(\nu') \\ \mathsf{M}(\nu) = \{\bigcirc\} \Rightarrow \mathsf{CSup}(\nu) > \sum\limits_{\nu' \rightsquigarrow \nu} \mathsf{CInf}(\nu') \end{array}\right\} \quad (C_4(\nu))$$

Proof. Such a gitree $\mathcal{R}_4^{\downarrow}(T)$ can be obtained by applying with a bottom-up strategy the following rule $\mathcal{R}_4$ to $\mathcal{R}_3^{\downarrow}(T)$

if	apply
$\mathsf{d}(\nu) = 1$ $\mathsf{M}(\nu) = \emptyset$	$\mathsf{M}(\nu) := \{\bigcirc\}$
$\mathsf{M}(\nu) = \{\bigcirc\}$ $\mathsf{CSup}(\nu) \leq \sum_{\nu' \rightsquigarrow \nu} \mathsf{CInf}(\nu')$	$\mathsf{M}(\nu) := \{\blacksquare, \bigcirc\}$
$\mathsf{M}(\nu) = \{\blacksquare\}$ $\mathsf{CInf}(\nu) \geq \sum_{\nu' \rightsquigarrow \nu} \mathsf{CSup}(\nu')$	$\mathsf{M}(\nu) := \{\blacksquare, \bigcirc\}$
$\mathsf{d}(\nu) = 1$ $\blacksquare \in \mathsf{M}(\nu)$	$\mathsf{M}(\nu) := \{\blacklozenge\}$
$\mathsf{CSup}(\nu) = 0$ $\mathsf{d}(\nu) = 0$	$\mathsf{VN} := \mathsf{VN} - \{\nu\}$ $\forall s \in \sigma^{-1}(\nu)$ $\sigma(s) := \emptyset_I$

$(\mathcal{R}_4(\nu))$

□

Lemma 10 (Checking inclusion constraints). *For each gitree T and each $X \subseteq \mathsf{VN}$ we define a unary predicate $\mathcal{I}_X$ on VN as the least fixed point of*

$$\begin{array}{l} \mathcal{I}_X(\nu) \Leftarrow \nu \in X \\ \mathcal{I}_X(\nu) \Leftarrow \blacklozenge \in \mathsf{M}(\nu) \wedge (\exists \nu' \in \mathsf{Sons}(\nu)\, \mathcal{I}_X(\nu)) \\ \mathcal{I}_X(\nu) \Leftarrow \blacksquare \in \mathsf{M}(\nu) \wedge (\forall \nu' \in \mathsf{Sons}(\nu)\, \mathcal{I}_X(\nu)) \\ \mathcal{I}_X(\nu) \Leftarrow \nu \rightsquigarrow \nu' \wedge \mathcal{I}_X(\nu') \end{array}$$

Then, if T satisfies C_1, C_2, C_3, C_4, then for all subsets $S, S' \subseteq \mathsf{SN}$, for $X' = \{\sigma(s) \mid s \in S' \wedge \sigma(s) \in \mathsf{VN}\}$ we have:

$$T \models [(\cup S) \subseteq (\cup S')] \quad \Longleftrightarrow \quad \forall s \in S \ \big(\sigma(s) = \emptyset_I \vee \mathcal{I}_{X'}(\sigma(s))\big)$$

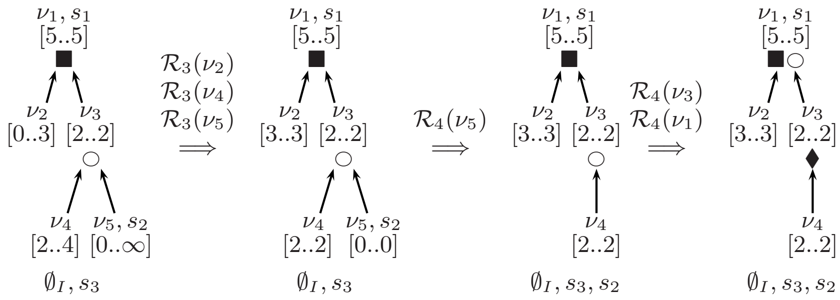


Fig. 3. Example of $\mathcal{R}_3$ and $\mathcal{R}_4$ derivations

Proof. The result is a consequence of the following observation: for each T satisfying C_1, C_2, C_3, C_4, for each $\nu \in \mathsf{VN}$ and each $X \subseteq \mathsf{VN}$ we have:

$$\mathsf{Sem}_0(T) \models [\nu \subseteq (\cup X)] \iff \mathcal{I}_X(\nu)$$

The proof of this claim relies on a refinement of the algorithm of model construction used in the proof of Lemma 6. □

Lemma 11 (Checking disjointness constraints). *For each gitree T we define the binary predicates $\mathcal{D}$ and $\mathcal{D}^*$ on $\mathsf{VN}\times\mathsf{VN}$ by*

$$\begin{aligned}\mathcal{D}(\nu,\nu') &\iff \nu \neq \nu' \wedge \exists \nu'' \in \mathsf{VN}, \{\nu,\nu'\} \subseteq \mathsf{Sons}(\nu'') \wedge \bigcirc \in \mathsf{M}(\nu'')\\ \mathcal{D}^*(\nu,\nu') &\iff \exists \nu_0, \nu_0' \in \mathsf{VN}, \nu \rightsquigarrow^* \nu_0 \wedge \nu' \rightsquigarrow^* \nu_0' \wedge \mathcal{D}(\nu_0,\nu_0')\end{aligned}$$

Then, if C_1, C_2, C_3, C_4, for all subsets S, S' of SN *we have*

$$T \models [(\cup S)\cap(\cup S') = \emptyset] \iff \forall (s,s') \in (S\times S') \begin{cases} \textit{either} \;\; \sigma(s) = \emptyset_I \vee \sigma(s') = \emptyset_I \\ \textit{or} \quad\;\; \mathcal{D}^*(\sigma(s), \sigma(s')) \end{cases}$$

Proof. The result is a consequence of the following observation, which again relies on a refinement of the algorithm of model construction: when T satisfies C_1, C_2, C_3, C_4 then for all $\nu, \nu' \in \mathsf{VN}$ we have $\mathsf{Sem}_0(T) \models [\nu \cap \nu' = \emptyset] \iff \mathcal{D}^*(\nu,\nu')$ □

We conclude this section by combining Lemmas 8,10, and 11 to prove the following theorem.

Theorem 1. *We can decide whether a gitree entails a* QFCSC *formula in polynomial time.*

5 Testing entailment of itrees in polynomial-time

The test of entailment between two arbitrary gitrees ($T \models T'$) is complicated by the existential quantifiers in the semantics of T' which prevent us from decomposing $\mathsf{Sem}(T')$ into a conjunction of independent atomic formulas. However, if we can express a gitree

T' as a QFCSC formula, the previous section yields a polynomial-time algorithm for checking whether a gitree entails T'. In this section we show that the condition

$$\sigma^{-1}(\nu) = \emptyset \quad \Rightarrow \quad \mathsf{CInf}(\nu) = 0 \wedge \mathsf{CSup}(\nu) = \infty \qquad (\mathsf{QE})$$

in the definition of itrees ensures that we can indeed compute a QFCSC formula associated with the itree, which motivates the definition of itrees as a subclass of gitrees.

As a first step, the following lemma gives a sufficient condition for a node of an itree (that is, a bound variable) to be expressible as a union of some free variables.

Lemma 12. *Given an itree T we define the unary predicate* Det *on* VN *as the least fixed point of*

$$\begin{array}{lcl}
\mathsf{Det}(\nu) & \Leftarrow & \sigma^{-1}(\nu) \neq \emptyset \\
\mathsf{Det}(\nu) & \Leftarrow & \blacksquare \in \mathsf{M}(\nu) \wedge \forall \nu' \in \mathsf{Sons}(\nu).\ \mathsf{Det}(\nu') \\
\mathsf{Det}(\nu) & \Leftarrow & \mathsf{M}(\nu) = \{\blacklozenge\} \wedge \exists \nu' \in \mathsf{Sons}(\nu).\ \mathsf{Det}(\nu') \\
\mathsf{Det}(\nu) & \Leftarrow & \exists \nu'.\ \nu \rightsquigarrow \nu' \wedge \mathsf{M}(\nu') = \{\blacklozenge\} \wedge \mathsf{Det}(\nu')
\end{array}$$

Then for each node ν we have: $\mathsf{Det}(\nu) \Rightarrow \big(\exists S_\nu \subseteq \mathsf{SN}.\ T \models [\nu = (\cup S_\nu)]\big)$
Moreover, the predicate Det *and a mapping $\nu \mapsto S_\nu$ are computable in polynomial time.*

When $\mathsf{Det}(\nu)$ holds for all nodes ν of a gitree T, it is clear that we can transform the formula $\phi = \mathsf{Sem}(T)$ into an equivalent formula ϕ' such that no quantified variable appears inside inclusion constraints or disjointness constraints. However, it is not sufficient to check that $\mathsf{Det}(\nu)$ holds for all nodes ν to ensure that $T \in$ QFCSC. Indeed, QFCSC only allows expressing cardinality constraints on free set variables and not on arbitrary union of set variables. It is for this reason that we are naturally interested in the class ITREE of gitrees for which non trivial cardinality constraints can only be enforced to nodes ν for which there exists $s \in \mathsf{SN}$ such that $\sigma(s) = \nu$.

Lemma 13. *For each itree $T \in$* ITREE *we can compute in polynomial time an equivalent itree $\mathcal{R'}^{\downarrow}(T)$ satisfying* $\mathsf{Det}(\nu)$ *for all nodes $\nu \in$* VN*.*

Proof. Given an itree T we can first compute an itree T_1 equivalent to T and satisfying the condition (C_1) on M. Because $\mathcal{R}_1$ does not preserve the QE condition, we compute T_1 in three steps: 1) discard the cardinality constraints of T by applying $\mathsf{CInf}(\nu) := 0$ and $\mathsf{CSup}(\nu) := \infty$ to every node; 2) apply $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4$; and 3) recover the initial cardinality constraints on the nodes that remain in the tree. Step 2) makes some subtrees of T empty and changes $\mathsf{M}, \mathsf{CSup}, \mathsf{CInf}$ for existing nodes, but never introduces new nodes or causes $\sigma^{-1}(\nu) = \emptyset$ to hold for additional nodes that remain in the tree. Thus, QE holds after step 3). We can compute the final itree $\mathcal{R'}^{\downarrow}(T)$, equivalent to T and T_1, by applying the rewriting rule $\mathcal{R}'$ of Figure 4 to T_1 using a bottom-up strategy. □

Lemma 14 **(ITREE $\subseteq$ QFCSC).** *For each itree $T \in$* ITREE *we can compute in polynomial time an equivalent formula of* QFCSC*.*

Given a mapping $\nu \mapsto S_\nu$ from Lemma 12, this QFCSC formula can be computed from $\mathsf{Sem}(\mathcal{R'}^{\downarrow}(T))$ by first substituting each ν with $\cup S_\nu$ in the formula $\mathsf{Sem}(\mathcal{R'}^{\downarrow}(T))$ and then eliminating the quantifiers.

if	apply
$\nu \rightsquigarrow \nu'$ $\mathsf{d}(\nu) = 0$ $\sigma^{-1}(\nu) = \emptyset$	$\mathsf{VN} := \mathsf{VN} - \{\nu\}$ $\mathsf{M}(\nu') := \mathsf{M}(\nu') - \{\blacksquare\}$
$\nu = \mathsf{Root}(T)$ $\mathsf{d}(\nu) = 0$ $\sigma^{-1}(\nu) = \emptyset$	$\mathsf{VN} := \mathsf{VN} - \{\nu\}$
$\mathsf{M}(\nu) \in \{\emptyset, \{\bigcirc\}\}$ $\nu \rightsquigarrow \nu'$ $\mathsf{M}(\nu') \neq \{\blacklozenge\}$ $\sigma^{-1}(\nu) = \emptyset$	$\mathsf{M}(\nu) := \mathsf{M}(\nu) \cup \{\blacksquare\}$ $\mathsf{M}(\nu') := \mathsf{M}(\nu') - \{\blacksquare\}$
$\mathsf{M}(\nu') = \{\blacklozenge\}$ $\forall \nu \in \mathsf{Sons}(\nu')$ $\quad \mathsf{M}(\nu) \in \{\emptyset, \{\bigcirc\}\}$ $\quad \sigma^{-1}(\nu) = \emptyset$	$\mathsf{M}(\nu') := \emptyset$ $\forall \nu \in \mathsf{Sons}(\nu')$ $\quad \mathsf{M}(\nu) := \mathsf{M}(\nu) \cup \{\blacksquare\}$

$(\mathcal{R}'(\nu))$

Fig. 4. Rewriting rule $\mathcal{R}'$

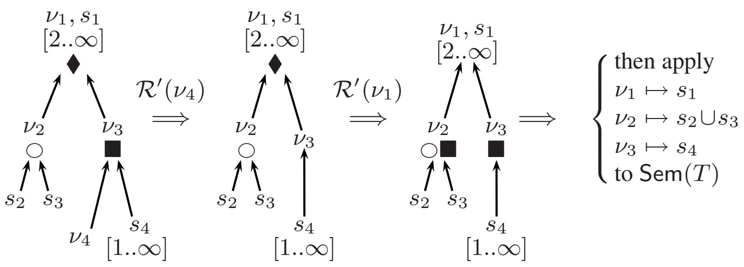


Fig. 5. Use of $\mathcal{R}'$ on an itree and quantifier elimination

Figure 5 gives an example of an application of $\mathcal{R}'$ to an itree and indicates which substitution can finally be applied to obtain a quantifier-free formula. Finally, combining Lemma 14 and Theorem 1, we obtain the main theorem of this paper.

Theorem 2. *We can decide entailment of itrees in polynomial time.*

6 Related work

We are not aware of any previously known constraints on sets with cardinality constraints that have polynomial-time entailment while supporting all the constraints present in the ITREE class.

Set algebras with cardinalities. Quantified formulas of boolean algebra are complete for the class of alternating exponential time with a linear number of alternations [14], and even a small number of alternations leads to exponential complexity [12]. Cardinality constraints naturally arise in quantifier elimination for boolean algebras [19]. The quantifier-free case of Boolean Algebra with Presburger Arithmetic is described in [5, Section 11], [25] with an non-deterministic exponential time decision procedure, which is also achieved as a special case of [17]. Recently, [15, Section 7.9] gave a non-deterministic *polynomial-time* algorithm for quantifier-free Boolean Algebra with Presburger Arithmetic. All these constraints are NP-hard.

Description logics. Description logics [3] can reason about sets (concepts) and relations (roles). However, polynomial-time description logics such as the ones described in [8, Section 7] and [2], [3, Section 3.9.2] do not support set unions; the presence of union is generally considered to lead to intractability. Note also that the subsumption in the context of description logic typically refers to testing $A \subseteq B$ for two defined concepts A and B, as opposed to testing whether a *conjunction* of constraints on sets implies another constraint on sets, as in our case. Furthermore, cardinality constraints in description logics typically apply to a relation and are used to designate a new set, as opposed to imposing a constraint on an existing set.

Horn clause fragments. Polynomial-time fragments of first-order logic Horn clauses such as [11,21] can in principle encode some relationships on sets by representing them as predicates, but they do not support cardinality constraints.

Constraint satisfaction problems. Constraint satisfaction problems (CSP) [7] also identify the important idea of propagating constraints along tree-like structures. For example, the Yannakakis algorithm has linear time complexity for the satisfiability of acyclic sets of constraints [24]. However, such algorithms typically work on concrete domains such as booleans or integers; we are not aware of their application to constraints that involve set variables along with their cardinalities. Indeed, an attempt to generalize itrees to acyclic graphs yields NP-hard constraints. Note that representing the values of set variables explicitly (as done in many constraint satisfaction problems over finite domains) would result in exponentially large models. Like [8, Section 7], our polynomial algorithm avoids this problem using polynomial representation of models, but, unlike [8, Section 7], can express conjunctions of constraints of the form $A = B \cup C$.

Tree-width. The notion of tree-width [22] can be used as a mesure of the "treeness" of a conjunctive formula and often leads to polynomial results on classes of formulas with bounded tree-width. However, although inclusion constraints in an itree form a tree and syntactically have bounded tree-width, disjointness and union constraints introduce dependencies between siblings of a tree. Therefore, the overall tree-width of an itree formula is not bounded. Similarly, the result [6], stating that monadic second-order logic queries over graph structures of bounded tree-width are polynomial, does not seem to simplify the problem of checking entailment (or satisfiability) of itrees. Indeed, there is no natural way of representing, for example, cardinality bounds on sets in monadic second-order logic.

Constraints in program analysis. Set constraints [1, 4] are incomparable to our constraints. On the one hand, set constraints are interpreted over ground terms and contain operations that apply a given free function symbol to each element of the set. On the other hand, unlike our constraints, set constraints do not support cardinality operators.

References

1. Alex Aiken, Dexter Kozen, Moshe Vardi, and Ed Wimmers. The complexity of set constraints. In *Proceedings of Computer Science Logic 1993*, pages 1–17, September 1993.
2. F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proc. 19th Int. Joint Conf. on Artificial Intelligence IJCAI-05*, 2005.
3. Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. CUP, 2003.
4. Leo Bachmair, Harald Ganzinger, and Uwe Waldmann. Set constraints are the monadic class. In *Logic in Computer Science*, pages 75–83, 1993.
5. Domenico Cantone, Eugenio Omodeo, and Alberto Policriti. *Set Theory for Computing*. Springer, 2001.
6. Bruno Courcelle. The monadic second-order logic of graphs III: tree-decompositions, minor and complexity issues. *ITA*, 26:257–286, 1992.
7. Rina Dechter. *Constraint Processing*. Morgan-Kaufmann, 2003.
8. Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. The complexity of concept languages. *Information and Computation*, 134(1):1–58, 1997.
9. Sophia Drossopoulou, Ferruccio Damiani, Mariangiola Dezani-Ciancaglini, and Paola Giannini. More Dynamic Object Re-classification: FickleII. *ACM Trans. Programming Languages and Systems*, 24(2):153–191, 2002.
10. Martin Fowler. *UML Distilled (Second Edition)*. Addison-Wesley, Reading, Mass., 2000.
11. Robert Givan and David Mcallester. Polynomial-time computation via local inference relations. *ACM Trans. Comput. Logic*, 3(4):521–541, 2002.
12. Erich Grädel. Domino games with an application to the complexity of boolean algebras with bounded quantifier alternations. In *STACS*, pages 98–107, 1988.
13. Daniel Jackson. *Software Abstractions: Logic, Language, & Analysis*. MIT Press, 2006.
14. Dexter Kozen. Complexity of boolean algebras. *Theoretical Computer Science*, 10:221–247, 1980.
15. Viktor Kuncak. *Modular Data Structure Verification*. PhD thesis, EECS Department, Massachusetts Institute of Technology, February 2007.
16. Viktor Kuncak, Patrick Lam, and Martin Rinard. Role analysis. In *Annual ACM Symp. on Principles of Programming Languages (POPL)*, 2002.
17. Viktor Kuncak, Hai Huu Nguyen, and Martin Rinard. Deciding Boolean Algebra with Presburger Arithmetic. *J. of Automated Reasoning*, 2006. http://dx.doi.org/10.1007/s10817-006-9042-1.
18. Patrick Lam, Viktor Kuncak, and Martin Rinard. Generalized typestate checking for data structure consistency. In *6th Int. Conf. Verification, Model Checking and Abstract Interpretation*, 2005.
19. L. Loewenheim. Über Mögligkeiten im Relativkalkül. *Math. Annalen*, 76:228–251, 1915.
20. Bruno Marnette, Viktor Kuncak, and Martin Rinard. On algorithms and complexity for sets with cardinality constraints. Technical report, MIT CSAIL, August 2005.
21. Flemming Nielson, Hanne Riis Nielson, and Helmut Seidl. Normalizable Horn clauses, strongly recognizable relations, and Spi. In *SAS*, pages 20–35, 2002.
22. Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.
23. James G. Schmolze and Thomas A. Lipkis. Classification in the KL-ONE knowledge representation system. In *IJCAI*, pages 330–332, 1983.
24. Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94, 1981.
25. Calogero G. Zarba. Combining sets with cardinals. *J. of Automated Reasoning*, 34(1), 2005.