On Algorithms and Complexity for Sets with Cardinality
Constraints

PSPACE and PTIME Logics for Program Analysis

Bruno Marnette
ENS de Cachan, MPRI
Paris, France
bruno@marnette.fr

Viktor Kuncak
MIT Computer Science and Artificial Intelligence Lab
Cambridge, USA
{vkuncak,rinard}@csail.mit.edu

Martin Rinard

Abstract

Typestate systems ensure many desirable properties of imperative
programs, including initialization of object fields and correct use of
stateful library interfaces. Abstract sets with cardinality constraints
naturally generalize typestate properties: relationships between the
typetates of objects can be expressed as subset and disjointness
relations on sets, and elements of sets can be represented as sets
of cardinality one. In addition, sets with cardinality constraints
provide a natural language for specifying operations and invariants
of data structures.

Motivated by these program analysis applications, this paper
presents new algorithms and new complexity results for constraints
on sets and their cardinalities. We study several classes of con-
straints and demonstrate a trade-off between their expressive power
and their complexity.

Our first result concerns a quantifier-free fragment of Boolean
Algebra with Presburger Arithmetic. We give a nondeterministic
polynomial-time algorithm for reducing the satisfiability of sets
with symbolic cardinalities to constraints on constant cardinalities,
and give a polynomial-space algorithm for the resulting problem.

The best previously existing algorithm runs in exponential space
and nondeterministic exponential time.

In a quest for more efficient fragments, we identify several
subclasses of sets with cardinality constraints whose satisfiability
is NP-hard. Finally, we identify a class of constraints that has
polynomial-time satisfiability and entailment problems and can
serve as a foundation for efficient program analysis. We give a sys-
tem of rewriting rules for enforcing certain consistency properties
of these constraints and show how to extract complete information
from constraints in normal form. This result implies the soundness
and completeness of our algorithms.

1. Introduction

Program analyses that reason about deep semantic properties are of
great value for software development; the value of such analyses
is growing with the adoption of language constructs that eliminate
low-level program errors. Many deep semantic properties are natu-

rally expressible in fragments of set theory, so constraint solving for
such fragments is of interest. This paper presents new algorithms
and improved complexity bounds for fragments of set theory. The
starting point of our constraints is the boolean algebra of finite (but
unbounded) sets.

Sets in program analysis. The boolean algebra of finite sets
is a fragment of set theory that allows the basic set operations
of intersection, union, and complement on sets of uninterpreted
elements. Although simple, it turns out that this fragment can
express many properties of interest in program analysis. Examples
include typestate properties and public interfaces of data structures.

Set specifications generalize typestate properties [29, 26]: the
fact that an object o is in the typestate t is represented as the set
membership of o in t. Through inclusion and disjointness con-
straints, sets can also express relationships (such as hierarchy or
orthogonality) between different typetates. Objects can be rep-
resented as sets of cardinality one using a cardinality constraint
|\{o\}| = 1, so set membership reduces to subset. Multiple set member-
ships can then encode constraints such as |t| \geq k for any constant
k.

Sets can also provide natural abstractions of container data
structures. When a content of a data structure is represented as an
abstract set s, an operation such as insertion can be characterized
by a postcondition s' = s \cup e where e is the set corresponding
to the element being inserted. By expressing both typetates and
data structure abstractions, sets can be used to combine the results
of different analyses operating on the same program. Such an
approach allows us to combine the scalability of typestate analysis
with the precision of shape analysis and theorem proving [30, 28,
27, 48].

Sets with cardinality constraints. The use of the cardinality op-
erator on sets leads to a connection between set algebra opera-
tions and integer linear arithmetic, as evidenced, for example, in
the condition |a \cup b| = |a| + |b| for disjoint sets a and b. It is
therefore natural to consider constraints that combine integer linear
arithmetic with set algebra operations. These constraints constitute
the Quantifier-Free Boolean Algebra with Presburger Arithmetic,
or QFBA for short — they are the quantifier-free fragment of
BAPA constraints whose decision procedure and complexity we
have studied in [23] [22]. QFBA constraints can be used to verify
an invariant such as |a| = |b| which allows us to conclude that
if a is nonempty, so is b, and therefore it is possible to call an op-
eration that removes an element from b. Similarly, if i is an integer
variable and s is a set, it is possible to verify an invariant |s| = i
stating that an integer i correctly maintains the size of the set s.

In our experience, specialized decision procedures such as [22]
are the only automated technique for deciding with non-trivial card-
nality constraints. Currently, however, the complexity of these de-
cision procedures limits their applicability. In this paper we give
new algorithms for solving set cardinality constraints; these algo-
rithms provide exponential improvements over existing approaches
and make the checking of cardinality constraints in larger formulas
more feasible.

Our paper provides a systematic study of constraints on sets in
the presence of cardinalities. We study both more expressive and
less expressive fragments and demonstrate a trade-off between the
expressive power and the efficiency of the algorithms. The main contributions of our paper are the following:

- **PSPACE algorithm for QFBA P.** The best previously known algorithms for QFBA [23, 22, 45] execute in non-deterministic exponential time, and involve searching for an exponentially large object. In this paper we first give a form of satisfiability and subsumption of constraints in this class is the simplest constraint be solved in deterministic exponential time, and involve searching for an exponentially large object. We show that this is indeed the case. Namely, we first show that Presburger arithmetic formulas generated by the algorithm from [22] can in fact be solved in deterministic exponential time. Our result reduces QFBA to a simpler system of CBAC constraints (shown in Figure 3), then applies a theorem by Papadimitriou [36] in a novel way. This leads to a deterministic EXPTIME decision procedure for QFBA satisfiability, which is an improvement on previously existing algorithms. Nevertheless, the question arises whether it is possible to avoid the construction of a non-deterministically large system of equations. It turns out that this is indeed possible: we present an alternating polynomial-time (and therefore, PSPACE) algorithm for QFBA. Therefore, it is possible to solve QFBA using solvers for quantifier-free formulas, so it is natural to examine whether more efficient algorithms exist for QFBA constraints. When applied to QFBA formulas, existing algorithms run in non-deterministic exponential time (NEXPTIME): the algorithm [45] requires nondeterministically guessing an exponentially large object, whereas the algorithm from [22] produces an exponentially large quantifier-free Presburger arithmetic formula. The question arises whether there exist algorithms that avoid non-deterministically guessing exponentially large objects. We show that this is indeed the case. Namely, we first show that Presburger arithmetic formulas generated by the algorithm from [22] can in fact be solved in deterministic exponential time. Our result reduces QFBA to a simpler system of CBAC constraints (shown in Figure 3), then applies a theorem by Papadimitriou [36] in a novel way. This leads to a deterministic EXPTIME decision procedure for QFBA satisfiability, which is an improvement on previously existing algorithms. Nevertheless, the question arises whether it is possible to avoid the construction of a non-deterministically large system of equations. It turns out that this is indeed possible: we present an alternating polynomial-time (and therefore, PSPACE) algorithm for QFBA. Therefore, it is possible to solve QFBA using solvers for quantifier-free formulas [9, 48, 37].

- **A Polynomial-Time Class.** Given that QFBA constraints are NP-hard, the question remains whether there are interesting fragments of sets with cardinalities which can be reasoned about in polynomial time. In a quest for such fragments, we identify several features of constraints, each of which leads to NP-hardness. By eliminating these features we have discovered a class (called i-trees) that has a polynomial-time satisfiability and entailment (subsumption) problems, while still supporting subset, union, disjointness, and arbitrarily large cardinality constraints. This class can therefore express generalized typestate constraints such as multiple orthogonal classifications into independent or disjoint sets. The identification of this polynomial-time class, and the development of algorithms for testing the satisfiability and subsumption of constraints in this class is the second contribution of this paper. While the resulting algorithms are efficient, the proof of their completeness is somewhat lengthy, and involves characterizations of normal forms of i-trees and the construction of models for i-trees in normal form. We therefore only summarize the main ideas; we refer the reader to the full version of the paper [22] for details. Additional proofs are also included in the Appendix.

We proceed by defining the fragment QFBA in Section 2. We present a PSPACE algorithm for QFBA in Section 3, defining the simpler CBAC constraints and identifying their NP-complete fragment, CBASC constraints.

### 2. Constraints on Sets with Cardinalities

#### Boolean Algebra with Presburger Arithmetic

- **Complement.** The complements, encoded in binary: a constant $k$ is encoded using $O(\log k)$ bits. The symbol $A$ in Figure 1 denotes atomic formulas; a literal is an atomic formula or its negation.

A quantified version of this language (BAPA) is studied in [23], where we give an algorithm that establishes a doubly exponential space upper bound on the complexity. Because quantified BAPA subsumes Presburger arithmetic, the doubly exponential nondeterministic time lower bound [15] applies to BAPA as well.

#### Preliminaries.

If $S$ is a finite set, $|S|$ denotes the number of elements in $S$. A literal is an atomic formula or its negation. $Z = \{0, 1, \ldots \}$ is the set of integers, $N = \{0, 1, \ldots \}$ is the set of natural numbers. $[a, b]$ denotes the set of integers $\{a, a + 1, \ldots, b\}$. If $f : A \rightarrow B$ is a function and $S \subseteq A$, we define $f[S] = \{f(a) \mid a \in S\}$.

If $A$ is a set, the notation $A^n$ has several potential meanings: the specific meaning should be clear from the context. $A^n$ for $n = \{1, 2, \ldots \}$ is the set of vectors $(a_1, \ldots, a_n)$ where $a_j \in A$ for $1 \leq j \leq n$, and $A^{m \times n}$ is the set of matrices $[a_{pq}]$ with $m$ rows and $n$ columns with elements $a_{pq}$ for $1 \leq p \leq m$ and $1 \leq q \leq n$. The expression $A^n$ denotes the complement of the set $A$. If $\alpha \in \{0, 1\}$, then $A^{\alpha}$ denotes $A$ for $\alpha = 1$ and $A^\alpha$ for $\alpha = 0$.

The relation $\equiv$ denotes the equality of the values of metavariables denoting syntactic objects, so if $f_1$ and $f_2$ are formulas, then $f_1 \equiv f_2$ means that they are the same formula. In the context of inclusion diagrams (Section 4), $\equiv$ will denote the semantic equivalence of diagrams (we use $\simeq$ to denote the equality of diagrams).

#### 3. A PSPACE Algorithm for QFBA

Verification conditions arising in program verification can often be expressed using quantifier-free formulas, so it is natural to examine whether more efficient algorithms exist for QFBA constraints. When applied to QFBA formulas, existing algorithms run in non-deterministic exponential time (NEXPTIME): the algorithm [45] requires nondeterministically guessing an exponentially large object, whereas the algorithm from [22] produces an exponentially large quantifier-free Presburger arithmetic formula. The question arises whether there exist algorithms that avoid nondeterministically guessing exponentially large objects. We show that this is indeed the case. Namely, we first show that Presburger arithmetic formulas generated by the algorithm from [22] can in fact be solved in deterministic exponential time. Our result reduces QFBA to a simpler system of CBAC constraints (shown in Figure 3), then applies a theorem by Papadimitriou [36] in a novel way. This leads to a deterministic EXPTIME decision procedure for QFBA satisfiability, which is an improvement on previously existing algorithms. Nevertheless, the question arises whether it is possible to avoid the construction of a non-deterministically large system of equations. It turns out that this is indeed possible: we present an alternating polynomial-time (and therefore, PSPACE) algorithm for QFBA. Therefore, it is possible to solve QFBA using solvers for quantifier-free formulas [9, 48, 37].

Figures 2 and 3 present our PSPACE algorithm for QFBA. The algorithm has two phases.

In the first phase, the non-deterministic polynomial-time algorithm in Figure 2 reduces QFBA constraints to a simpler class of constraints. We call these simpler constraints Conjunctions of Boolean Algebra expressions with Cardinalities (CBAC). CBAC constraints have a very simple syntactic structure (see Figure 4), but capture the key difficulty in solving QFBA: the need to consider exponentially large cardinalities on exponentially many set partitions.

In the second phase, the algorithm in Figure 3 checks the satisfiability of CBAC in alternating polynomial time and therefore in polynomial space. The key insight behind our algorithm is that it is possible to use a divide and conquer approach to avoid explicitly representing all possible regions in the Venn diagram.

---

**Figure 1.** Quantifier-Free Formulas of Boolean Algebra with Presburger Arithmetic (QFBA)
Let \( f \) be the input QFBAPA formula.

1. Replace each \( \mathbb{Z} \)-variable with a difference of two \( \mathbb{N} \)-variables:
   \[
   C[i_1, \ldots, i_n] \rightarrow C[i'_1 - i''_1, \ldots, i'_n - i''_n]
   \]
   \( i'_1, i''_1, \ldots, i'_n, i''_n \) are fresh \( \mathbb{N} \)-variables

2. Ensure that all set algebra expressions appear within cardinality constraints by normalizing with the following rules:
   \[
   C[b_1 = b_2] \rightarrow C[b_1 \subseteq b_2 \land b_2 \subseteq b_1]
   \]
   \( C[b_1 \subseteq b_2] \rightarrow C[b_1 \land b_2] = 0 \)

3. Eliminate divisibility constraints:
   \[
   C[k \cdot d \cdot t] \rightarrow C[k = t], \text{ } i \text{ is fresh } \mathbb{N}-variable
   \]

4. Move all cardinality constraints to top level:
   \[
   C[[b_1, \ldots, b_m] \rightarrow f_1 \land f_2
   \]
   where \( f_1 \) is \( \equiv C[i_1, \ldots, i_w] \)
   \[
   f_2 \equiv (1) = \text{MAXC} \land \bigwedge_{j=1}^{w} b_j = i_j
   \]
   and \( i_1, \ldots, i_w \) are fresh \( \mathbb{N} \)-variables; let \( m = w + 1 \).

5. Let \( p \) be a propositional formula such that \( p(a_1, \ldots, a_m) \equiv 1 \) for atomic formulas \( a_1, \ldots, a_m \).
   Non-deterministically select the truth value \( \alpha \in \{0, 1\} \) for each atomic formula \( a_j \), so that \( p(a_1, \ldots, a_m) \) is true. Let \( \alpha \)
   \[
   f_{11} \triangleq \bigwedge_{j=1}^{m_0} \alpha_j
   \]
   and \( f_{11} \) is \( \equiv C[i_1, \ldots, i_w] \)
   \[
   f_{12} \equiv (1) = \text{MAXC} \land \bigwedge_{j=1}^{w} b_j = i_j
   \]
   and \( i_1, \ldots, i_w \) are fresh \( \mathbb{N} \)-variables; let \( m_0 = w + 1 \).

6. For each conjunct \( \neg(t_1 = t_2) \) in \( f_{11} \) non-deterministically replace the conjunct with one of the conjuncts \( (t_1 + 1 \leq t_2) \) or \( (t_1 + 1 \leq t_1) \).

7. Transform linear integer constraints to normal form:
   \[
   C[\neg(t_1 \leq t_2)] \rightarrow C[t_2 + 1 \leq t_1]
   \]
   \[
   C[t_1 \leq t_2] \rightarrow C[t_1 - t_2 + i = 0]
   \]
   \[
   C[t_1 = t_2] \rightarrow C[\sum_{j=1}^{m} c_{ij} = k]
   \]

8. Let \( n_0 \) be the number of \( \mathbb{N} \)-variables in the entire formula. The resulting system is of the form of:
   \[
   Ax + d \land \bigwedge_{j=1}^{m_0} b_j = i_p
   \]
   where \( A \in \mathbb{Z}_p^{n_0 \times m_0}, d \in \mathbb{Z}_m, v = (i_1, \ldots, i_{m_0}) \)
   where each \( i_j \) is an \( \mathbb{N} \)-variable and \( 1 \leq p, \ldots, p_{m_0} \leq m_0 \) are variables denoting cardinalities of sets. Let \( S \) be the total number of set variables in \( b_1, \ldots, b_m \).
   Let \( m = m_0 + m_1, n = \max(n_0, 2^S) \).
   \[
   a = \max\{(1) \cup \{a_{pq} \mid 1 \leq p \leq n_0, 1 \leq q \leq m_0\}
   \]
   \[
   \cup \{a_j \mid 1 \leq j \leq m_0\}
   \]
   where \( A = [a_{pq}]_{p,q}, d = (d_1, \ldots, d_{m_0}), \) and let \( M = n(a)2^{m_0+1} \).

9. Non-deterministically select a vector \( k = (k_1, \ldots, k_{m_0}) \)
   where \( k_j \in \{0, 1, \ldots, M\} \) for \( 1 \leq j \leq n_0 \), such that \( Ak = d \).

10. Call CBAC decision procedure on \( \bigwedge_{j=1}^{m_0} b_j = k_p \).

\section*{Figure 2. An NP Algorithm for Reducing QFBAPA Constraints to CBAC Constraints of Figure 3.}

We next discuss our algorithm in more detail and argue that it is correct. We begin with the description of the steps of the algorithm in Figure 2 which reduces symbolic cardinalities to large constant cardinalities.

1. \textbf{Non-negative integers.} To simplify the later steps, the first step makes all integer variables range over non-negative integers \( \mathbb{N} \), by replacing each integer variable \( i \) with a difference \( i_1 - i_2 \) of fresh non-negative integer variables \( i_1, i_2 \).

2.3. \textbf{Eliminating set equality and subset, and integer divisibility.} The next step converts set equality and subset into cardinality constraints. This step helps the later separation between the boolean algebra part and the integer linear arithmetic part. We then eliminate any divisibility relations using multiplication and a fresh variable.

4. \textbf{Flattening.} The next step separates the formula into the boolean algebra part, denoted \( f_1 \) and the integer linear arithmetic part, denoted \( f_2 \). This step simply amounts to naming the cardinality of each set by a fresh integer variable.

5.6. \textbf{From quantifier-free formulas to conjunctions.} An obvious source of NP-completeness of QFBAPA is the presence of arbitrary propositional combinations of atomic formulas. An effective way of dealing with propositional combinations is to enumerate the
satisfying assignments of the propositional formula using a SAT solver, and then solve the conjunctions of literals \[16\] steps 5 and 6 of the non-deterministic algorithm in Figure 4 are an abstract description of such procedure. The goal of step 6 is to eliminate disequations, which involve non-deterministic choice between the two inequalities.

7. Normal form for integer constraints. The algorithm eliminates the remaining negations of atomic formulas and transforms linear constraints into normal form \[ Av = d \].

8. Normal form for integer constraints. The algorithm eliminates the remaining negations of atomic formulas and transforms linear constraints into normal form \[ Av = d \].

9. Normal form for integer variables. The resulting system contains linear integer equations of the form \( \sum_{j=1}^{n} c_j t_j = k \), and set cardinality constraints of the form \( |b| = i \). The algorithm computes an upper bound \( M \) on integer variables in any potential solution of the system, using several parameters: the number of conjuncts \( n \), the number of integer variables \( n_0 \) and the number of set variables \( S \). The computation of the upper bound is based on an observation that the satisfiability of the conjunction of constraints \( |b| = i \) can be reduced to the satisfiability of equations of the form \( \sum_{j=1}^{n} t_j = i \), where variables \( t_j \) denote sizes of set partitions (regions in Venn diagram) whose union is the set \( b \); this is a specialization of the idea in \[16\] to the case of quantifier-free formulas.

Let \( s_1, \ldots, s_k \) be all set variables appearing in formula and consider a constraint \( |b| = i \). Consider all partitions \( \prod_{j=1}^{n} s_j^{a_j} \) for \( a_j \in \{0, 1\} \). For each such partition \( p \), introduce a fresh \( \Pi \)-variable \( \hat{t}_p \), which denotes the cardinality of cube \( p \). Then consider a constraint of the form \( |b| = i \). Each set is a union of regions in the Venn diagram (by the disjunctive normal form theorem) so suppose that \( b = b_{p_1} \cup \ldots \cup b_{p_k} \). Then replace the term \( |b| = i \) with \( \sum_{p=1}^{k} \hat{t}_p = i \). We use the term “CBAC linear equations” to denote a system of linear equations resulting from the constraints \( |b| = i \) as described above.

As a result, we obtain a system of \( m_0 + m_1 \) linear equations over non-negative integers, where \( m_0 \) equations have a polynomial number of variables, and \( m_1 \) equations (CBAC linear equations) have exponentially many variables. It is easy to see that there exists a surjective mapping of solutions of the original constraints on sets onto solutions of the resulting linear equations (the mapping computes the cardinality of each Venn diagram). Therefore, the original system is satisfiable if and only if the resulting equations are satisfiable. Moreover, we have the following fact.

**Fact 1 (Papadimitriou [36]).** Let \( A \) be an \( m \times n \) integer matrix and \( b \) an \( m \)-vector, both with entries from \( \{-a, a\} \). Then the system \( Ax = b \) has a solution in \( \mathbb{N}^n \) if and only if it has a solution in \( \{0, M\}^n \) where \( M = n(ma)^{m+1} \).

Fact 1 implies that the estimate \( M \) computed in step 8 of the algorithm in Figure 4 is a correct upper bound. Using this estimate, step 9 of the algorithm non-deterministically guesses the values of all integer variables such that the original linear equations \( Ax = d \) are satisfied. All this computation can be performed in nondeterministic polynomial time, and (unlike \[22\]), does not involve constructing explicitly a system with exponentially many equations. Having picked the values of integer variables, including the variables \( i \) on the right hand side of constraints \( |b| = i \), we obtain a conjunction of constraints of the form \( |b| = k \) where \( k \) is a constant whose binary representation has polynomially many bits—these are precisely the CBAC constraints in Figure 3. We have therefore shown the following.

**Lemma 1.** The algorithm in Figure 3 reduces in nondeterministic polynomial time the satisfiability of a QFBAPA formula to the satisfiability of CBAC formulas.

It remains to find an algorithm for CBAC constraints.

**A PSPACE algorithm for CBAC.** One correct way to solve CBAC constraints is to solve the associated CBAC linear equations. This system has exponentially many variables, each of which can take any value from \([0, M]\). Therefore, guessing the values of each of these variables can be done in non-deterministic exponential time; similar approaches not based on equations also require guessing exponentially large objects \[15\]. Note, however, that there are only polynomially many CBAC linear equations. Using the idea of the proof \[16\] corollary 1, we can therefore show that a dynamic programming algorithm can be used to solve the system in polynomial time. In fact, we can use the dynamic programming algorithm from the proof of \[16\] corollary 1. Instead of fixing the size of the equations \( m_1 \) to be constant, we simply observe that \( m_1 \) is polynomial in the size of the input, whereas the number of variables is singly exponential. The bound \( M \) therefore yields a singly exponential deterministic time dynamic programming algorithm for CBAC. While this is better than existing results, we show that an even better result is achievable.

Clearly, any algorithm that explicitly constructs CBAC equations will require at least exponential time and space. Our solution is therefore to adapt the dynamic programming algorithm to a divide and conquer approach that always represents the equations in terms of their original, polynomially sized, boolean algebra expression. Such an algorithm runs in alternating polynomial time, consuming polynomial space, and is presented in Figure 4. To see the idea of our PSPACE algorithm, consider the CBAC linear system of equations written in the vector form: \( \sum_{j=1}^{n} a_j l_j = d \), where \( a_j \) are vectors and \( l_j \) are the variables for \( 1 \leq j \leq 2^m \). The algorithm guesses the vectors \( d_0, d_1 \in \mathbb{N}^m \) such that \( d_0 + d_1 = d \), and recursively solves two equations:

\[
\sum_{j=1}^{2^m-1} a_j l_j = d_0 \quad \land \quad \sum_{j=2^m}^{2^{2^m}-1} a_j l_j = d_1
\]

This algorithm creates an OR-AND tree whose leaves gives the answer to the original problem. A position in the tree is given by the propositional assignment \( v_1, \ldots, v_s \) to boolean variables. Each edge in the tree is given by a complete assignment \( v_1, \ldots, v_s \) to set variables. Note that we never need to explicitly maintain the system during the divide phase of the algorithm, it suffices to determine in the leaf case \( p = 0 \) whether the coefficient \( a_j \) is 0 or 1. The algorithm does this by simply evaluating each boolean algebra expression \( b \) for the assignment \( v_1, \ldots, v_s \).

**Theorem 1.** The algorithm in Figure 4 checks the satisfiability of CBAC constraints in PSPACE. The algorithm given by Figures 4 and 1 checks the satisfiability of QFBAPA constraints in PSPACE. Theorem 1 improves the existing algorithms for QFBAPA from both a complexity theoretic and an implementation viewpoint. A deterministic realization of previous NEXPTIME algorithms runs in doubly exponential worst-case time and requires exponential space; a deterministic realization of our new algorithm runs in singly exponential time and consumes polynomial space. Previous algorithms would require running a constraint solver such as a SAT solver \[47\] on an exponentially large constraint; the new algorithm can be solved by running a quantified boolean algebra solver \[48\] on a polynomially large constraint.

**NP fragments of CBAC.** We have seen that both CBAC and QFBAPA constraints are in PSPACE. Both of these classes of constraints are NP-hard, because the constraint \( |b| = 1 \) is satisfiable iff \( b \) corresponds to a satisfiable propositional formula. Moreover, Lemma 1 shows that QFBAPA constraints are in NP iff CBAC constraints are in NP. For some subclasses of CBAC constraints we can indeed show membership in NP. Define conjunctions of boolean algebra expressions with small cardinalities, denoted CBASC, to be the same as CBAC but with constant integers encoded in unary.
notation, where an integer \( x \) is represented in space \( O(x) \); such encoding can therefore be exponentially less compact.

**Lemma 2.** The satisfiability of CBASC constraints is NP-complete.

CBASC solutions are NP-hard because \(|b| = 1 \) is a CBASC constraint. One way to prove membership in NP is to observe that CBASC is subsumed by the language of set-valued fields which was proven to be in NP [24, 25] by reduction to the universal class of first-order logic formulas, which has the small model property [7] Page 258]. Another way is to consider the notion of sparse solutions of CBAC arc linear equations. An \( M \)-sparse solution is a solution to CBAC linear constraints with at most \( M \) non-zero elements. An \( M \)-sparse solution to CBAC linear constraints with \( 2^S \) variables can be encoded as an \( M \)-tuple of pairs \([v_1, \ldots, v_S] \), where each \( v_s \) encodes one of the \( 2^S \) integer variables, and \( k \) specifies the value of that integer variable. This encoding is polynomial in \( M \) when \( w \) is the number of bits for representing the largest component of the solution. For any CBAC linear constraint \( A^w_{j=1} b_j = k_j \), each solution is \( M \)-sparse where \( M = \max(k_1, \ldots, k_m) \). For CBASC solutions, \( M \) is polynomial in the size of the CBASC representation because each \( k_s \) is encoded in unary, so sparse solutions can be guessed in polynomial time. This proves that CBASC constraints are in NP.\(^1\)

## 4. Inclusion Diagrams

This section introduces inclusion diagrams (i-diagrams), a graph representation of CBASC constraints. Figure 5 shows a formula with \( \{s_1\} \cup \{s_2\} \), where each view is a nonempty set of sons that is a partition of \( \Phi \). Splitting sets \( \{s_1\} \) and \( \{s_2\} \) is a set of nodes that represent pairs disjoint sets, and \( \text{Comp}(S) \) is a set of complete views each of which is a set of nodes that represent sets whose union is equal to the father; we require

- \( \bigcup \text{Split}(S) = \text{Sons}(S) \)
- \( \bigcup \text{Comp}(S) \subseteq \text{Sons}(S) \)

\( D \) is such that \( \text{ClInf}(\{s_1\}) = \{s_1\}, \text{CSup}(\{s_1\}) = \{s_2\} \), and \( \text{Comp}(\{s_1\}) = \{s_1, s_2\} \).

**Definition 2** (Semantics of i-diagrams). An interpretation of \( \text{SN} \) and \( \text{PN} \) is a triple \((\Delta, \alpha, \Xi)\) where

- \( \Delta \) is a finite set (the universe); \( \alpha : \text{SN} \rightarrow \text{P}(\Delta) \) specifies the values of sets; \( \Xi : \text{PN} \rightarrow \text{P}(\Delta) \) specifies the values of unary predicates.

An interpretation \( I \) is a model for an i-diagram \( D \), denoted \( I \models D \), iff \( \forall S \in \text{Comp}(S) \), \( \forall s \in \text{Sons}(S) \).

### An example i-diagram and an equivalent formula

In a graphical representation of an i-diagram, we represent each element \( S \in \Delta \) where \( S = \{s_1, \ldots, s_n\} \) and \( s \) by an arrow leading to \( s \) in \( S \) and an arrow leading to \( S \) in \( S \). We represent inclusion \( S_1 \subseteq S_2 \) by an arrow from \( S_1 \) to \( S_2 \). We represent a split \( Q \in \text{Split}(S) \) by a circle connected with undirected edges to \( S_1, \ldots, S_n \) and an arrow leading to \( S \). We represent a complete view similarly, using a filled square instead of a circle. For each node \( S \in \Delta \) we indicate its cardinality bounds by annotating the node with \( [a..b] \) where \( a = \text{ClInf}(S), b = \text{CSup}(S) \). We represent \( \Xi(S) = \{\pm P_1, \ldots, \pm P_n\} \) by annotating \( S \) with \( \pm P_1, \ldots, \pm P_n \). We represent \( \emptyset = \{s_1, \ldots, s_n\} \) by annotating the node \( \{s_1, \ldots, s_n\} \) with \( \emptyset \).

\(^1\) Sparse solutions are interesting for general CBASC constraints as well. As of yet we have no example of a CBASC constraint whose associated CBASC equation system is satisfiable but has no sparse solutions; moreover, we can generalize the notion of sparse solutions to solutions representable using binary decision diagrams [5] while preserving polynomial-time verifiability.
\[ \alpha(s_1) = \ldots = \alpha(s_n) \]

accordingly, define \( \overrightarrow{\alpha}(S) \) as follows:

\[ \text{Clnf}(S) \leq \overrightarrow{\text{Clnf}}(S) \leq \overrightarrow{\text{CSup}}(S) \]

\[ \forall P. ((+P) \in \Phi(S) \Rightarrow \overrightarrow{\text{Clnf}}(S) \subseteq \overrightarrow{\text{CSup}}(P)) \]

\[ \forall P. (-P) \in \Phi(S) \Rightarrow \overrightarrow{\text{Clnf}}(S) \subseteq \overrightarrow{\text{CSup}}(P) \]

\[ \forall \overrightarrow{\text{Sons}}(S), \overrightarrow{\text{Clnf}}(S) \subseteq \overrightarrow{\text{CSup}}(S) \]

\[ \forall Q \in \text{Split}(S). \forall S_1, S_2 \in Q. S_1 \neq S_2 \Rightarrow \overrightarrow{\text{Clnf}}(S_1) \cap \overrightarrow{\text{Clnf}}(S_2) = \emptyset \]

\[ \forall Q \in \text{Comp}(S). \forall S_1 \in Q. S_1 \subseteq \overrightarrow{\text{Clnf}}(S) \]

We use the standard notions of satisfiability, subsumption (entailment), and equivalence:

\[ \begin{align*}
D \text{ is satisfiable} & \iff \exists I. I \models D \\
D' \models D & \iff \forall I. I \models D' \Rightarrow I \models D \\
D' \equiv D & \iff D' \models D \land D \models D' 
\end{align*} \]

**Definition 3 (Explicit Disjointness).**

We write \( \text{disj}_{D,0}(S_1, S_2) \) as a shorthand for

\[ S_1 \neq S_2 \land \exists Q \in \text{Sons}(S_1), S_1, S_2 \in Q \]

and we say that \( S_1, S_2 \) are explicitly disjoint, and we write \( \overrightarrow{\text{disj}}_{D,0}(S_1, S_2) \) iff

\[ \exists S_1', S_2', S_3, S_4 \in S, S_1 \supseteq S_1', S_2 \supseteq S_2' \land \text{disj}_{D,0}(S_1', S_2') \]

**Lemma 3.** I-diagrams have the same expressive power as CBAC constraints.

By “same expressive power” we here mean that there is a natural pair of mappings between the models of i-diagrams and solutions to CBAC constraints.

We can define the following operations:

**Definition 4 (Factor-i-diagram).** Let \( p \subseteq S \times S \) be an equivalence relation on nodes. We define \( D/p \) as follows. Define \( \perp_i = \perp \). Let \( D = (S, \theta_d, \text{Sons}, \text{Split}, \text{Comp}, \text{Clnf}, \text{CSup}, \Phi) \). We define \( D/p \) as \( D' = (S', \theta_d', \text{Sons}', \text{Split}', \text{Comp}', \text{Clnf}', \text{CSup}', \Phi') \) as follows. Define \( h \) so that if \( \{S_1, \ldots, S_n\} \) is the equivalence class of \( S \) under \( p \), then \( h(S) = S \cup \ldots \cup S_n \). If \( S \subseteq S \), define \( h(Q) = h(S) \mid S \in Q \). Let then \( S' = h[S] \). Consider \( S' \subseteq S' \). Both \( S \) and \( S' \) are partitions, and given \( S' \subseteq S' \) there is a unique set \( \{S_1, \ldots, S_n\} \subseteq S \) such that \( S' = S_1 \cup \ldots \cup S_n \). Then define:

\[ \begin{align*}
\text{Clnf}'(S') & = \max(\text{Clnf}(S_1'), \ldots, \text{Clnf}(S_n)) \\
\text{CSup}'(S') & = \min(\text{CSup}(S_1'), \ldots, \text{CSup}(S_n)) \\
\text{Sons}'(S) & = \{h[S_1] \mid S_1 \subseteq S \} \\
\text{Phi}'(S) & = \Phi(S_1) \cup \ldots \cup \Phi(S_n) \\
\text{Split}'(S') & = \{h(Q) \mid Q \in \text{Split}(S_1) \cup \ldots \cup \text{Split}(S_n)\} \\
\text{Comp}'(S') & = \{h(Q) \mid Q \in \text{Comp}(S_1) \cup \ldots \cup \text{Comp}(S_n)\}
\end{align*} \]

**Definition 5 (Merge).** For any i-diagram \( D \) we define the i-diagram \( D'[\text{Merge}(Q)] \equiv D/p \) for the equivalence relation \( p = \{\{S_1, S_2\} \mid S_1, S_2 \in Q \} \cup \{\{S, S\} \mid S \in S\} \)

In the sequel we impose the following restrictions on the form of i-diagrams:

**Definition 6 (Simple Diags).** A diagram is \( D \) is simple if

\[ D = \theta_d \text{ or all of the following conditions hold for all } S \subseteq S: \]

\[ a) \ (S, \ell) \text{ has no cycles, in particular } S \notin \text{Sons}(S) \]

\[ b) \ \theta_d \notin \text{Sons}(S) \]

\[ c) \ 0 \neq \text{Split}(S) \land 0 \notin \text{Comp}(S) \]

\[ d) \ \forall Q, Q'. Q \in \text{Split}(S) \land Q' \subseteq Q \Rightarrow Q' \notin \text{Split}(S) \]

\[ e) \ \forall Q, Q'. Q \in \text{Comp}(S) \land Q' \subseteq Q \Rightarrow Q' \notin \text{Comp}(S) \]

\[ f) \ \text{CSup}(\theta_d) = 0, \text{Sons}(\theta_d) = \Phi(\theta_d) = 0 \]

**proc Simplify(D) :**

1. use fixpoint iteration to compute \( \rho \) as the smallest equivalence relation such that:

\[ a) \ (S, \ell_0) \in \rho \land S \sim S_1 \Rightarrow (S, \ell_0) \in \rho \]

\[ b) \ (S, \ell_0) \in \rho \land S \sim S \Rightarrow (S, \ell_0) \in \rho \]

\[ c) \ 0 \notin \text{Comp}(S) \Rightarrow (S, \ell_0) \in \rho \]

\[ d) \ \text{disj}_{D,0}(S_1, S_2) \in \rho \Rightarrow (S_2, \ell_0) \notin \rho \]

\[ e) \ \text{disj}_{D,0}(S_1, S_2) \in \rho \Rightarrow (S_2, \ell_0) \notin \rho \]

\[ f) \ \text{Comp}(\ell_0) = 0 \Rightarrow (S, \ell_0) \in \rho \]

\[ g) \ \text{Split}(\ell_0) = 0 \Rightarrow (S, \ell_0) \in \rho \]

\[ h) \ \text{CSup}(\ell_0) = 0 \Rightarrow (S, \ell_0) \in \rho \]

2. \( D := D/\rho \)

3. \( \text{Comp}(S) = \{Q \mid Q \notin D \cap \text{Split}(S), S \notin D\} \)

4. \( \text{Sons}(S) = \{Q \mid Q \notin \text{Comp}(S), S \notin D\} \)

5. \( \text{Split}(S) \subseteq \text{Split}(S) - \{0\} \)

6. return \( D \)

Where \([a \leftarrow b]\) denotes the result of updating the component \( a \) of i-diagram \( D \) with value \( b \).

**Figure 6.** Polynomial-time algorithm Simplify to compute an equivalent simple i-diagram

Simplicity eliminates redundancy from diagrams, but does not restrict their expressive power, as the following lemma shows.

**Lemma 4.** For every i-diagram \( D \) we can obtain an equivalent simple i-diagram using the polynomial-time algorithm Simplify in Figure 4

**5. Sources of NP Hardness and Definition of I-Trees**

The satisfiability of i-diagrams is NP-hard because i-diagrams have the same expressive power as CBAC constraints. We have observed that the general directed acyclic graph structure of i-diagrams allows us to encode NP-complete problems; this motivates the following two restrictions.

**Definition 7.**

An i-diagram \( D \) is tree shaped iff \( (S, \ell) \) is a tree (with an additional isolated node \( \ell_0 \)).

An i-diagram \( D \) has independent views iff for all \( Q_1, Q_2 \in \text{Split}(S) \cup \text{Comp}(S) \) at least one of the following two conditions holds:

\[ a) \ Q_1 \cap Q_2 = \emptyset \]

\[ b) \ Q_1 \notin \text{Split}(S) \land Q_2 \in \text{Comp}(S) \land Q_1 \subseteq Q_2 \]

Recall that, by Lemma 4, it suffices to consider i-diagrams with acyclic graphs of the subset relation. The tree shape condition is then a natural next restriction on the structure of i-diagrams. However, due to the presence of Split and Comp, the tree shape condition by itself does not reduce the expressive power of i-diagrams, and further restrictions are necessary. The independent views con-
dion extends the tree condition to the entire graphical representation of i-diagrams, including the circles and squares that represent Split and Comp views. The conjunction of these two conditions can be expressed by saying that the graphical representation of i-diagram is a tree.

**Remark 1.** We can express the combination of the conditions: being simple, being tree shaped, and having independent views by saying that there are only four kinds of edges in the corresponding graphical representation:

- from an element \( S \in S - \{\emptyset\} \) to a circle
- from a circle to a square, indicating that all nodes of a split view belong to a complete view
- from a circle to an element \( S \in S - \{\emptyset\} \)
- from a square to an element \( S \in S - \{\emptyset\} \).

Unfortunately, the restrictions on tree shape and independent views are not sufficient to guarantee a polynomial-time decision procedure in the presence of predicates associated with nodes. The reason is that the ability to encode disjointness of arbitrary sets leads to NP-hardness, yet even with tree structure and independent views it is possible to assert that two arbitrary sets \( S_1 \) and \( S_2 \) are disjoint by letting \((+P)\in \Phi(S_1)\) and \((-P)\in \Phi(S_2)\) for some uninterpreted predicate \( P \). A simple way to avoid this problem is to require that \( \Phi \) contains only positive atoms \((+P)\). A more flexible restriction is the following.

**Definition 8.** An i-diagram \( D \) has independent signatures iff for every pair of distinct nodes \( S_1, S_2 \) such that \((-P)\in \Phi(S_1)\) and \((+P)\in \Phi(S_2)\) for some \( P \in \text{PN} \), at least one of the following two conditions holds:

1. \( S_1 \) and \( S_2 \) are explicitly disjoint, that is, \( \text{disj}^*_P(S_1, S_2) \)
2. \( S_1 \) and \( S_2 \) have compatible signatures, that is, there exists a node \( S \) such that

\[
S_1 \cong S \land S_2 \cong S \land \text{Sig}(S_1) \cap \text{Sig}(S_2) \subseteq \text{Sig}(S)
\]

where \( \text{Sig}(S) = \{ P \mid (+P) \in \Phi(S) \lor (-P) \in \Phi(S) \} \).

The independent signatures condition ensures that any disjointness conditions are either 1) a result of the fact that the ancestors of \( S_1 \) and \( S_2 \) are explicitly stated as disjoint, or 2) a result of a contradictory predicate assignment (the case when \( S_1 \) and \( S_2 \) have compatible signatures, so there exists a parent that resolves which of \((+P)\) or \((-P)\) hold for both \( S_1 \) and \( S_2 \).

The discussion above leads to the definition of i-trees, for which the graphical representation of i-diagrams, including the circles and squares that represent Split and Comp views.

**Definition 9 (i-trees, \( \text{iT} \)).** An i-tree \( T \) is a simple i-diagram such that \( T = \text{\iTree} \) or such that all of the following three conditions hold:

1. \( T \) is tree shaped
2. \( T \) has independent views
3. \( T \) has independent signatures.

We denote by \( \text{iT} \) the set of i-trees.

The following theorem justifies why all three conditions in our definition of i-trees are necessary. Its proof is based on a reduction from graph 3-colorability, which can be encoded using slightly different i-diagrams for each of the three cases. The common property of these diagrams is that they can encode disjointness of arbitrary pairs of nodes.

**Theorem 2.** Omitting any one out of three conditions from Definition\[5\] yields a class of diagrams whose satisfiability is NP-hard.

We note that in addition to NP-hardness, the omission of tree shaped or independent views properties in fact retains the full expressive power of CBAC constraints, using a similar argument as in Lemma\[4\].

Our ability to specify i-trees as a natural subclass of i-diagrams justifies the definition of i-diagrams themselves. For example, the definition of i-trees would have been more complex had we chosen to represent disjointness using a binary relation \( s_1 \cap s_2 = \emptyset \).

Let us also observe that, despite the imposed restrictions, i-trees are fairly expressive. In particular they can express hierarchical decomposition of a set given by a node \( S \) into disjoint sets \( S_1, \ldots, S_n \), by letting \( \{S_1, \ldots, S_n\} \in \text{Split}(S) \cap \text{Comp}(S) \). Despite the independent view condition, we can have multiple orthogonal decompositions, so \( \{S_1, \ldots, S_m\} \in \text{Split}(S) \cap \text{Comp}(S) \) for \( \{S_1, \ldots, S_m\} \cap \{S_1, \ldots, S_n\} = \emptyset \). This allows i-trees to naturally express generalized typestate constraints.

### 6. Deciding the Satisfiability of I-Trees

In this section we prove that the satisfiability of i-trees is decidable in polynomial time. For this purpose we introduce a set of weak consistency conditions \( C_i \) (Definition\[10\]) such that:

**Claim 1.** We can enforce weak consistency for any satisfiable i-tree using a rewriting system \( R^w \) (Definition\[11\]) with the following properties (Lemma\[12\]):

- \( R^w \) is semantic-preserving;
- if a non-\( \bot \) i-tree in \( R^w \) normal form, then it satisfies weak consistency conditions;
- for a particular strategy (Figure\[2\]) the system \( R^w \) terminates in polynomial time.

Every i-tree that satisfies weak consistency conditions is satisfiable; Lemma\[14\] gives an algorithm for constructing a model for any i-tree that satisfies weak consistency conditions.

Figure\[3\] summarizes the polynomial-time satisfiability decision procedure whose correctness (Theorem\[3\]) follows from the results of this section.

**Definition 10 (Weak Consistency).** An i-tree satisfies weak consistency iff \( T \not\equiv_{\bot} \) and \( T \) satisfies the following conditions for all \( S \in E \):

\[
\forall S' \in \text{Sons}(S), \ \Phi(S') \supseteq \Phi(S) \quad (C_1)
\]

\[
\text{CSup}(S) > 0 \Rightarrow \forall P \in \text{PN} \ { (+P) \not\in \Phi(S) } \quad (C_2)
\]

\[
\forall Q \in \text{Comp}(S), \ \text{CSup}(S) \leq \Sigma(\text{CSup}[Q]) \quad (C_3)
\]

\[
\forall Q \in \text{Split}(S), \ \text{CInf}(S) \geq \Sigma(\text{CInf}[Q]) \quad (C_4)
\]

\[
\text{CInf}(S) \leq \text{CSup}(S) \quad (C_5)
\]

### 6.1 A Rewriting System \( R^w \) for Enforcing Weak Consistency

We introduce the following rewriting system to enforce weak consistency properties when possible.

**Definition 11 (System \( R^w \)).** For each tuple \((k, \text{name}, \text{condition}, \text{effect})\) in Figure\[4\] we define a rewriting rule on i-diagrams by

\[
D \stackrel{\text{spot name}}{\rightarrow} D' \overset{\text{def}}{=} (D \not\equiv_{\bot} \land \text{condition} \land D' = D[\text{effect}])
\]

for each assignment spot of the free variables appearing in the condition column. We define \( R_k \) by

\[
D \stackrel{\text{spot name}}{\rightarrow} D' \overset{\text{def}}{=} \exists \text{spot}. \ D \stackrel{\text{spot name}}{\rightarrow} D'
\]

We define \( R^w \) as union of \( \rightarrow_{R_k} \) for \( 1 \leq j \leq 5 \).
| k  | name   | conditions                          | effect                        |
|----|--------|-------------------------------------|-------------------------------|
| 1  | DnPhi  | \(a_1 \cdot S \in \text{Sons}(S')\) \(b_1 \cdot \phi_0 \models \Phi(S) \cup \Phi(S')\) \(c_1 \cdot \phi_0 \not\subseteq \Phi(S)\) | \(\Phi(S) \models \phi_n\) |
| 2  | Unsat  | \(a_2 \cdot \{(+P, -P)\} \subseteq \Phi(S)\) \(b_2 \cdot n \equiv 0\) \(c_2 \cdot \text{CSup}(S) > n\) | \(\text{CSup}(S) \models n\) |
| 3  | UpSup  | \(a_3 \cdot Q \in \text{Comp}(S)\) \(b_3 \cdot n \equiv \sum(\text{CSup}(Q))\) \(c_3 \cdot \text{CSup}(S) > n\) | \(\text{CSup}(S) \models n\) |
| 4  | UpInf  | \(a_4 \cdot Q \in \text{Split}(S)\) \(b_4 \cdot n \equiv \sum(\text{Clnf}(Q))\) \(c_4 \cdot \text{Clnf}(S) < n\) | \(\text{Clnf}(S) \models n\) |
| 5  | Error  | \(a_5 \cdot \text{Clnf}(S) > \text{CSup}(S)\) | \(D \models \bot_d\) |

**Figure 7.** System \(\mathcal{R}^w\) for ensuring weak consistency

![Diagram](image)

**Figure 8.** An example sequence of rewriting steps for \(\mathcal{R}^w\)

Figure 8 shows an example sequence of rewriting steps applied to an i-tree.

**Lemma 5 (Properties of \(\mathcal{R}^w\)).**

1. \(\mathcal{R}^w\) is iT-stable, that is:
   \[
   T \in \text{iT} \land T \xrightarrow{\mathcal{R}^w} T' \Rightarrow T' \in \text{iT}
   \]

2. \(\mathcal{R}^w\) preserves the semantics, that is:
   \[
   D \xrightarrow{\mathcal{R}^w} D' \Rightarrow D \equiv D'
   \]

3. \(\mathcal{R}^w\) enforces weak consistency when possible, that is, a diagram \(D\) in \(\mathcal{R}^w\) normal form is either equal to \(\bot_d\) or it is weakly consistent

4. \(\mathcal{R}^w\) terminates in polynomial time for the strategy corresponding to the algorithm \(\mathcal{R}_{\text{Itree}}\) in Figure 9

**Proof sketch.**

1. Follows easily from the fact that \(\mathcal{R}^w\) rules do not modify Sons, Split, Comp.
2. Follows by construction of \(\mathcal{R}^w\) rules. Suppose \(D \xrightarrow{\mathcal{R}^w} D'\). Then \(D \models D'\) follows from conditions \(a_i (1 \leq i \leq 5)\), and \(D' \models D\) follows from conditions \(c_i (1 \leq i \leq 4)\).
3. For every \(k = 1..5\), the condition of application of the rule \(R_k\) corresponds to the negation of \(C_k\). When a diagram is in normal form for the rule \(R_k\), it either satisfies \(C_k\), or is \(\bot_d\).
4. To prove that \(\mathcal{R}_{\text{Itree}}\) corresponds to a polynomial strategy, we prove by induction that applying the rule \(R_k\) in the specified direction (from the root to the leaves or from the leaves to the root), enforces \(C_k\) everywhere, and when \(C_k\) holds, the rule is not applicable anymore. Finally, we prove that each rule \(R_k\) for \(k = 1..5\) preserves the conjunction of properties \(A_{j=1...(k-1)} R_j\), and as a consequence, we never need to reapply any of the rules \(R_j\) for \(j < k\).

**6.2 Constructing Models for Weakly Consistent I-Trees**

The following Lemma 6 is crucial for the completeness of our algorithm, and justifies the definition of weak consistency.

**Lemma 6 (Model Construction).** If an i-tree \(T\) is weakly consistent, then we can construct a model for \(T\).

The high-level idea of the proof of Lemma 6 is to first build the first two components \((\Delta, \alpha)\) of the model, and then extend the model with \(\Sigma\) using the independent signatures condition for i-trees. We build the \((\Delta, \alpha)\) part of the model by building a model for each subtree using an induction on the height of the i-tree. To construct models that satisfy Split and Comp constraints in the inductive step, we use a stronger induction hypothesis: we show that there exists a model \((\Delta, \alpha)\) for a tree rooted in node \(S\) with \(|\Delta| = k\) for all \(\text{Clnf}(S) \leq k \leq \text{CSup}(S)\), and we rely on the properties of weak consistency to prove the inductive step. The proof of this lemma is interesting because similar ideas are used when building example models that show the completeness in Section 7.

Putting all results in this section together using the argument at the beginning of the section, we obtain the following theorem.

**Theorem 3 (ItreeSAT Correctness).** \(T\) is satisfiable if and only if \(\mathcal{R}_{\text{Itree}}(T) \neq \bot_d\). Therefore, the algorithm ItreeSAT in Figure 9 is a sound and complete polynomial-time decision procedure for the satisfiability of i-trees.

**7. Deciding Subsumption of I-Trees**

The goal of this section is to prove that we can decide the subsumption of i-trees in polynomial time. Note that the subclass of i-trees is not closed under negation or implication, so we cannot decide \(T \models T'\) by checking the satisfiability of \(\neg(T \Rightarrow T')\). Instead, our approach is to bring \(T\) into a form where the properties of the models of \(T\) are easy to read from \(T\). We then check that \(T\) entails each of the conditions that correspond to the semantics of \(T'\).
The intuitive condition of being easy to read in the notion of strong consistency. We build on the system \( R^w \) from the previous section to create a larger rewriting system \( R \) for ensuring strong consistency. We introduce a polynomial-time strategy for \( R \) that transforms every i-tree into \( \perp_d \) or into an i-tree that is strongly consistent, and we give polynomial-time algorithms for extracting the information from strongly consistent i-trees.

**Definition 12 (Strong Consistency).** An i-tree \( T \) is strongly consistent if it is weakly consistent and satisfies all of the following properties:

\[
\begin{align*}
\forall Q \in \text{Comp}(S), \forall S_0 \in Q, \\
\text{Clnf}(S_0) \geq \text{Clnf}(S) - \Sigma(\text{CSup}[Q - \{S_0\}]) \quad (C_6) \\
\forall Q \in \text{Split}(S), \forall S_0 \in Q, \\
\text{CSup}(S_0) \leq \text{CSup}(S) - \Sigma(\text{Clnf}[Q - \{S_0\}]) \quad (C_7) \\
\forall Q \in \text{Split}(S), Q \notin \text{Comp}(S) \Rightarrow \\
\text{CSup}(S) > \Sigma(\text{Clnf}[Q]) \quad (C_8) \\
\forall Q \in \text{Comp}(S), Q \notin \text{Split}(S) \Rightarrow \\
\text{Clnf}(S) < \Sigma(\text{CSup}[Q]) \quad (C_9) \\
\forall Q \in \text{Comp}(S), \exists \Phi(Q) \Rightarrow \\
\left| \Phi(Q) \right| \leq \Phi(S) \quad (C_{10}) \\
S \neq \emptyset \Rightarrow \text{CSup}(S) > 0 \quad (C_{11}) \\
Q \in \text{Comp}(S) \Rightarrow |Q| > 1 \quad (C_{12})
\end{align*}
\]

7.1 A rewriting system \( R \) to enforce strong consistency

This section follows the development of Section 6.1.

**Definition 13 (System \( R \)).** The system \( R \) extends \( R^w \) with the additional rules of Figure 10, analogously to Definition 17.

**Lemma 7 (Properties of \( R \)).** 1. \( R \) is IT-stable, that is \( T \in \text{IT} \land T \xrightarrow{R} T' \Rightarrow T' \in \text{IT} \)

2. \( R \) preserves the semantics, that is \( D \xrightarrow{R} D' \Rightarrow D \equiv D' \)

**Figure 10. Rules for System \( R \)**

---

We formalize the intuitive condition of being easy to read in the notion of strong consistency. We build on the system \( R^w \) from the previous section to create a larger rewriting system \( R \) for ensuring strong consistency. We introduce a polynomial-time strategy for \( R \) that transforms every i-tree into \( \perp_d \) or into an i-tree that is strongly consistent, and we give polynomial-time algorithms for extracting the information from strongly consistent i-trees.

**Definition 12 (Strong Consistency).** An i-tree \( T \) is strongly consistent if it is weakly consistent and satisfies all of the following properties:

\[
\begin{align*}
\forall Q \in \text{Comp}(S), \forall S_0 \in Q, \\
\text{Clnf}(S_0) \geq \text{Clnf}(S) - \Sigma(\text{CSup}[Q - \{S_0\}]) \quad (C_6) \\
\forall Q \in \text{Split}(S), \forall S_0 \in Q, \\
\text{CSup}(S_0) \leq \text{CSup}(S) - \Sigma(\text{Clnf}[Q - \{S_0\}]) \quad (C_7) \\
\forall Q \in \text{Split}(S), Q \notin \text{Comp}(S) \Rightarrow \\
\text{CSup}(S) > \Sigma(\text{Clnf}[Q]) \quad (C_8) \\
\forall Q \in \text{Comp}(S), Q \notin \text{Split}(S) \Rightarrow \\
\text{Clnf}(S) < \Sigma(\text{CSup}[Q]) \quad (C_9) \\
\forall Q \in \text{Comp}(S), \exists \Phi(Q) \Rightarrow \\
\left| \Phi(Q) \right| \leq \Phi(S) \quad (C_{10}) \\
S \neq \emptyset \Rightarrow \text{CSup}(S) > 0 \quad (C_{11}) \\
Q \in \text{Comp}(S) \Rightarrow |Q| > 1 \quad (C_{12})
\end{align*}
\]

7.1 A rewriting system \( R \) to enforce strong consistency

This section follows the development of Section 6.1.

**Definition 13 (System \( R \)).** The system \( R \) extends \( R^w \) with the additional rules of Figure 10, analogously to Definition 17.

**Lemma 7 (Properties of \( R \)).** 1. \( R \) is IT-stable, that is \( T \in \text{IT} \land T \xrightarrow{R} T' \Rightarrow T' \in \text{IT} \)

2. \( R \) preserves the semantics, that is \( D \xrightarrow{R} D' \Rightarrow D \equiv D' \)

**Figure 11. Polynomial-time algorithm \( R_{nf}(T) \) to compute \( R \) normal form**

3. \( R \) enforces strong consistency when possible, that is, a diagram \( D \) in \( R \) normal form is either equal to \( \perp_d \) or it is strongly consistent.

4. \( R \) terminates in polynomial time for the strategy corresponding to the algorithm \( R_{nf} \) described in Figure 11.

**Proof sketch.**

1. The IT-stability is trivial for the rules \( \text{DnInf}, \text{DnSup}, \text{UpPhi} \). The other rules are marked with a star and we use the algorithm Simplify. In fact, we can show that it is not necessary to apply Simplify in its full generality, but only to remove any redundant views introduced by \( \text{CCmp} \) and \( \text{CSplit} \), remove any self edges introduced by the operation \( \text{Merge} \) used in the rules \( \text{Equal} \) and \( \text{Void} \), and to remove the edges going to \( \emptyset_d \) that can be introduced by the rule \( \text{Void} \).

2.3. Follow by construction as in the previous section.

4. This part is significantly more difficult than for system \( R^w \), because the interactions between the rules are more complex, but follows the same structure as the proof for \( R^w \).

7.2 Extracting Information from Strongly Consistent I-Trees

In this section we start from a strongly consistent i-tree \( T \) and consider the problem of checking \( T \mid= D' \). Analyzing Definition 2, we observe that a diagram corresponds to a conjunction of constraints. Therefore, the subsumption problem \( T \mid= D' \) corresponds to the problem of verifying that \( T \) entails atomic formulas of the form \( s = \emptyset, s_1 = s_2, s_1 \subseteq s_2, a \leq |s| \leq b, s \subseteq P, s \subseteq P', s_1 \cap s_2 = \emptyset \) and \( s \subseteq \{s_1, \ldots, s_n\} \). Without the danger of confusion, we write \( T \mid= A \) when the atomic formula \( A \) holds in all models for \( T \).

**Theorem 4.** Let \( T \) be a strongly consistent i-tree and let \( H_A^T \) for atomic formula \( A \) be as defined in Figure 12. Then \( T \mid= A \) if and only if \( H_A^T \).
Algorithm for Computing \( T \models D' \) for a an i-tree \( T \) and an arbitrary diagram \( D' \).

It is easy to verify that \( H^T_i \) implies \( T \models A \). The proof of the converse is based on the following two lemmas, which provide a link between strong and weak consistency.

**Lemma 8 (Bounds Refinement).** Let \( T \) be a strongly consistent i-tree, \( S \subseteq S \) such that \( \text{Clnf}(S) \leq i \leq s \leq \text{CSup}(S) \), let \( T' = T \langle \text{Clnf}(S) \rightarrow i, \text{CSup}(S) \rightarrow s \rangle \) and \( T_{\text{nf}} = R_{\text{nf}}(T) \). Then

1. \( T_{\text{nf}} \neq \bot_d \)
2. \( T_{\text{nf}} \models T \)

**Proof.** If \( S \neq \emptyset \), we have \( \text{CSup}(S) > 0 \) by \( C_{11} \) and therefore the i-tree \( T' \models T \langle \text{Clnf}(S) \rightarrow \max(1, \text{Clnf}(S)) \rangle \) subsumes \( T \). By Lemma 8, \( T' \) is satisfiable, and we can take any model of \( T' \) as a model of \( T \).

**Lemma 10.** If an i-tree \( T \) is strongly consistent, then for all \( S \in S \) we have

\[ S \neq \emptyset \Rightarrow \exists M, M \models T \wedge \neg \exists M(S) \neq \emptyset \]

**Proof.** If \( S \neq \emptyset \), we have \( \text{CSup}(S) > 0 \) by \( C_{11} \) and therefore the i-tree \( T' \models T \langle \text{Clnf}(S) \rightarrow \max(1, \text{Clnf}(S)) \rangle \) subsumes \( T \). By Lemma 8, \( T' \) is satisfiable, and we can take any model of \( T' \) as a model of \( T \).

**Lemma 11.** If an i-tree \( T \) is strongly consistent, then for all \( S \in S \) we have

\[ \exists M, M \models T \wedge \neg \exists M(S) = \text{Clnf}(S) \]

\[ \exists M, M \models T \wedge \neg \exists M(S) = \text{CSup}(S) \]

**Proof.** According to Lemma 8, the two i-trees \( T_1 \models T \langle \text{CSup}(S) \rightarrow \text{Clnf}(S) \rangle \) and \( T_2 \models T \langle \text{Clnf}(S) \rightarrow \text{CSup}(S) \rangle \) are satisfiable, and both \( T_1 \) and \( T_2 \) trivially subsume \( T \). Any model \( M \) of \( T_1 \) is such that \( \neg \exists M(S) \models \text{Clnf}(S) \), and any model \( M \) of \( T_2 \) is such that \( \exists M(S) \models \text{CSup}(S) \).

**Lemma 12.** If an i-tree \( T \) is strongly consistent, \( S_0 \subseteq S \), \( C \in P(S) \), and \( \text{Included}(S_0, C, T) \) returns false, then

\[ \exists M, M \models T \wedge \neg \exists M(S_0) \subset \neg \exists M(C) \]

**Proof.** Assume that \( \text{Included} \) returns false. We argue that the model \( M \) exists in several steps. Let \( Q_0 \) be the smallest...
set of nodes such that:
\[ S_0 \sim S \Rightarrow S \in Q_0 \]
\[ S \in Q_0 \land C_1 \in \text{Comp}(S) \]
\[ \land S_1 \in C_1 \land -\text{Incl}(S_1) \] \[ \Rightarrow S_1 \in Q_0 \]

By definition of Incl, we have \( Q_0 \cap C = \emptyset \).

\( Q_0 \) is tree-shaped by construction, but may contain two nodes which are explicitly disjoint. We therefore compute a subtree \( Q_1 \) of \( Q_0 \) by starting from the root and keeping at most one son for each complete view. In this process we ensure that \( Q_1 \) contains \( S_0 \), by avoiding to cut the branch which leads to \( S_0 \).

We then apply Lemma 9 to \( Q_1 \) and construct a model of the resulting i-tree while enforcing that a certain element \( x \in \pi(S) \) is such that \( x \in \pi(S') \Leftrightarrow S' \in Q_1 \) for all \( S' \in S \). More precisely, we prove by induction on \( n \) that for each node \( S_1 \) of \( Q_1 \) of depth \( n \) in the tree \( Q_1 \) we can construct a model \( \langle \Delta, \alpha, \Xi \rangle \) for the sub-i-tree of \( T \) with root \( S_1 \) such that
\[ \forall S', S' \sim S_1 \Rightarrow (x \in \pi(S') \Leftrightarrow S' \in Q_1) \]

If \( n = 0 \) then \( S_1 \) is a leaf of \( (Q_1, \sim) \) and has no complete view by construction of \( Q_1 \). Then using \( C_8 \) we show that we can construct a model of the sub-i-tree with root \( S_1 \) containing a fresh element (not included in any of the sons of \( S_1 \)).

If \( n > 0 \), we can deal with the split views in the same way, but this time \( S_1 \) can have some complete views. If this complete view contains a unique split view, we avoid merging \( x \) in a son of \( S_1 \) with elements in the other sons of \( S_1 \). If there exist more than one split view, we can use \( C_8 \) and construct the model using a refinement of the ideas of Lemma 6.

Finally, since \( S_0 \in Q_1 \), we have \( x \notin \pi(S_0) \), and since \( C \cap Q_1 = \emptyset \) we have \( x \notin \pi(C) \).

**Lemma 13.** If an i-tree \( T \) is strongly consistent, for all \( S_1, S_2 \in S \) such that \( S_1 \neq \emptyset \) and \( S_2 \neq \emptyset \) we have
\[ \neg (S_1 \sim S_2) \Rightarrow \exists M. \ M \models \langle T \land \pi_M(S_1) \notin \pi_M(S_2) \rangle \]

**Proof.** The property \( \exists M. \ M \models \langle T \land \pi_M(S_1) \notin \pi_M(S_2) \rangle \) can be checked using Included\( (S_1, S_2, T) \). Using \( C_{12} \) we show that this test is equivalent to test \( S_1 \sim S_2 \).

**Lemma 14.** If an i-tree \( T \) is strongly consistent, for all \( S_1, S_2 \in S \) we have
\[ S_1 \neq S_2 \Rightarrow \exists M. \ M \models \langle T \land \pi_M(S_1) \neq \pi_M(S_2) \rangle \]

**Proof.** If \( S_1 \neq S_2 \), then \( \neg (S_1 \sim S_2) \) or \( \neg (S_1 \sim S_1) \). In either case the result follows from Lemma 13.

**Lemma 15.** If an i-tree \( T \) is strongly consistent, then for all \( S \in S \) and \( P \in \Pi_N \) we have
\[ (+P) \notin \Phi(S) \Rightarrow \exists M. \ M \models \langle T \land \pi_M(S) \notin \Xi(P) \rangle \]
\[ (-P) \notin \Phi(S) \Rightarrow \exists M. \ M \models \langle T \land \pi_M(S) \notin \Xi(P) \rangle \]

**Proof sketch.** Let \( S \in S \) be such that \((+P) \notin \Phi(S) \). We define \( Q \triangleq \{ S' \in S \mid (+P) \notin \Phi(S') \} \). Using \( C_{10} \) and \( C_1 \) we show that Included\( (S, Q, T) \) returns false. By Lemma 12 there exists a model such that \( \pi(S) \notin \bigcup_{Q \in P} \pi(Q) \). We then change the model by redefining \( \Xi \) on \( \Pi_N \) as \( \Xi(P) = \bigcup Q \pi(Q) \). Then \( \pi(S) \notin \Xi(P) \).

The case \(( -P) \notin \Phi(S) \) follows from the previous case by swapping \((+P) \) and \(( -P) \) in the i-tree and taking complements of \( \Xi(P) \).

**Lemma 16.** If an i-tree \( T \) is strongly consistent, then for all \( S_1, S_2 \in S \) such that \( S_1 \neq \emptyset \), \( S_2 \neq \emptyset \) we have
\[ \neg (\text{disj}_T(S_1, S_2)) \Rightarrow \exists M. \ M \models \langle T \land \pi_M(S_1) \cap \pi_M(S_2) \neq \emptyset \rangle \]

From the previously stated lemmas, we can prove Theorem 4.

From Theorem 4 and Lemma 7 we conclude that the algorithm in Figure 13 is a correct and complete test for subsumption, not only of between trees, but also between a tree and an arbitrary diagram.

## 8. Related Work

**Boolean algebras with cardinalities.** Quantifier-free formulas of boolean algebra are NP-complete [33]. Quantified formulas of boolean algebra are in alternating exponential space with a linear number of alternations [31]. Cardinality constraints naturally arise in quantifier elimination for boolean algebras. [31] [32] [43]. Quantifier elimination implies that each first-order formula of the language of boolean algebras is equivalent to some quantifier-free formula with constant cardinalities; however, quantifier elimination may introduce an exponential blowup. The first-order theory of boolean algebras of finite sets with symbolic cardinalities, or, equivalently, boolean algebras of sets with equicardinality operator is shown decidable in [44]. These results are repeated, motivated by constraint solving applications, in [23] [39] and a special case with quantification over elements only is presented in [24]. Upper and lower bounds on the complexity of this problem were shown in [22] which also introduces the name BAPA, for Boolean Algebra with Presburger Arithmetic. The quantifier-free case of BAPA was studied in [45] with an NEXPTIME decision procedure, which is also achieved as a special case of [23] [22]. The new decision procedure in the present paper improves this bound to PSPACE and gives insight into the problem by reducing it to boolean algebras with binary-encoded large cardinalities, and showing that it is not necessary to explicitly construct all set partitions.

Several decidable fragments of set theory are studied in [10]. Cardinality constraints also occur in description logics [5] and two-variable logic with counting [35] [12] [38]. However, all logics of counting that we are aware of have complexity that is beyond PSPACE.

We are not aware of any previously known fragments of boolean algebras of sets with cardinality constraints that have polynomial-time satisfiability or subsumption algorithms. Our polynomial-time result for i-trees is even more interesting in the light of the fact that our constraints can express some "disjunction-like" properties such as \( A = B \cup C \).

**Set constraints.** Set constraints [11] [12] [2] [3] are incomparable to the constraints studied in our paper. 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, which makes them suitable for encoding type inference [2] and interprocedural analysis [20] [34]. Researchers have also explored the efficient computation of the subset relation for set constraints [13]. On the other hand, set constraints do not support cardinality operators that are useful in modelling databases [40] [11] and analysis of the sizes of data structures [29]. Tarskian constraints use uninterpreted function symbols instead of free function symbols and have very high complexity [19].

## 9. Conclusions

Constraints on sets and relations are very useful for analysis of software artifacts and their abstractions. Reasoning about sets and relations often involves reasoning about their sizes. For example, an integer field may be used to track the size of the set of objects stored in a data structure. In this paper, we have presented new complexity results and algorithms for solving constraints on boolean algebra of sets with symbolic and constant cardinality constraints. We have presented symbolic constraints and large constant constraints, gave more efficient algorithm for quantifier-free symbolic constraints, identified several sources of NP-hardness of constraints, and presented a new class of constraints for which satisfiability and en-
On Algorithms and Complexity for Sets with Cardinality Constraints

References

[1] A. Aiken. Introduction to set constraint-based program analysis. Science of Computer Programming, 35:79–111, 1999.
[2] A. Aiken, D. Kozen, M. Vardi, and E. Wimmers. The complexity of set constraints. In Proceedings of Computer Science Logic 1993, pages 1–17, September 1993.
[3] A. Aiken, D. Kozen, and E. Wimmers. Decidability of systems of set constraints with negative constraints. Information and Computation, 122, 1995.
[4] A. Aiken and E. L. Wimmers. Type inclusion constraints and type inference. In Proc. 7th ACM conference on Functional Programming and Computer Architecture, Copenhagen, Denmark, 1993.
[5] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. The Description Logic Handbook: Theory, Implementation and Applications. CUP, 2003.
[6] L. Bachmair, H. Ganzinger, and U. Waldmann. Set constraints are the monadic class. In Logic in Computer Science, pages 75–83, 1993.
[7] E. Börger, E. Grädel, and Y. Gurevich. The Classical Decision Problem. Springer-Verlag, 1997.
[8] R. E. Bryant. Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers, C-35(8):677–691, August 1986.
[9] M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. An algorithm to evaluate quantified boolean formulae and its experimental evaluation. J. Autom. Reasoning, 28(2):101–142, 2002.
[10] D. Cantone, E. Omodeo, and A. Policriti. Set Theory for Computing. Springer, 2001.
[11] P. P.-S. Chen. The entity-relationship model—toward a unified view of data. ACM Transactions on Database Systems (TODS), 1(1):9–36, 1976.
[12] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 1997, release 1999.
[13] M. Fähndrich, J. Foster, Z. Su, and A. Aiken. Partial online cycle elimination in inclusion constraint graphs. In PLDI ’98, 1998.
[14] S. Feferman and R. L. Vaught. The first order properties of products of algebraic systems. Fundamenta Mathematicae, 47:57–103, 1959.
[15] M. J. Fisher and M. O. Rabin. Super-exponential complexity of presburger arithmetic. SIAM-AMS Proceedings, 7, 1974.
[16] C. Flanagan, R. Joshi, X. Ou, and J. B. Saxe. Theorem proving using lazy proof explication. In CAV, pages 355–367, 2003.
[17] H. Ganzinger, G. Hagen, R. Nieuwenhuis, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In R. Alur and D. Peled, editors, 16th CAV, 1997. release 1999.
[18] C. Flanagan, R. Joshi, X. Ou, and J. B. Saxe. Theorem proving using lazy proof explication. In CAV, pages 355–367, 2003.
[19] E. Grädel, M. Otto, and E. Rosen. Two-variable logic with counting quantifiers. Proceedings of the 29th International Conference on Logic in Computer Science (ICALP’03), 2003.
[20] J. Kodumal and A. Aiken. The set constraint/CFL reachability problem. In Proc. ACM PLDI, June 2004.
[21] D. Kozen. Complexity of boolean algebras. Theoretical Computer Science, 10:221–247, 1980.
[22] V. Kuncak, H. H. Nguyen, and M. Rinard. An algorithm for deciding BAPA: Boolean Algebra with Presburger Arithmetic. In 20th International Conference on Automated Deduction, CADE-20, Tallinn, Estonia, July 2005.
[23] V. Kuncak and M. Rinard. The first-order theory of sets with cardinality constraints is decidable. Technical Report 958, MIT CSAIL, July 2004.
[24] V. Kuncak and M. Rinard. On decision procedures for set-valued fields. Technical Report 975, MIT CSAIL, November 2004.
[25] V. Kuncak and M. Rinard. Decision procedures for set-valued fields. In 1st international workshop on abstract interpretation of object-oriented languages (AIOOL’05), 2005.
[26] P. Lam, V. Kuncak, and M. Rinard. Generalized typestate checking using set interfaces and pluggable analyses. SIGPLAN Notices, 39:46–55, March 2004.
[27] P. Lam, V. Kuncak, and M. Rinard. On our experience with modular pluggable analyses. Technical Report 965, MIT CSAIL, September 2004.
[28] P. Lam, V. Kuncak, and M. Rinard. Cross-cutting techniques in program specification and analysis. In 4th International Conference on Aspect-Oriented Software Development (AOSD’05), 2005.
[29] P. Lam, V. Kuncak, and M. Rinard. Generalized typestate checking for data structure consistency. In 6th International Conference on Verification, Model Checking and Abstract Interpretation, 2005.
[30] P. Lam, V. Kuncak, and M. Rinard. Hob: A tool for verifying data structure consistency. In 14th International Conference on Compiler Construction (tool demo), April 2005.
[31] L. Loewenheim. Über Möglichkeiten im relativenkalkül. Math. Annalen, 76:228–251, 1915.
[32] B. Marnette, V. Kuncak, and M. Rinard. On algorithms and complexity for sets with cardinality constraints. Technical report, MIT CSAIL, August 2005.
[33] K. Marriott and M. Odersky. Negative boolean constraints. Technical Report 94/203, Monash University, August 1994.
[34] D. Melski and T. Reps. Interconvertibility of a class of set constraints and context-free language reachability. TCS, 248:29–98, November 2000.
[35] J. Rumbaugh, I. Jacobson, and G. Booch. The Unified Modeling Language Reference Manual. Addison-Wesley, Reading, Mass., 1999.
[36] M. Sipser. Introduction to the Theory of Computation. PWS Publishing Company, 1997.
[37] B. Marnette, V. Kuncak, and M. Rinard. On algorithms and complexity for sets with cardinality constraints. Technical report, MIT CSAIL, August 2005.
[38] G. U. Propp. Über “Produktations- und Summationsprobleme”, welche gewisse Klassen von Aussagen betreffen. Skrifter utgit av Videnskapsselskapet i Kristiania, I. klasse, no. 3, Oslo, 1919.
[39] A. Tarski. Arithmetical classes and types of boolean algebras. Bull. Amer. Math. Soc., 55, 64, 1192, 1949.
[40] C. G. Zarba. A quantifier elimination algorithm for a fragment of set theory involving the cardinality operator. In 18th International Workshop on Unification, 2004.
[41] C. G. Zarba. Combining sets with cardinals. Journal of Automated Reasoning, 34(1), 2005.
[42] K. Zee, P. Lam, V. Kuncak, and M. Rinard. Combining theorem proving with static analysis for data structure consistency. In International Workshop on Software Verification and Validation (SVV 2004), Seattle, November 2004.
[43] L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient conflict driven learning in boolean satisfiability solver. In ICCAD, pages 279–285, 2001.
[44] L. Zhang and S. Malik. Conflict driven learning in a quantified boolean satisfiability solver. In ICCAD ’02: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design, pages 442–449, New York, NY, USA, 2002. ACM Press.
A. Proofs

1. I-Diagrams

Lemma 4. I-diagrams have the same expressive power as CBAC constraints.

Proof. We translate an i-diagram into a CBAC constraint as follows. As in Figure 2 we note that $b_1 = b_2, b_1 \subseteq b_2$ can be expressed in the form $|b_1| = 0$, so we may assume that they are part of CBAC. Similarly, $|b_2| \subseteq b$ can be expressed as $b_2 \subseteq S \wedge |s| = k$ for a fresh variable $s$, and $|b_1| \geq b$ can be expressed as $s \subseteq \exists b \wedge |s| = k$. We translate $\bigwedge_i \text{e.g. } |b_1| = 1$. Next consider D = $\exists S, \emptyset \subseteq S, \text{Sons}, \text{Split}, \text{Comp}, \text{Clnf}, (\text{Csup}, \Phi)$. For each $s_i \in S$, let $\eta(s_i) \in S$ be a representative set name. For each $S_i \in S \setminus \{\eta(s_i)\}$ introduce conjunct $S_i = \eta(s_i)$. Next, for each $S_i \in \text{Sons}(S)$, introduce a conjunct $S_i \subseteq S$. For each $+P \in \Phi(S)$, introduce conjunct $S \subseteq P$, and for each $+P \in \Phi(S)$ conjunct $S \subseteq P'$. Express the bounds on conjuncts $|S| \leq \text{Csup}(S)$ and $|S| \geq \text{Clnf}(S)$.

For each $Q \in \text{Split}(S)$ and $S_1, S_2 \in Q$ where $S_1 \neq S_2$, introduce conjunct $|S_1 \cap S_2| = 0$. For each $S_1, \ldots, S_n \in \text{Comp}(S)$ introduce conjunct $S = S_1 \cup \ldots S_n$.

We translate a CBAC constraint into i-diagram using the following observations. It is sufficient to translate the following boolean algebra expressions: $s_0 = s_1 \cup s_2, s_0 = s_1$, and $s_0 \geq k$. We construct an i-diagram whose nodes are singletons. We pick one set variable $u$ to act as a universal set and put $\{s\} \in \text{Sons}(\{u\})$ for every set variable $s$ in the i-diagram. We translate $s_0 = s_1 \cup s_2$ as $\{\{s_1\}, \{s_2\}\} \in \text{Comp}(\{s_0\})$ and translate $s_0 = s_1^k$ as $\{\{s_0\}, \{s_1\}\} \in \text{Split}(\{u\}), \{\{s_0\}, \{s_1\}\} \in \text{Comp}(\{u\})$. We translate $|s| = k$ as $\text{Clnf}(|s|) = k$ and $\text{Csup}(|s|) = k$. Then for each satisfiable assignment of CBAC there is a model for the constructed i-diagram where $\{u\}$ is interpreted as a universal set. Conversely, for a model of i-diagram there is a model for the constructed i-diagram.

Lemma 5. For every i-diagram D we can obtain an equivalent simple i-diagram using the polynomial-time algorithm in Figure 2.

Proof. We argue that algorithm in Figure 2 produces diagram that is i) well-formed, ii) simple, iii) equivalent to the original diagram.

We first observe that after step 2, the following two conditions hold:

C) if $Q \in \text{Split}(S)$ and $S \in Q$, then $Q \subseteq \{S, \emptyset\}$. This condition holds because the step 1.5 of the algorithm merges all nodes $Q \setminus \{S\}$ with $\emptyset$ when $S \in Q \in \text{Split}(S)$.

D) if $Q \in \text{Comp}(S)$, then $Q \neq \emptyset$ (by step 1.3), if $Q = \{\emptyset\}$ then $S = \emptyset$ (by step 1.6).

i) To see that the resulting i-diagram is well-formed, it suffices to check the conditions $\bigcup \text{Split}(S) = \text{Sons}(S)$ and $\bigcup \text{Comp}(S) \subseteq \text{Sons}(S)$. This condition is preserved by factor-diagram construction (for any equivalence relation). It is preserved by step 3 for the following reason. The only nodes removed from $\text{Sons}(S)$ are $\emptyset$ and $S$. These nodes do not appear in $\bigcup \text{Comp}(S)$ because $\emptyset$ is removed from each view $Q$, and views with $S \in Q$ are removed. It remains to check that $\text{Sons}(S) \subseteq \bigcup \text{Split}(S)$ after step 3, and this holds because our condition (C) implies that no element other than $S, \emptyset$ is lost from $\bigcup \text{Split}(S)$ in step 3. The well-formedness condition is preserved by step 4 because this step does not change $\bigcup \text{Comp}(S)$ or $\bigcup \text{Split}(S)$. Step 5 does not violate this condition either because it sets the components of $\emptyset$ to $\emptyset$.

ii) To see that the resulting diagram is simple, we show that it satisfies conditions $a), \ldots , f)$ of Definition 6. After step 2 of the algorithm, the resulting factor-diagram has no cycles of length 2 or more, there are only potentially some self-cycles. These are eliminated in step 3 and no further edges are introduced. Hence, $a)$ holds. For each of the following condition, they are enforced in certain step and not violated afterwards, according to the following table:

| b) | c) | d) | e) | f) |
|----|----|----|----|----|
| 3  | 4  | 4  | 4  | 5  |

iii) We show that semantics is preserved when executing each sequence of steps 1, ..., $k$ for $2 \leq k \leq 5$, that is, each step preserves the semantics provided that it is executed after the previous steps.

$k = 2$. Each equality introduced into $\rho$ is a semantic consequence of the diagram.

$1. \text{Sup}(S_1) \subseteq \text{Sup}(S_2) \subseteq \text{Sup}(S_1)$.

$2. \text{Sup}(S_1) \subseteq \text{Sup}(\emptyset) = \emptyset$.

$3. \text{Sup}(S_1) \subseteq \bigcup \emptyset = \emptyset$.

$4. \text{Sup}(S_1) \cap \text{Sup}(S_2) = \emptyset$ for $\text{Sup}(S_1) = \text{Sup}(S_2)$ so $\text{Sup}(S_2) = \emptyset$, or

$5. \text{Sup}(S_1) \cap \text{Sup}(S_2) = \emptyset$.

It follows that the condition on equality of sets, as well as the conditions on $\text{Clnf}, \text{Csup}, \text{Sons}, \Phi, \text{Comp}$ are all semantically equivalent when applied to the original and the factor diagram. The only semantic condition which can be lost in factor-diagram construction is $\bigcup \text{Split}(S_1, S_2)$ when $S_1$ and $S_2$ nodes are merged, that is, when $S_1 \cap S_2 \in \rho$. However, in this case the disjointness condition follows from $\text{Sup}(S_1) = \emptyset$, which is enforced in 1.3. Therefore, for the particular relation constructed in step 1, factor-diagram is an equivalence preserving transformation.

$k = 3$. We need to show that no information is lost by removing $\emptyset \cup S$ from the sons, as well as split and complete views of $S$. Clearly, removing $\emptyset \cup S$ from $\text{Sons}(S)$ does not change the subset conditions because $\emptyset \subseteq \bigcup \text{Sup}(S)$ and $\bigcup \text{Sup}(S) \subseteq \bigcup \text{Sup}(S)$. Eliminating $\emptyset \cup S$ from $Q \in \text{Comp}(S)$ is justified because the view has the same semantics with or without $\emptyset \cup S$. Dropping a view $Q \in \text{Comp}(S)$ for $S \in Q$ is justified because in that case $\bigcup \text{Sup}(S) \subseteq \bigcup \text{Split}(S) \cap \bigcup \text{Sons}(S_1)$ holds trivially. Eliminating $\emptyset \cup S$ from $Q \in \text{Split}(S)$ is justified because intersection with empty set is always empty, so this condition does not bring any new information. Finally, dropping a $Q \in \text{Split}(S)$ with $S \in Q$ is justified because condition (C) implies that in such case $Q \subseteq \emptyset$, so the Split condition is trivial.

$k = 4$. Removing $\{\emptyset\}$ from $\text{Split}(S)$ preserves semantics because such view carries no information. Similarly, because all maximal views are preserved, removing their subsets does not change the semantics. For $\text{Comp}(S)$, we consider two cases. In first case $\emptyset \notin \text{Comp}(S)$. In this case, removing $\emptyset$ does not have any effect, and it is sound to remove all non-minimal views because they are implied by the minimal views. The second case is $\emptyset \in \text{Comp}(S)$. By condition D) on the step 1, we know that $Q \neq \emptyset$ after the step 2, and the only node removed in step 3 is $\emptyset$, so it must have been the case that $Q = \{\emptyset\}$ after step 2. By condition D), we then have $S = \emptyset$. Because the semantic condition on $\text{Comp}$ for $Q = \emptyset$ reduces to $\text{Sup}(S) = \emptyset$, this condition brings no new information, so we can remove it.

$k = 5$. Because $\bigcup \text{Sup}(\emptyset) = \bigcup \text{Clnf}(\emptyset) = 0$ does not change semantics, similarly for $\Phi(\emptyset) = \emptyset$. We also know that $\text{Sons}(S) \subseteq \{\emptyset\}$ because this condition is ensured by step 2 and is not violated afterwards. Because we have already observed that the diagram is well-formed, we conclude $\text{Comp}(S) \subseteq \{\emptyset\}$ and $\text{Split}(S) \subseteq \{\emptyset\}$, so setting these values to $\emptyset$ does not change the semantics.

Theorem 2. Omitting any one out of three conditions from Definition 5 (1. being tree-shaped, 2. having independent views, and 3. having independent signatures) yields a class of diagrams whose satisfiability is NP-hard.

Proof. Suppose that at least one of the three conditions does not apply to a class of i-diagrams. We then give a reduction from the
On Algorithms and Complexity for Sets with Cardinality Constraints

S = reflexive relation, we first build the $i$-diagram $A$. 2. Termination of System

3COL denotes the NP-complete problem of deciding, given an

$n \in N \times N$ is a symmetric ir-

reflexive relation, such that $\exists \alpha, \exists \exists$ such

Then for each edge $(n_1, n_2) \in E$ we encode the fact that

$n_1$ and $n_2$ must have different colors by enforcing the property

$h(n_1) \cap h(n_2) = 0$ on the models of $D$. We encode this constraint in different ways depending on the class of $i$-diagrams:

- If $D$ allows dependent signatures, we introduce a fresh predicate symbol $P_{n_1, n_2}$, and add $(+P_{n_1, n_2})$ to $\Phi(n_1)$, and $(-P_{n_1, n_2})$ to $\Phi(n_2)$.

- If $D$ allows dependent views, we add $\{n_1, n_2\}$ to $\text{Split}(U)$.

- If $D$ allows multiple fathers, then we simulate dependent

views by introducing a new node $m(n_1, n_2)$. We let

$\text{Cl}(m(n_1, n_2)) = \text{Sup}(m(n_1, n_2)) = \{n_1, n_2\}$, and $\Phi(m(n_1, n_2)) = \emptyset$. We then remove $n_1, n_2$ from $\text{Sons}(U)$, and add $m(n_1, n_2)$ to $\text{Sons}(U)$ instead.

None of these contructions violates more than one of the three considered restrictions. It is straightforward to verify that the diagram is satisfiable iff the graph is colorable, and that the construction of $D$ can be done in polynomial time. This proves that the satisfiability of $i$-diagrams with any of the three restrictions removed is NP-hard.

A.2 Termination of System $R$

Lemma 17 (Invariants of $R$). For every $k \in [1..12]$ the rule $R_k$ preserves $\bigwedge_{j=1..(k-1)} C_j$, or returns $\bot$.

Proof. We analyze each rule $R_k$, for two i-trees $T$ and $T'$ such that $T \rightarrow R_k T'$, assuming that $T$ satisfies $\bigwedge_{j=1..(k-1)} C_j$, and

more precisely, $T \rightarrow_{R_k} T'$, where $spot$ are variable names as they appear in the definition of $R$.

1. (DnPhi). Trivial.
2. (Unsat). $(C_2)$ does not depend on $\text{Csup}$.
3. (UpSup). If $n = 0$, $(C_2)$ is trivially true for $S$ in $D'$. If $n > 0$, by $(C_3)$ $\text{Csup}(S) > 0$ and we have already $\forall P \in \text{PN}. \{+P, -P\} \subseteq \Phi(S)$ and since $\Phi' = \Phi$, $(C_2)$ holds in $T'$.
4. (UpInf). Neither $(C_1), (C_2)$ nor $(C_3)$ depend on $\text{Cl}(S)$.
5. (Error). $T' = \bot$.
6. (DnInf). Only $(C_4)$ and $(C_5)$ depends on $\text{Cl}(S)$.

- $(C_4)$ is maintained for $S$ in $T$ because, noticing that $(C_5)$ is maintained for $S' \neq S$, we have

\[
\text{Cl}(S') = \text{Cl}(S') - \Sigma(\text{Csup}(S)) \\
\leq \text{Csup}(S') - \Sigma(\text{Csup}(Q)) \\
\leq \text{Csup}(S) \\
\]

To prove that $(C_4)$ is maintained in $T'$ we need to check

that $(C_5)$ is maintained for $S$, which is trivial by $(C_6)$, and that $(C_4)$ is maintained for the father $S'$ of $S$ and the views $Q \in \text{Split}(S')$ containing $S$. By property of independent

views there exists only one such view $Q = \{S \cup Q_0\}$

such that $Q_0' \subseteq Q_0$ and

\[
\Sigma(\text{Csup}(Q')) = \text{Cl}(S') + \Sigma(\text{Cl}(S_0')) \\
= \text{Cl}(S' - \Sigma(\text{Csup}(Q_0))) + \Sigma(\text{Cl}(S_0')) \\
= \text{Cl}(S') - \Sigma(\text{Csup}(Q_0 - Q_0)) \\
= \sum_{Q_0 \subseteq Q_0} (\text{Cl}(S_0') - \Sigma(\text{Csup}(Q_0')) \\
\leq \text{Cl}(S') - \Sigma(\text{Csup}(Q_0 - Q_0)) \\
\leq \text{Cl}(S') \\
= \text{Cl}(S')
\]

7. (DnSup). Only $(C_6)$, $(C_7)$ and $(C_8)$ depend on $\text{Csup}$. $(C_4)$ is maintained thanks to $(c_5)$ as for the case of rule $\text{UpSup}$.

- $(C_5)$ is maintained for $S$ in $T'$ because, noticing that $(C_5)$ is maintained for $S' \neq S$ we have $\text{Csup}(S) = \text{Csup}(S') - \Sigma(\text{Cl}(Q_0))$

\[
\geq \text{Cl}(S') - \Sigma(\text{Cl}(Q_0)) \\
\geq \text{Cl}(S') \\
\]

To prove that $(C_4)$ is maintained in $T'$ we need to check that $(C_5)$ is maintained for $S$, which is trivial by $(c_7)$, and that $(C_4)$ is maintained for the father $S'$ of $S$ and the views $Q \in \text{Comp}(S')$ containing $S$. By property of independent views there exists only one such view $Q = \{S \cup Q_0\}$ and

\[
\text{Cl}(S') = \text{Cl}(S') + \Sigma(\text{Cl}(Q_0)) \\
= \text{Cl}(S' - \Sigma(\text{Csup}(Q_0))) + \Sigma(\text{Cl}(Q_0)) \\
= \text{Cl}(S') - \Sigma(\text{Csup}(Q_0 - Q_0)) \\
\leq \text{Cl}(S') - \Sigma(\text{Csup}(Q_0 - Q_0)) \\
\leq \text{Cl}(S') \\
\leq \text{Cl}(S')
\]

8. (CmCmp). Only $(C_1)$ and $(C_6)$ depend on $\text{Comp}$.

- $(C_6)$ is maintained for $S$, $Q$ because

\[
\text{Cl}(S) = \text{Cl}(S) - \Sigma(\text{Cl}(Q)) \\
\leq \text{Cl}(S) - \Sigma(\text{Cl}(Q)) \\
= \text{Cl}(S) \\
\]

- $(C_6)$ is maintained for $S$, $Q$ and all $S_0 \in Q$ because

\[
\text{Cl}(S) = \text{Cl}(S) - \Sigma(\text{Cl}(Q)) \\
\leq \text{Cl}(S) - \Sigma(\text{Cl}(Q)) \\
= \text{Cl}(S) \\
\]

(Remark). If ever we use a simplification afterwards, as indicated by the star in figure $\text{f}$, it can only consists in removing a complete view $Q'$ such that $Q' \subseteq Q$. This operation trivially maintains $\bigwedge_{j=1..(7)} C_j$ because in every properties of consistency where complete views appear they are universally quantified.

9. (CSplit). Only $(C_1)$ and $(C_7)$ depend on $\text{Split}$.

- $(C_7)$ is maintained for $S$, $Q$ because

\[
\text{Cl}(S) = \text{Cl}(S) - \Sigma(\text{Cl}(Q)) \\
\leq \text{Cl}(S) - \Sigma(\text{Cl}(Q)) \\
= \text{Cl}(S) \\
\]

- $(C_7)$ is maintained for $S$, $Q$ and all $S_0 \in Q$ because

\[
\text{Cl}(S) = \text{Cl}(S) - \Sigma(\text{Cl}(Q)) \\
\leq \text{Cl}(S) - \Sigma(\text{Cl}(Q)) \\
= \text{Cl}(S) \\
\]

On Algorithms and Complexity for Sets with Cardinality Constraints

14

2008/2/1
10. (UpPhi). Only (C1) and (C2) depend on Φ

- (C1) is maintained for S and all S′ ∈ Q because
  \[ Φ'(S) = Φ(S) \cup ∩Φ(Q) = Φ(S) \cup (Φ(S') \cap Φ(Q - \{S\})) \]
  \[ ⊆ Φ(S) \cup Φ(S') \)
  \[ ⊆ Φ(S') \]
  \[ = Φ'(S') \]

- We can also prove that (C2) is maintained for S. Suppose there exists P in PN such that \{+P, -P\} ∈ Φ'(S). Then we distinguish three cases.
  - if \{+P, -P\} ⊆ Φ(S), by C2, CSup(S) = 0
  - if \{+P, -P\} \cap Φ(S) = ∅, then \{+P, -P\} ∈ ∩Φ(Q).
    Then for all S′ ∈ Q, CSup(S') = 0 by C2. Then by C3, CSup(S) = 0
  - if \{+P, -P\} ∩ Φ(S) = ±P for some atom ±P ∈ \{±P, -P\}. Then the opposite atom ±P belong to Φ(Q) and by C1, ±P belong to Φ(Q). By C2, each node S′ ∈ Q is such that CSup(S') = 0 and by C3, CSup(S) = 0

In the three cases CSup'(S) = CSup(S) = 0.

11. (Void). If S is the root, the resulting i-tree T is such that S′ = {SN} = \{∅\} and C1 is trivial for all i ∈ [1..11]. Otherwise, we have to check that removing the node S from the sons of the father of S (as indicated in the step 3 of the procedure simplify) maintains \((i, i + 1.10)\). We denote by S0 the father of S and by Q0 the split view of S0 containing S. If S is also contained in a complete view of S0, we denote by C0 this complete view.

- (C1) and (C2) are trivially maintained.
- (C3) is maintained because, if C0 exists and C0′ = C0 - \{S\} is not empty
  \[ CSup'(S0) = CSup(S0) \]
  \[ ≤ ΣCSup(C0) \quad (by \ C3) \]
  \[ = ΣCSup(C0') \quad (by \ a12) \]
  \[ = ΣCSup''(C0') \]
- (C4) is maintained because, if Q0′ = Q0 - \{S\} is not empty
  \[ Clnf'(S0) = Clnf(S0) \]
  \[ ≥ ΣClnf(C0) \quad (by \ C4) \]
  \[ ≥ ΣClnf(C0′) \quad (by \ a12) \]
  \[ = ΣClnf''(C0′) \]
- (C5) is maintained for the node θ_i = S ∪ θ_i of T′ because by (a12) and (C5), Clnf(θ_i) ≤ CSup(S) = 0, by simplicity and (C5), Clnf(θ_i) ≤ CSup(θ_i) = 0, and therefore Clnf''(θ_i) = Max(Clnf(θ_i), Clnf(θ_i)) = 0 ≤ CSup''(θ_i).
- (C6) is maintained for S0, C0 and every S_i ∈ C0 such that S_i ≠ S

- (C7) is maintained for S0, Q0 and every S_i ∈ Q0 such that S_i ≠ S because
  \[ CSup'(S_i) = CSup(S_i) \]
  \[ ≤ CSup(S0) - Σ(Clnf(C0 - \{S_i\})) \quad (by \ C7) \]
  \[ ≤ CSup(S0) - Σ(Clnf(C0 - \{S\} - \{S_i\})) \]
  \[ = CSup(S0) - Σ(Clnf''(C0 - \{S\} - \{S_i\})) \]
- (C8) is maintained for Q0 because
  \[ Clnf'(S0) = Clnf(S0) \]
  \[ ≥ Σ(Clnf(Q0 - \{S\})) \quad (by \ a11) \]
  \[ = Σ(Clnf''(Q0)) \]
- (C9) is maintained for C0 if (exists and C0′ = C0 - \{S0\} ≠ ∅)
  \[ Clnf'(C0) = Clnf(C0) \]
  \[ ≤ Σ(Csup(Q0) \quad (by \ C9) \]
  \[ = Σ(Csup(Q0 - \{S\})) \]
  \[ = Σ(Csup''(Q0)) \]
- (C10) is maintained for C0 if (exists and C0′ = C0 - \{S0\} ≠ ∅)
  \[ ∩ Clnf'(C0) = ∩ Clnf(C0 - \{S\}) \]
  \[ ⊆ Clnf(C0) \]
  \[ ⊆ Clnf''(C0) \quad (by \ C10) \]

12. (Equal). Before to merge S and S′ for \{S′\} ∈ Comp(S) we have

- Clnf(S) = Clnf(S′) by C4 and C6
- CSup(S) = CSup(S′) by C5 and C7
- Φ(S) = Φ(S′) by C1 and C10

Therefore all the properties Ci for i = 1..10 are trivially maintained.

A.3 Model Construction

Lemma 2 (Model Construction) If an i-tree T is weakly consistent, then we can construct a model for T.

Proof. Let T be a weakly consistent i-tree.

We construct a model \((Δ, α, Ξ)\) by first constructing a partial model \((Δ, α)\) for all parts of T except Φ, and then extending \((Δ, α)\) with Ξ to satisfy Φ. Constructing \((Δ, α)\). We write \((Δ, α) \models T\) to denote that Δ and α satisfy those conditions on Φ, Sons, Split, Comp, CSup, Clnf that do not mention Φ in Definition 2.

To show we can construct \((Δ, α)\) such that \((Δ, α) \models T\) we prove by induction on n the following more general claim.

Claim 1. For every i-tree T of height n with root SR:

- \((Δ, α) \models T\)
- \((Δ, α) \models T\), \((Δ, α) \models T\)

If n = 1, the claim holds by C5 taking

- \(Δ = Clnf(SR) = \{1, \ldots, k\}\)

For n > 1, consider an i-tree T with root SR and k ∈ [Clnf(SR), CSup(SR)]. By examining the constraints in T, we choose the cardinalities for subtrees of T, use the induction hypothesis to construct models for subtrees, and paste the models for subtrees into a model for T. We decompose this process into three steps:
1. For each $C \in \text{Comp}(S_R)$, consider a subtree $T_C$ built from $T$ by removing all sons outside $\bigcup C$. We construct a model $M_C$ for $T_C$ of cardinality $k$.
2. For each remaining $Q \in \text{Split}(S_R)$ where $Q \not\subseteq \bigcup \text{Comp}(S_R)$, consider a subtree $T_Q$ with the same root $S_R$ but without the sons outside $\bigcup Q$. We construct a model $M_Q$ for $T_Q$ of cardinality $k$.
3. Because the constructed models have the same cardinality, we can easily merge them to obtain a model for $T$.

**Step 1.** Let $C \in \text{Comp}(S_R)$, and $C \subseteq \text{Split}(S_R)$ such that $C = \bigcup C$. For each $Q \in C$, let $t_Q \overset{def}{=} \min(k, \Sigma(C_{\text{Sup}}(Q)))$. Using $k \geq \text{Clfn}(S_R)$ and $C$, we can show

$$\Sigma(\text{Clfn}(Q)) \leq t_Q \leq \Sigma(C_{\text{Sup}}(Q)) \quad (H1)$$

To each node $S \in Q$ we can therefore assign an integer $K(S) \in [\text{Clfn}(S), C_{\text{Sup}}(S)]$ such that $t_Q = \Sigma(K(Q))$. Let $T_Q$ be the subtree of $T$ rooted at $S$. By induction hypothesis, let $M_S$ be the model of $T_Q$ of cardinality $K(S)$. We can then take the disjoint union of these models to construct a model $M_{Q} \mid_{Q}$ of size $t_Q$ for the forest $\bigcup_{Q \in C} T_Q$. For all $Q \in C$, we have $t_Q \leq k$ by definition of $t_Q$ and $k$.

We finally obtain

$$\max_{Q \in C} t_Q \leq k \quad (H2)$$

Thanks to (H2), we can build a model for the i-tree $T_C$, as follows.

**Step 2.** Let $Q \in \text{Split}(S_R)$, such that $Q$ is not included in any complete view. We construct a model $M_Q$ of size $k$ for the i-forest $T_Q$ by first building a model of size $K(S') = \text{Clfn}(S')$ for each $S' \in Q$. Because $k \geq \text{Clfn}(S) \geq \Sigma(\text{Clfn}(Q))$, by $C$, the disjoint union of these models has cardinality smaller than $k$. By adding the correct number of fresh elements to $\Delta$, we obtain a model of cardinality $k$.

**Remark 3.** (Existence of fresh elements) In Section 4 we use the following property: For all $Q \in \text{Split}(S)$ and $Q \not\subseteq \text{Comp}(S)$, if $\Sigma(\text{Clfn}(Q)) < K(S)$, then there exists a model such that $\overline{\alpha}(S)$ contains an element which does not belong to any $\overline{\alpha}(S')$ for any of the sons of $S$.

**Step 3.** We can apply an arbitrary bijection $\sigma_\alpha : \Delta_\alpha \to [1,n]$ to each model $M_C$, constructed as previously described before to build a model for the entire i-tree. We let $\overline{\alpha}(S_R) = \{1,\ldots,n\}$ and for all $S \neq S_R$, $\overline{\alpha}(S) = \overline{\alpha}(C)$ where $C$ is the view containing an ancestor of $S$ in $\text{Split}(S_R)$ or $\text{Comp}(S_R)$.

**Remark 4.** (Freedom in the choice of $\sigma_\alpha$) If we know that $K(S) > 0$, for any pair $S_1, S_2$ of sons such that $S_1, S_2$ belong neither to the same split view nor to the same complete view, for each choice of elements $x_1, x_2$ in the models $T_{S_1}$ and $T_{S_2}$, we can choose $\sigma_1, \sigma_2$ such that $\sigma_1(x_1) = \sigma_2(x_2) \overset{def}{=} x$ and the resulting i-tree will be such that $x \in \overline{\alpha}(S_1) \cap \overline{\alpha}(S_2)$.

**Extending the model with $\Xi$.** Let $(\Delta,\alpha) \models T$. Then for each $P \in \text{PN}$, define

$$\Xi(P) = \bigcup \{\overline{\alpha}(S) \mid S \in S \land P \in \Phi(S)\}$$

Then $(+P) \in \Phi(S) \Rightarrow \overline{\alpha}(S) \subseteq \Xi(P)$ holds by construction, it remains to show $(-P) \not\in \Phi(S) \Rightarrow \overline{\alpha}(S) \not\subseteq \Xi(P)$ for every node $S$. Consider $S \in S$ such that $(-P) \not\in \Phi(S)$. If $S_1 = 0$, then $\overline{\alpha}(S_1) = \emptyset$, so the condition trivially holds. Similarly, if $(+P) \not\in \Phi(S_1)$, then $\overline{\alpha}(S_1) = \emptyset$ and the condition holds. Otherwise, assume $(+P) \in \Phi(S_1)$. For the sake of contradiction suppose that there exists an element $x \in \overline{\alpha}(S_1)$, $x \in \Xi(P)$. By definition of $\Xi(P)$, there exists a node $S_2 \neq S_1$ such that $x \in \overline{\alpha}(S_2)$ and $(+P) \not\in \Phi(S_2)$. By the condition on independent signatures, one of the following two cases applies.

1. $\text{disj}_{\overline{\alpha}}(S_1, S_2).$ Then $\overline{\alpha}(S_1) \cap \overline{\alpha}(S_2) = \emptyset$ by the semantics of i-diagrams, which is a contradiction with $x \in \overline{\alpha}(S_1)$ and $x \in \overline{\alpha}(S_2)$.
2. $S_1$ and $S_2$ have compatible signatures. Then there exists a node $S$ such that $S_1 \sim S, S_2 \sim S$ and $\text{Sig}(S_1) \cap \text{Sig}(S_2) \subseteq \text{Sig}(S)$. Because $(-P) \not\in \Phi(S_1)$, $(+P) \in \text{Sig}(S_1)$, and because $(+P) \not\in \Phi(S_2)$, $(+P) \in \text{Sig}(S_2)$. Therefore $P \in \text{Sig}(S)$.

We have reached the contradiction in each case, so we conclude $\overline{\alpha}(S_1) \not\subseteq \Xi(P)$.

**A.4 Details of the Proofs for Subsumption Completeness**

**A.4.1 Refinements of Lemma 6**

According to the remarks in the proof of Lemma 6 if an i-tree $T$ is weakly consistent, there exists a choice of cardinalities $K : S \to N$, such that we can build a model $(\Delta, \alpha, \Xi)$ for $T$ with the property $\overline{\alpha}(S) = K(S)$ for all $S \in S$. For a fixed choice of cardinalities $K$, we can, in certain cases, enforce some additional properties by choosing which element we merge in the steps 1 and 3 of the construction. The following Lemmas are based on this idea.

**Lemma 18** (Non-empty intersection (1)). Let $S_1$ and $S_2$ be nodes in a weakly consistent i-tree $T$ such that

$$\text{Clfn}(S_1) > 0 \land S_1 \not\sim S_1' \land S_1' \subsetneq Q_1 \land \text{Clfn}(S_2) > 0 \land S_2 \not\sim S_2' \land S_2' \subsetneq Q_2$$

for some $S, S_1, S_2 \in S$, $Q_1, Q_2 \in \text{Split}(S)$ where $Q_1 \neq Q_2$ and $-(3C \in \text{Comp}(S))$. Then there exists a
Lemma 19 (Non-empty intersection (2)). Let $S_1$ and $S_2$ be nodes in a weakly consistent i-tree $T$ such that

$$\overline{\text{CInf}}(S_1) \cap \overline{\text{CInf}}(S_2) \neq \emptyset.$$ 

Proof. We use the construction described in Lemma 8 with the exception of step 3 of the construction of the model $\mathcal{M}_T$ for the subtree $T_S$ with root $S$, where we do the following:

- We choose an element $x_1 \in \overline{\text{CInf}}(S_1)$ in the model $\mathcal{M}_{T_S}$ built for $T_S$. (We know there exists one such $x_1$ because $\text{CInf}(S_1) > 0$)
- Analogously, we choose an element $x_2 \in \overline{\text{CInf}}(S_2)$ in the model $\mathcal{M}_{T_S}$ built for $T_S$.
- We choose the bijections $\sigma_{Q_1}$ and $\sigma_{Q_2}$ in step 3 such that $\sigma_{Q_1}(x_1) = \sigma_{Q_2}(x_2)$. \n
LEMMA 19 (Non-empty intersection (2)). Let $S_1$ and $S_2$ be nodes in a weakly consistent i-tree $T$ such that

$$\overline{\text{CInf}}(S_1) \cap \overline{\text{CInf}}(S_2) \neq \emptyset.$$ 

Proof. We use the construction described in the proof of Lemma 8 except for the step 1 of the construction of the model $\mathcal{M}_T$ for the subtree $T_S$ with root $S$, for which we do the following. From $K(S) \leq \text{CSup}(S) < \Sigma(\text{CSup}(C))$, we conclude that the choice of cardinalities $t_Q$ for $Q \in C$ in the proof of Lemma 8 is such that $\Sigma_{Q \in C} t_Q > K(S)$, by considering two cases.

1. There exists $Q \in C$ such that

$$t_Q = \min(K(S), \Sigma(\text{CSup}(Q))) = K(S).$$

Then we choose $Q_i \in \{Q_1, Q_2\}$ such that $Q_i \neq Q$. Since $\text{CInf}(S_1) > 0 \land S_1 \nrightarrow S_1' \nrightarrow S$ and repeatedly applying $C_4$ and using $C_5$, we have

- $\Sigma(\text{CSup}(Q_i)) \geq \text{CSup}(S') \geq \text{CInf}(S') \geq \text{CInf}(S_1) > 0$
- $K(S) \leq \text{CInf}(S') \leq \text{CInf}(S_1) > 0$

As a consequence $t_{Q_i} = \min(K(S), \Sigma(\text{CSup}(Q_i))) > 0$ and

$$\Sigma_{Q \in C} t_Q \geq t_{Q_i} + t_{Q_i} > K(S).$$

2. For all $Q \in C$, $t_Q = \Sigma(\text{CSup}(Q))$. Then $\Sigma_{Q \in C} t_Q = \Sigma(\text{CSup}(C)) > \text{CSup}(S) \geq K(S).$

Because $\Sigma_{Q \in C} t_Q > K(S)$, we can apply Remark 8 and choose one element $x_1 \in \overline{\text{CInf}}(S_1)$ in the model $\mathcal{M}_{T_S}$ built for $T_S$, and an element $x_2 \in \overline{\text{CInf}}(S_2)$ in the model built for $T_{S_2}$, and decide to merge these elements in the step 1 of the construction. We know that such elements $x_1, x_2$ exist because $\text{CInf}(S_1) > 0$ and $\text{CInf}(S_2) > 0$. \n
LEMMA 20 (Isolated element). Let $T$ be a weakly consistent i-tree and $(Q_1, \nrightarrow)$ a subtree of $(S, \nrightarrow)$ with the same root $S_R$ as $T$, such that for all $S \in Q_1$ all the following conditions hold:

1. $\text{CInf}(S) > 0$
2. $\forall C \in \text{Comp}(S) \exists S' \in \text{Split}(S). Q \subseteq C \land |Q \cap Q_1| = 1$
3. $\forall Q \in \text{Split}(S) Q \cap Q_1 \neq \emptyset \Rightarrow \text{CInf}(Q) < \text{CInf}(S)$.

Then we can construct a model for $T$ such that

$$\exists \alpha \in \overline{\text{CInf}}(R) \forall S \in S. (x \in \overline{\text{CInf}}(S) \Leftrightarrow S \in Q_1)$$

Proof. We use a variation of the construction in the proof of Lemma 8. We apply the assumptions about the subtree $Q_1$ to show that we always have enough “slack” to avoid merging one specific element from $Q_1$ with the elements of neighbors. We being by describing a slightly modified Step 1 of the proof of Lemma 8.

Step 1’ (for nodes of $Q_1$). Consider the root $S_R \in Q_1$. Let $C \in \text{Comp}(S_R)$, and $C \nsubseteq \text{Split}(S_R)$ such that $C = \text{UC}_C$. By construction, there exists a unique node $S' \in Q_1$ of $S_R$ and a corresponding split view $Q'$ such that $S' \in Q'$ and $C = \{Q'\} \cup C_0$. We define $t_{Q'} \leftarrow \min(k, \Sigma(\text{CSup}(Q')))$. Then for each $Q_0 \in C_0$, we define $t_{Q_0} = \min(k-1, \Sigma(\text{CSup}(Q_0)))$. For each $Q_0 \in C_0$, since $k \geq \text{CInf}(S_R)$ by choice of $k$ and $\text{CInf}(S_R) > \Sigma(\text{CInf}(Q_0))$ by hypothesis 3 on $T$, we have $\Sigma(\text{CInf}(Q_0)) \leq t_{Q_0} \leq \Sigma(\text{CSup}(Q_0))$ (H1) and the property (H1) also holds for $t_{Q'}$, because $t_{Q'}$ is defined as in Lemma 8.

By definition of $t$ we clearly have $\max_{Q \in C} t_Q \leq k$. We next show $\forall Q \in C, t_Q \geq k$ by considering the following cases.

- $\forall Q \in C, t_Q = k$. Then the claim is obvious.
- For all $Q \in C$ we have $t_Q = \Sigma(\text{CSup}(Q))$. The claim follows from $C_3$ and the choice of $k$ because $\Sigma t_Q \geq \Sigma(\text{CSup}(C)) \geq \text{CSup}(S_R) \geq k$.
- There exists $Q_0 \in C_0$ such that $t_{Q_0} = k - 1$. Using $\Sigma(\text{CSup}(Q_0)) \geq \text{CSup}(Q') > 0$ and $k \geq \text{CInf}(S_R) > 0$, we obtain $t_{Q'} > 0$, so

$$\Sigma t_Q \geq t_{Q_0} + t_{Q'} \geq (k - 1) + 1 \geq k.$$ 

We finally obtain

$$\max_{Q \in C} t_Q \leq k \leq \Sigma t_Q \leq \Sigma_{Q \in C} t_Q$$

(H2)

By definition of all $t_{Q_0}$ we then have

$$\forall Q_0 \in C_0. t_{Q_0} < k$$

(H3)

According to Remark 8, H3 allows us to choose an element of the model $\mathcal{M}_{S'}$ constructed for the subtree $T_{S'}$, and decide not to merge it with any other element. This observation allows us to recursively enforce $x \in \overline{\text{CInf}}(S) \Leftrightarrow S \in Q_1$.

Indeed, consider a node $S_R \in Q_1$ and let $\{S_1, \ldots, S_k\} = \text{Sons}(S_R) \cap Q_1$ be its sons in $Q_1$. For each i, we can then recursively ensure $x_i \in \overline{\text{CInf}}(S) \Leftrightarrow S \in Q_i$ for each $S_i$ in the $S_R$ subtree i.e. for each $S$ for which $S \nrightarrow S_R$. By definition of $Q_0$, each $S_R$ is in a different complete view, so we can apply bijection to the submodels (Remark 8) and let $\sigma_i(x_i) = \ldots = \sigma_i(x_R)$ = $x$.

We ensure that $x$ does not belong to any subtree rooted at a node $S_0 \in \text{Sons}(S_R) \setminus Q_1$, using Remark 8 to make sure that $x$ is not merged with any of the elements of $\overline{\text{CInf}}(S_0)$, which is possible thanks to H3. Finally, for the base case, when $S$ has no sons, we pick $x$ to be a fresh element, which is possible by assumption 3 on the subtree $Q_1$, as noted in Remark 8.

A.4.2 Links between weak and strong consistency

Lemma 8 (Bounds Refinement) Let $T$ be a strongly consistent i-tree, $S \in S$, $i, s$ such that $\text{CInf}(S) \leq i \leq s \leq \text{CSup}(S)$, let $T' = T[\text{CInf}(S) \leftarrow i, \text{CSup}(S) \leftarrow s]$ and $T_{\text{ref}} = T'_{\text{ref}}(T')$. Then

1) $T_{\text{ref}} \neq T_{\text{ref}} \leftarrow i, 2) T_{\text{ref}} \leftarrow T$, and 3) if $(S \nrightarrow S_0)$, then

$$R_{\text{ref}}(S_0) = \text{CSup}(S_0) = \text{CSup}_{\text{ref}}(S_0).$$

Proof. We prove this result by induction on the depth of $S$ in the tree $(S, \nrightarrow)$. The key step of this proof is to show that the application of $\text{UpSup}$ and/or $\text{UpInf}$ to the father $S'$ of $S$ do not produce
a situation where \(a_5\) holds in the resulting diagram \(T''\) (and therefore the rule Error is not applicable in \(T'\)). We distinguish three different cases:

- When \(\text{UpSup}\) and \(\text{UpInf}\) are both applicable to this node \(S'\) we have \(\text{Clnf}'(S') \leq \text{CSup}''(S')\) because for \(C \in \text{Comp}(S'), Q \in \text{Split}(S')\) such that \(Q \subseteq C, Q = \{S' \uplus Q_0, C = \{S\} \uplus C_0\) and \(T' \uplus C \subseteq T'\) we have

  \[
  \text{Clnf}'(S') = \Sigma(\text{Clnf}'(Q_0)) + \text{Clnf}'(S) \quad \text{(by \(b_4\)}
  \]

- When only \(\text{UpSup}\) is applicable to this node \(S'\) we have \(\text{Clnf}'(S') \leq \text{CSup}'(S')\) because for \(C \in \text{Comp}(S'), C = \{S\} \uplus C_0\) and \(T' \uplus C \subseteq T''\) we have

  \[
  \text{CSup}'(S') = \Sigma(\text{CSup}'(C_0)) + \text{CSup}'(S) \quad \text{(by \(b_3\)}
  \]

- When only \(\text{UpInf}\) is applicable to this node \(S'\) we have \(\text{Clnf}'(S') \leq \text{CSup}'(S')\) because for \(Q \in \text{Split}(S'), Q = \{S' \uplus Q_0\) and \(T' \uplus C \subseteq T''\) we have

  \[
  \text{CSup}'(S') = \Sigma(\text{CSup}'(Q_0)) + \text{Clnf}'(S) \quad \text{(by \(b_3\)}
  \]

2. Follows easily from the hypothesis \(\text{Clnf}(S) \leq i \leq s \leq \text{CSup}(S)\) and the fact that \(R_{\text{BF}}^{\text{ne}}\) is semantics preserving.

3. It is enough to notice that only rules \(\text{UpInf}\) and \(\text{UpSup}\) are used when applying \(R_{\text{BF}}^{\text{ne}}\), and these rules are applied in the bottom-up direction.

\[\text{Lemma 2} (\text{Parallel Bounds Refinement (1)})\]

Let \(T\) be a strongly consistent i-tree, and \((Q_0, \ldots)\) a sub-tree of \(T\) which has the same root as \(T\) and is such that

- The nodes of \(Q_0\) are pairwise independent, that is

\[
\forall S_1, S_2 \in Q_0, (\neg \text{disj}_T(S_1, S_2))
\]

Then the i-tree \(T'\) defined by

\[
T' \overset{\text{def}}{=} T[\forall S \in Q_0, \text{Clnf}(S) \rightarrow \text{CSup}(S)]
\]

is such that its \(R''\) normal form \(T_{\text{BF}}^{\text{ne}} \overset{\text{def}}{=} R_{\text{BF}}^{\text{ne}}(T')\) satisfies

1. \(T_{\text{BF}}^{\text{ne}} \neq \bot_d\)
2. \(T_{\text{BF}}^{\text{ne}} \models T\)
3. \(\forall S \in Q_0, \forall Q \in \text{Split}(S_0), Q \cap Q_0 = \emptyset \Rightarrow \Sigma(\text{Clnf}_T(Q)) < \text{Clnf}_T(S)\)

\[\text{Proof}\]

If we apply \([\text{Clnf}'(S') \rightarrow \text{CSup}'(S')]\) to every node \(S'\) of \(Q_0\), starting from the root to the leaves, we always maintain \(C_1, C_2, C_3\) because \(\Phi\) and \(\text{CSup}\) are never modified. We also maintain \(C_1\) because for each \(S \in Q_0\) and each view \(Q \in \text{Split}(S)\) such that there exists \(S' \in Q \cap Q_0\), by \(\neg \text{disj}_T(S_1, S_2)\), we know that \(S'\) is the only modified node, and

\[
\text{Clnf}'(S) = \text{CSup}(S) \geq \Sigma(\text{Clnf}'(Q - \{S'\}) + \text{CSup}'(S')) \quad \text{(by \(C_7\)}
  \]

Therefore, \(T'\) is already in normal form, so \(T_{\text{BF}}^{\text{ne}}\) is identical to \(T'\) and is clearly distinct from \(\bot_d\), proving condition 1. Condition 2 holds because \(T' \models T\) because the cardinality bounds in \(T'\) are at least as strong as in \(T\). Condition 3 holds because

\[
\Sigma(\text{Clnf}'(Q)) = \Sigma(\text{Clnf}(Q)) \quad \text{(because \(Q \cap Q_0 = \emptyset\)}
\]

\[
< \text{CSup}(S) \quad \text{(by \(C_4\)}
\]

\[
= \text{Clnf}'(S) \quad \text{(by definition of \(T'\)}
\]

\[\text{Lemma 21 (Parallel Bounds Refinement (2))}\]

Let \(T\) be a strongly consistent i-tree and \(S_1, S_2, S_1', S_2'\) are such that \(C \in \text{Comp}(S), Q_1, Q_2 \in \text{Split}(S), S_1 \overset{\text{def}}{=} S_1' \land S_1' \subseteq Q_1 \land Q_1 \subseteq C\) and \(S_2 \overset{\text{def}}{=} S_2' \land S_2' \subseteq Q_2 \land Q_2 \subseteq C\) \(Q_1 \neq Q_2\)

Define

\[
T' \overset{\text{def}}{=} T[\forall S', S_1, S_1' \overset{\text{def}}{=} S' : \text{Clnf}(S') \leftarrow \text{CSup}(S')] \quad \forall S', S_1, S_1' \overset{\text{def}}{=} S' : \text{Clnf}(S') \leftarrow \text{CSup}(S')]
\]

\[
T_{\text{BF}}^{\text{ne}} \overset{\text{def}}{=} R_{\text{BF}}^{\text{ne}}(T')
\]

Then \(T'' \neq \bot_d, T'' = T\) and \(\text{CSup}'(S) < \Sigma(\text{CSup}'(C))\).

\[\text{Proof}\]

As in the proof of \text{Lemma 2}, weak consistency conditions hold in \(T''\) for all nodes \(S\) such that \(S_1 \overset{\text{def}}{=} S_1' \land S_1' \subseteq S_2 \land S_2 \subseteq C\). Therefore, the only rewrite step that mat be applicable in \(T''\) is the application of \(\text{UpInf}\) to \(S\). This application may lead to applications of other instances of \(\text{UpInf}\), but the proof of \text{Lemma 6} shows that this process will result in a weakly consistent i-tree, so \(T_{\text{BF}}^{\text{ne}} \neq \bot_d\). Moreover, the process of computing \(T_{\text{BF}}^{\text{ne}}(\text{CSup}'(S) \leftarrow \text{Clnf}(S))\) is identical to applying \text{Lemma 8} to \(T'\) with bounds \(i = s = \text{Clnf}(S), S\) and therefore leads to a weakly consistent i-tree \(T''\), so \(T'' \neq \bot_d\).

The condition \(T'' \models T\) follows because \(R_{\text{BF}}^{\text{ne}}\) is semantics preserving, and the updates of trees only shrink the bounds on nodes, so they convert a diagram into a stronger one.

To prove \(\text{CSup}'(S) < \Sigma(\text{CSup}'(C))\), observe first that \(\text{CSup}'(S) = \text{Clnf}_{\text{BF}}(S)\) by definition of \(T''\), and \(\Sigma(\text{CSup}'(C)) = \Sigma(\text{CSup}(C))\) because \(\text{CSup}\) does not change for any ancestors of \(S\). Therefore, it suffices to show

\[
\text{Clnf}_{\text{BF}}(S) < \Sigma(\text{CSup}(C))
\]

We prove this condition by distinguishing two cases.

1. \(\text{UpInf}\) is not applicable to \(S\). Then \(\text{Clnf}_{\text{BF}}(S) = \text{Clnf}(S)\) and the condition follows by \(C_9\).

2. \(\text{UpInf}\) is applicable to \(S\). Then for some \(a, b\) where \(\{a, b\} = \{1, 2\}\) we have

\[
\text{Clnf}_{\text{BF}}'(S) = \Sigma(\text{Clnf}'(Q_a)) \quad \Sigma(\text{Clnf}'(Q_b)) \quad \Sigma(\text{Clnf}'(C)) \quad \Sigma(\text{CSup}(C))
\]

\[\text{A.4.3 Completeness of the algorithm Subsumes}\]

\[\text{Theorem 4}\]

Let \(T\) be a strongly consistent i-tree and let \(H_{\text{ne}}^A\) for atomic formula \(A\) be as defined in \text{Figure 2}. Then \(H_{\text{ne}}^A\) if and only if \(T \models A\).

The \((\Rightarrow)\) direction of \text{Theorem 4} is trivial by the semantics of i-diagrams. For \((\Leftarrow)\) direction we prove the following characteriza-
On Algorithms and Complexity for Sets with Cardinality Constraints

Lemma 15

If an i-tree \( T \) is strongly consistent, \( S_0, \in \mathbb{S}, C \in \mathcal{P} (\mathbb{S}) \), and Included \((S_0, C, T)\) returns false, then

\[ \exists M. \, \mathfrak{m} (S) \not\subseteq \bigcup \mathfrak{m} \left[ C \right] \]

Proof. Let \( Q_0 \) be the smallest set of nodes such that:

\[ S_0 \not\subseteq S, \quad S \in Q_0 \quad \Rightarrow S \in Q_0 \]

\[ S \in Q_0 \wedge Q_1 \in \text{Comp}(S) \wedge S \equiv S \quad \Rightarrow S \in Q_0 \]

By definition of \text{Incl} we know that \( Q_0 \cap C = \emptyset \).

\( Q_0 \) is tree-shaped by construction, but may contain two nodes which are explicitly disjoint. We compute a subtree \( Q_1 \) of \( Q_0 \), by starting from the root and keeping each time at most one son for each complete view. We also impose that \( Q_1 \) contains \( S \), by avoiding to cut the branch which leads to \( S \).

We then define

\[ T' = \text{Rinf}(T \mid \forall S \in Q_1 : \text{Clnf}(S) \leftarrow \text{CSup}(S)) \]

According to Lemma 2 (Parallel Bounds Refinement) we have 

\[ T' \not\equiv T \]

We then apply Lemma 20 to construct a model for the weakly consistent i-tree \( T' \) such that

\[ \forall S' \in \mathbb{S}. \, (x \in \mathfrak{m} (S') \Rightarrow S' \in Q_1) \]

for some element \( x \in \mathfrak{m} (S) \). Because \( S \in Q_1 \), we have \( x \notin \bigcap \mathfrak{m} (C) \).

Lemma 16

If an i-tree \( T \) is strongly consistent, then for all \( S \in \mathbb{S} \) and \( P \in \mathbb{P} \) we have

\[ (+P) \not\in \Phi (S) \Rightarrow \exists M. \, M \equiv T \wedge \mathfrak{m} (S) \not\subseteq \mathfrak{m} (P) \]

\[ (-P) \not\in \Phi (S) \Rightarrow \exists M. \, M \equiv T \wedge \mathfrak{m} (S) \not\subseteq \mathfrak{m} (P) \]

Proof. Let \( S \in \mathbb{S} \) be such that \((+P) \not\in \Phi (S)\). We define \( Q_{+P} \triangleq \{ S' \in \mathbb{S} | (+P) \not\in \Phi (S') \} \). Using \( C_{10} \) we show that \text{Included}(S, Q_{+P}, T) \) cannot return true. Then, there exists a model such that \( \mathfrak{m} (S) \not\subseteq \bigcup \mathfrak{m}(Q_{+P}) \). We then change the model by redefining \( \Xi' \) by:

\[ \forall P \in \mathbb{P}, \Xi' (P) = \bigcup \{ \mathfrak{m} (S') | S' \in Q_{+P} \} \]

to ensure that \( \Xi' (P) \not\subseteq \mathfrak{m} (S) \). The case of \(-P) \not\in \Phi (S)\) is analogous by taking a model such that \( \mathfrak{m} (S) \not\subseteq \bigcup \mathfrak{m}(Q_{-P}) \) and redefining \( \Xi' \) by:

\[ \forall P \in \mathbb{P}, \Xi' (P) = \Delta - \bigcup \{ \mathfrak{m} (S') | S' \in Q_{-P} \} \]

Lemma 17

If an i-tree \( T \) is strongly consistent, then for all \( S_1, S_2 \in \mathbb{S} \) such that \( S_1 \not\subseteq S_2 \) and \( S_2 \not\subseteq S_1 \) we have

\[ \neg \left( \text{disj}_T^\ast (S_1, S_2) \right) \Rightarrow \exists M. \, \mathfrak{m} (S_1) \cap \mathfrak{m} (S_2) \not\subseteq \emptyset \]

Proof. Let \( S_1, S_2 \in \mathbb{S} \) such that \((\neg \text{disj}_T^\ast (S_1, S_2))\). If \( S_1 = S_2 \) we can find a model \( M \) where \( \mathfrak{m} (S_1) = \mathfrak{m} (S_2) \not\subseteq \emptyset \) by Lemma 10 in this model \( \mathfrak{m} (S_1) \in \mathfrak{m} (S_2) = \mathfrak{m} (S_2) \not\subseteq \emptyset \). Suppose \( S_1 \neq S_2 \). Define \( S_1', S_2', S_0 \) as the unique nodes such that \( S_0 \) is the least common ancestor of \( S_1 \) and \( S_2 \) in \( T \), and \( S_1', S_2' \in \text{Sons} (S_0) \) are the ancestors of \( S_1 \) and \( S_2 \), respectively. We distinguish two cases:

\[ \bullet \, S_1' \text{ and } S_2' \text{ do not belong to a same complete view of } S_0. \text{ Then apply Lemma 20 on the subtree} \]

\[ Q_0 = \{ S \in \mathbb{S} | S_1 \not\subseteq S \text{ and } S_2 \not\subseteq S \} \]

whose nodes are pairwise independent by the hypothesis \( \text{disj}_T^\ast (S_1, S_2) \). The resulting tree \( T'_{\text{inf}} \) satisfies the hypothesis of Lemma 15 so there exists a model \( M = (\Delta, \alpha, \Xi) \) for \( T'_{\text{inf}} \) such that

\[ \mathfrak{m} (S_1) \cap \mathfrak{m} (S_2) \not\subseteq \emptyset \]

\[ \bullet \, S_1' \text{ and } S_2' \text{ belong to a same complete view } C. \text{ Define} \]

\[ T''_{\text{inf}} (S_1 \not\subseteq S \text{ and } S_2 \not\subseteq S) \]

\[ \{ \forall S' \in \mathbb{S} | \text{Clnf}(S') \leftarrow \text{CSup}(S') \} \]

By Lemma 21 then \( T''_{\text{inf}} \leftarrow T \) and \( \text{CSup}''(S) < \Sigma (\text{CSup}'(C)) \). This last property allows us to apply Lemma 19 and prove the existence of a model \((\Delta, \alpha, \Xi)\) for \( T''_{\text{inf}} \) such that \( \mathfrak{m} (S_1) \cap \mathfrak{m} (S_2) \not\subseteq \emptyset \).

B. Example transformation to CBAC constraints

We illustrate the idea of the algorithm in Figure 2 through an example of checking the validity of the formula

\[ |A \cup B| = |A| + |B| - |A \cap B| \]

that is, checking the unsatisfiability of the formula

\[ |A \cup B| \neq |A| + |B| - |A \cap B| \]

This formula has no integer variables initially, so the first step does not apply. Furthermore, all set algebra expressions appear already within cardinality constraints, so step 2 is unnecessary as well, as is step 3 because there are no divisibility constraints.

In step 4, we introduce a non-negative integer variable for each cardinality term, yielding the system:

\[ i_{A \cup B} \neq i_A + i_B - i_{A \cap B} \]

\[ i_1 = \text{max} \]

\[ |A \cup B| = i_{A \cup B} \]

\[ |A| = i_A \]

\[ |B| = i_B \]

\[ |A \cap B| = i_{A \cap B} \]

Because the formula is already in the form of a conjunction, there is no need to non-deterministically guess the conjunction in step 5, and we continue with the current constraints.

In step 6, we non-deterministically replace \( i_{A \cup B} \neq i_A + i_B - i_{A \cap B} \) with \( i_{A \cup B} + 1 \leq i_A + i_B - i_{A \cap B} \) or \( i_A + i_B - i_{A \cap B} + 1 \leq i_{A \cup B} \) and we illustrate the first case (the other case needs to be checked as well unless the satisfying assignment is found in the
In step 8, we identify \( m \) and \( n \) such that
\[
M = \text{max}(m, n)
\]
In step 7, we introduce a slack variable \( i_0 \) to eliminate the first inequation, and transform the equation to normal form:
\[
i_{A \cup B} - i_A - i_B + i_{A \cap B} + i_0 = -1
\]
Therefore, if there exists a solution to the system of equations, there exists a solution that can be represented by at most 32 bits in binary.

In step 8, we identify \( v = (\text{MAXC}, i_{A \cup B}, i_A, i_B, i_{A \cap B}, i_0) \) and have \( A = \{0, 1, -1, -1, 1, 1\} \) and \( d = (-1) \). We have \( m_0 = 1, m_1 = 5, n_0 = 6 \), and \( S = 2 \). Therefore, \( m = 6, n = 6, \) and \( a = 1 \). From these values we compute \( M = 6 \cdot (6 \cdot 1)^{11} = 6^{12} < 2^{32} \).

Therefore, if there exists a solution to the system of equations, there exists a solution that can be represented by at most 32 bits in binary.

In step 9, we guess non-deterministically a solution vector bounded by \( M \) that satisfies the first equation (that is, \( Ak = -1 \)). One such solution is \( k = (101, 100, 77, 40, 10, 6) \) (written in decimal notation). For this guessed solution, we generate the CBAC constraint:
\[
|1| = 101
\]
\[
|A \cup B| = 100
\]
\[
|A| = 77
\]
\[
|B| = 40
\]
\[
|A \cap B| = 10
\]
This example CBAC constraint does not have a solution, and neither do the remaining examples generated by the non-deterministic algorithm. The search tree corresponding to the non-deterministic algorithm returns false in all branches, so the formula is unsatisfiable, and its negation is valid.

We note that, in this case, the dimensions of the system of equations are such that the estimate \( M \) can be improved if we consider the system of equations where the variables \( \text{MAXC}, i_{A \cup B}, i_A, i_B, i_{A \cap B} \) are all substituted into the the original integer part of the QFBAPA problem. In general, this alternative estimate is given by \( M' = n'(m'a')2^{m'+1} \) where
\[
m' = m_0
\]
\[
n' = \max(n_0 - m_1, 2^S)
\]
\[
a' = \max \left( \max_{1 \leq q \leq m_0} |d_q|, \max_{p=1}^{m_0} \max_{q \in R} |a_{pq}|, \max_{p=1}^{m_0} \max_{q \in Q} \frac{\sum_{a_{pq} > 0} a_{pq}}{a_{pq} < 0} (-a_{pq}) \right)
\]
where \( Q = \{p_1, \ldots, p_{m_1}\} \) are indices of variables denoting cardinalities (and that are being substituted), and \( R = \{1, \ldots, n_0\} \setminus R \).

In our example,
\[
m' = 1
\]
\[
n' = 4
\]
\[
a' = \max(1, 1, 2)
\]