TECHNISCHE UNIVERSITÄT MÜNCHEN
Lehrstuhl für Informatik II

Normalization of Horn Clauses with Disequality Constraints

Andreas Reuß

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Alois Knoll
Prüfer der Dissertation:	1. Univ.-Prof. Dr. Helmut Seidl
	2. Univ.-Prof. Dr. Dr. h.c. Javier Esparza

Die Dissertation wurde am 25.10.2012 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 02.05.2013 angenommen.

Abstract

Horn clauses constitute a convenient, Turing-powerful tool for the specification and representation of analysis problems. However, it is not possible to express disequality conditions directly and in a natural way by means of Horn clauses. This thesis therefore proposes an extension of Horn clauses with several kinds of disequality constraints.

In order to provide an automatic analysis framework capable of expressing explicit disequality conditions, the considered class of Horn clauses is restricted to the class $\mathcal{H}_1$, for which the satisfiability problem is decidable. $\mathcal{H}_1$ is chosen as a large and expressive class which can approximate in a natural way any finite set of Horn clauses, making it particularly useful for approximative analyses of term-manipulating programs based on abstract interpretation. We build on an existing normalization procedure that transforms finite sets of $\mathcal{H}_1$-clauses to equivalent tree automata, and show that this procedure can be adapted such that finite sets of $\mathcal{H}_1$-clauses extended with disequality constraints are normalized into equivalent tree automata with disequality constraints.

Since emptiness is decidable for these automata, this reduction proves satisfiability decidable for $\mathcal{H}_1$-clauses extended with disequality constraints. In particular, we provide corresponding proofs for the following extensions: disequalities between terms, disequalities between subterms specified by *paths*, and hom-disequalities, i.e., disequalities between images of terms under a given tree homomorphism. Such hom-disequalities allow to test whether two terms are distinct modulo a semantic interpretation, enabling to neglect information that is considered irrelevant for the intended comparison.

We furthermore provide an extension of tree automata with term equality and disequality constraints, generalizing the known class of tree automata with constraints between brothers of Bogaert and Tison from 1992, enjoying full Boolean closure, but still having a decidable emptiness problem — whereas emptiness for automata with *path equalities* has been shown undecidable by Mongy in 1981.

The four main contributions of the thesis are published in [SR11], [SR12], [RS12], and [RS10], respectively.

Zusammenfassung

Hornklauseln sind ein geeignetes, turingmächtiges Werkzeug zur Modellierung von Analyse-Problemen. Es ist jedoch nicht möglich, mit Hornklauseln direkt und auf natürliche Weise Ungleichheitsbedingungen auszudrücken. Dieses Werk schlägt daher eine Erweiterung von Hornklauseln mit unterschiedlichen Arten von Ungleichheits-Constraints vor.

Um ein automatisches Analyse-Framework zu erreichen, mit dessen Hilfe es möglich ist, explizite Ungleichheiten auszudrücken, schränken wir die betrachtete Klasse von Hornklauseln ein auf die Klasse $\mathcal{H}_1$, für die das Erfüllbarkeits-Problem entscheidbar ist. Diese Klasse ist sehr ausdrucksstark und kann auf natürliche Weise beliebige Hornklauseln approximieren. Aufgrund dieser Eigenschaft ist $\mathcal{H}_1$ insbesondere für auf der Technik der abstrakten Interpretation basierende approximative Analysen von Programmen, die Terme verarbeiten, sehr nützlich.

Wir bauen auf einem bereits bestehenden *Normalisierungs*-Verfahren auf, welches endliche Mengen von $\mathcal{H}_1$-Klauseln in einen dazu äquivalenten Baum-Automaten überführt, und passen dieses Verfahren dergestalt an, dass es endliche Mengen von $\mathcal{H}_1$-Klauseln mit Ungleichheits-Constraints in einen dazu äquivalenten Baum-Automaten mit solchen Ungleichheits-Constraints überführt.

Da das Leerheits-Problem für solche Automaten entscheidbar ist, wird durch diese Reduktion gezeigt, dass das Erfüllbarkeits-Problem für $\mathcal{H}_1$-Klauseln mit Ungleichheits-Constraints entscheidbar ist. Für die folgenden Erweiterungen beweisen wir auf diese Weise Entscheidbarkeit: Ungleichungen zwischen beliebigen Termen, Ungleichungen zwischen beliebigen Subtermen, welche durch *Pfade* spezifiziert werden, und Ungleichungen zwischen homomorphen Bildern von Termen. Die letzteren beiden ermöglichen es, beim Test auf Ungleichheit Teilterme unberücksichtigt zu lassen.

Wir führen des Weiteren eine Erweiterung von Baum-Automaten mit *Gleichheits*- und Ungleichheits-Constraints für Terme ein, welche die bekannte Klasse der Automaten mit *constraints between brothers* von Bogaert und Tison aus dem Jahr 1992 verallgemeinert, wie diese unter Booleschen Operationen für Baum-Sprachen abgeschlossen ist, und für deren Leerheits-Problem wir überraschenderweise Entscheidbarkeit beweisen können – wohingegen etwa für Automaten mit Pfad-Gleichheits-Constraints schon 1981 durch Mongy die Unentscheidbarkeit des Leerheits-Problems gezeigt wurde.

Contents

1 Introduction **1**

2 Horn Clauses and Constraints **13**
2.1 Basics 13
2.2 Tree Automata with Term Constraints 23
2.2.1 Generalized Tree Automata with Term Disequalities . . . 24
2.2.2 Tree Automata with Term Equalities and Disequalities . . 26
2.3 Complex Preconditions, Auxiliary Variables 34
2.4 Tree Automata with Hom-Equalities 36

3 Normalization of Horn Clauses **39**
3.1 $\mathcal{H}_1$-Clauses with Term Disequalities 39
3.1.1 Bounding the Number of Terms 40
3.1.2 Normalizing Constrained $\mathcal{H}_1$-Clauses 42
3.2 $\mathcal{H}_1$-Clauses with Path Disequalities 51
3.2.1 Increased Expressiveness 51
3.2.2 Challenges in Computing with Subterms 54
3.2.3 Termination-sensitive Resolution in Presence of Paths . . 55
3.2.4 Splitting Paths 59
3.3 $\mathcal{H}_1$-Clauses with Hom-Disequalities 65
3.3.1 Tree Automata with Hom-Disequalities 66
3.3.2 Expressiveness 71
3.3.3 $\mathcal{H}_1$-Normalization modulo Tree Homomorphism 73

4 Perspectives **79**

Chapter 1

Introduction

First-order Horn[1] clauses serve as a convenient tool for representing analyses of term-manipulating programs. Horn clauses are used in the areas of automated deduction, theorem proving, and logic programming, e.g., in Prolog. A Prolog program is a set (put logically: a conjunction) of definite Horn clauses together with a query. Consider as an example the following set of definite Horn clauses:

$$\begin{aligned} human(socrates) &\Leftarrow \\ mortal(X) &\Leftarrow human(X) \\ human(X) &\Leftarrow ancestor(f(X,Y)), human(Y) \\ ancestor(f(sophroniskos, socrates)) &\Leftarrow \end{aligned}$$

Here, the right-hand sides are the preconditions of the clauses, and the left-hand sides are the conclusions. Conclusions consist of single literals, while preconditions are comma-separated lists of literals. These lists might also be empty. The meaning of a precondition is given by the conjunction over the occurring literals. "$\Leftarrow$" denotes the logical implication (from the right to the left). The variables, denoted by the upper-case letters, are meant to be universally quantified in each clause. Besides variables, the terms occurring in literals may also contain constants and data constructors. The literal in the fourth clause, e.g., contains the data constructor f (abbreviating "father") and the constants $sophroniskos$ and $socrates$. This clause allows to conclude, from an empty precondition, that the predicate $ancestor$ holds for the composed term $f(sophroniskos, socrates)$ – telling us that Sophroniskos is the father of Socrates.

A set of Horn clauses allows to deduce *facts* which can be proven by the implications. E.g., the first clause has no preconditions and no variables, and allows to conclude the fact that Socrates is a human. To be more precise, the predicate $human$ holds for the term $socrates$. Using this fact, it can be deduced

[1] Alfred Horn, 1918–2001, American logician and mathematician

from the second clause that Socrates is also mortal. For this, the variable X must be substituted by the term $socrates$. If X and Y are substituted with the term $sophroniskos$ and the term $socrates$, respectively, in the third clause, then both literals of the precondition become facts, i.e., are already proven. This substitution therefore allows to derive the new fact $human(sophroniskos)$. By that, we have deduced that, surprisingly, Sophroniskos is a human, too.

While we write a Horn clause in the form of an implication, it is in fact a finite disjunction of negated or non-negated literals (i.e., predicates applied to terms), containing at most one non-negated literal. We concentrate on *definite* Horn clauses, which have *exactly* one non-negated literal. This includes facts such as $human(socrates)$, given by the first clause of the example, but excludes clauses with an empty conclusion. The latter type of clause may be seen as a query. E.g., the query

$$\mathsf{false} \Leftarrow human(sophroniskos)$$

together with the clauses from the example allows for a resolution-based unsatisfiability proof, since it is possible to deduce $human(sophroniskos)$, which can further be resolved to false with this query. Resolution [RV01] is a technique that is not only used for automated deduction but also serves as the operational model of Prolog implementations.

In general, the predicates may take a (fixed) number of arguments, yet we only consider unary predicates in this work. This restriction is convenient and without loss of generality as the arguments of non-unary predicates may be equipped with a constructor on top, i.e., literals $p(t_1, \ldots, t_k)$ are rewritten as $p(c_k(t_1, \ldots, t_k))$ where c_k is a symbol of arity $k \geq 0$. The predicates of a (finite) set of clauses then characterize possibly infinite sets of variable-free terms with uninterpreted function symbols. Consider, e.g., the set $\mathcal{C}$ consisting of the two clauses:

$$\begin{array}{lcl} human(socrates) & \Leftarrow & \\ human(parent\_of(X)) & \Leftarrow & human(X) \end{array}$$

The two clauses are equivalent to the logical formula

$$human(socrates) \ \wedge \ (\ \forall X.\, human(X) \implies human(parent\_of(X))\ )$$

expressing that

- predicate $human$ holds for the term $socrates$, and
- if $human$ holds for a term t, then it also holds for the term $parent\_of(t)$.

Hence, in the *least model* of the set $\mathcal{C}$, $human$ holds for all terms

$$socrates,\ \ parent\_of(socrates),\ \ parent\_of(parent\_of(socrates)), \ldots$$

We also say that these terms are *recognized*, or *accepted* by the predicate $human$. A predicate p is *satisfiable* w.r.t. a set $\mathcal{C}$ of clauses if p accepts at least one term in the least model of $\mathcal{C}$.

There always is a *unique* least model for finite sets of definite Horn clauses [vEK76] – while formulas in first-order logic in general do not enjoy this property. But this does not automatically mean that we can also compute the least model. Horn clauses are Turing-powerful, i.e., there is no restriction on the computational power (roughly speaking, everything that is expressible in some model of computation is also expressible by a finite set of Horn clauses). Accordingly, the satisfiability problem is not decidable for finite sets of Horn clauses.

Researchers therefore have aimed at finding large, expressive subclasses of general Horn clauses for which satisfiability is decidable. In [Wei99], one such class has been identified for which satisfiability is decidable in exponential time. This class has been called $\mathcal{H}_1$ in [NNS02]. It differs from general Horn clauses with unary predicates in that the terms in the *heads* (i.e., the conclusions) may contain at most one constructor and no variable may occur twice in a head. (Thus, non-unary predicates in heads of $\mathcal{H}_1$-clauses do not contain any explicit constructor at all.) Finite conjunctions of $\mathcal{H}_1$-clauses can effectively be transformed to equivalent finite *tree automata* [NNS02] – for which efficient standard algorithms are available for the decision problems such as emptiness, membership, or universality. The technique for this transformation, which we call *normalization*, uses (ordered) resolution steps with *splitting* [GL05]. The resulting tree automaton is given by a set of *automata clauses*, a subclass of $\mathcal{H}_1$. When considering a tree automaton as a set of automata clauses, the following correspondencies exist:

Horn clause	$\leftrightarrow$	transition rule
predicate	$\leftrightarrow$	state
constructor / constant (also called *atom*)	$\leftrightarrow$	symbol
variable-free term	$\leftrightarrow$	tree
satisfiability of p	$\leftrightarrow$	non-emptiness of p

Satisfiability of a unary predicate p in the original set of $\mathcal{H}_1$-clauses then corresponds to non-emptiness of (the language accepted by) the corresponding state p in the equivalent tree automaton. Non-unary predicates must be equipped with a constructor on top of their arguments prior to the transformation, in order to allow for a tree representation.

What makes the class $\mathcal{H}_1$ even more attractive, is that there is a natural scheme how arbitrary Horn clauses can be abstracted by $\mathcal{H}_1$-clauses. In so doing, any finite conjunction of Horn clauses can be approximated by a finite set of $\mathcal{H}_1$-clauses, and the decision procedures for $\mathcal{H}_1$ can be used for automatic program analysis based on abstract interpretation – given that the analysis problem can conveniently be described by means of Horn clauses. This approach has been successfully applied

to the automatic analysis of secrecy in cryptographic protocols [Bla01] as well as the implementation of these in real-world programming languages [GLP05].

Finite bottom-up tree automata accept the *regular* tree languages, which are closed under regular operations such as union, intersection, complementation, projection, etc. For the case of finite trees, regular languages correspond to those languages that are expressible by *monadic second-order* (MSO) logic [TW68]. But tree automata do not only allow to conveniently represent possibly infinite regular sets of trees. They also allow for efficient algorithms for the decision of basic problems — in particular, *emptiness* and *membership*, but also universality, finiteness, inclusion, etc. For this reason, tree automata have been widely used for the specification of tree languages. Also, program analyses often *result* in tree automata.

As tree automata are restricted to inspect their input only locally during a bottom-up traversal, many efforts have been made to enhance their expressiveness. One way to achieve this is to add constraints to the transitions. In [Mon81], Mongy introduces automata that allow transitions to be constrained by equalities of subtrees, specified by *paths*. According to the naming scheme of Section 2.1, these automata are tree automata (respectively automata clauses) with *path equalities*. Since the class of languages accepted by such automata is not closed under complementation, a generalization is proposed in [CDG$^+$07] by additionally allowing disequality constraints between subtrees. For both classes, however, emptiness is undecidable. In fact, Tommasi showed in 1992 that already for automata with equality tests between *cousins* (i.e., subterms at depth one), emptiness is undecidable [Tom92].

For this reason, Bogaert and Tison have considered automata where only equalities and disequalities between *direct* subterms, i.e., constraints on paths of length one, are allowed [BT92]. This class is known as the class of tree automata with *constraints between brothers*. The class has a decidable emptiness problem. In the Horn clause formalism, constraints between brothers are comparisons between *variables* occurring in the core clause. Even when only disequalities are used, such constraints increase the expressiveness.

Example 1 In general, the language $L = \{f(t_1, t_2) \mid t_1 = t_2\}$ is not regular. Accordingly the language $\overline{L} = \{f(t_1, t_2) \mid t_1 \neq t_2\}$ cannot be regular either. The latter language, however, can easily be expressed by automata clauses with disequalities between variables:

$$p(f(X_1, X_2)) \Leftarrow \top(X_1), \top(X_2),\ X_1 \neq X_2$$

where the predicate $\top$ is defined such that it accepts any term constructible with the set of available symbols for the language $\overline{L}$. Consequently, automata clauses

with disequalities between variables are strictly more expressive than ordinary tree automata. □

In 2010, we introduced tree automata with *term* equality and disequality constraints [RS10]. We call these automata TCA (term-constrained automata). TCA are strictly more expressive than automata with constraints between brothers. While constraints between brothers correspond to equalities and disequalities between variables in the Horn clause formalism, TCA allow comparisons between arbitrary terms over the variables that occur in the clause. As with brother constraints, the *constraint language* of TCA enjoys closure under Boolean operations (when representing disjunctions as alternative clauses that only differ in their constraints). Therefore, also the languages accepted by TCA are still closed under Boolean operations. Moreover, as the main result for this extension, we were able to prove that emptiness is decidable for TCA. The latter does not only contrast with the undecidability of emptiness for the automata with path equalities of [Mon81]. It is also a surprising result as equality constraints for tree automata in general were assumed to be a source of undecidability — with the notable exception of the brother constraints.

In 1994, Caron et al. observed that emptiness is also decidable for automata with arbitrary path constraints if the number of equality tests along each path in the accepted trees is bounded [CCC+94]. These results have been applied to derive a decision procedure for inductive reducibility of rewrite systems [CJ94] and, more generally, for the first-order theory of encompassment [CCD93]. In fact, automata with *path disequalities* only are sufficient for deciding inductive reducibility [CJ97]. Automata with equality constraints based on a given tree homomorphism and path disequality constraints have recently been applied to solve the HOM problem for regular tree languages [GGRÀ10]. Extensions of constraints between brothers in ranked trees to constraints between siblings in *unranked* trees are studied by Löding and Wong [LW09]. *Global* equality and disequality constraints are considered by Godoy et al. in the context of MSO logic with isomorphism tests and unification with membership constraints [FTT08, BCG+10] – although it is not clear how this concept is related to the independent *local* constraints for transitions which we consider here.

While finite sets of Horn clauses are Turing-powerful, the clauses themselves also have certain restrictions. In particular, Horn clauses have difficulties to express *negative information* such as that two values must be different. In order to compensate for this deficiency, our goal is to extend Horn clauses with explicit constraints that express disequality conditions. As an example, consider *freshness* of keys or nonces, a property which is used in cryptographic protocols. A key is fresh if it does not occur in the list of already used keys. That a key *does* occur could be detected by *pattern matching* based on multiple occurrences of the

same variable within a precondition. But it is unclear how to directly specify the negation of such an equality condition by means of a Horn clause – as in the implication expressed by such a clause, only positive literals occur. That is, based on the equalities which can be enforced by multiple occurrences of the same variable within a clause, more equality conditions might be directly inferred – but not their negations.

In our extension of $\mathcal{H}_1$-clauses, such a condition can be added to the precondition of a clause. We also consider extensions with equality conditions (such as $X = Y$). Of particular interest, however, are *explicit disequalities*. While equalities such as

$$X = t \qquad (X \text{ a variable, } t \text{ a term})$$

could in principle be "inlined" in the clause by replacing each occurrence of X by the term t, matters are different with disequality conditions. Also it turns out that disequalities have much nicer properties than equalities w.r.t. decidability of the corresponding extension. $\mathcal{H}_1$-clauses with simple equalities between variables, e.g., already yield an undecidable class.

Example 2 When disequalities can be added to the clauses, it becomes possible to express and propagate negative conditions. The set of $\mathcal{H}_1$-clauses *extended with disequalities*

$$\begin{array}{lcl} \mathit{fresh\_key}(X) & \Leftarrow & q(X, [\,]) \\ q(X, Z) & \Leftarrow & q(X, :: (Y, Z)),\ X \neq Y \\ q(X, Y) & \Leftarrow & \mathit{key}(X), \mathit{old\_keys}(Y) \end{array}$$

can *conveniently* define a predicate $\mathit{fresh\_key}$ which expresses that a key is not contained in the list $\mathit{old\_keys}$. (Here, the upper case letters represent variables, $[\,]$ denotes an empty list, and :: is the list constructor.) The second clause ensures by means of the disequality $X \neq Y$ that, within the least model, $q(t, [\,])$ only holds in the first clause if $q(t, l)$ holds in the second clause for a list l in which t is not contained. □

Example 3 The extra expressiveness due to disequalities also allows to conveniently express that a predicate p contains all elements of another predicate q up to a finite language L:

$$p(X) \Leftarrow q(X), \phi_L$$

where $\phi_L \equiv \bigwedge\{X \neq t, \mid t \in L\}$. Likewise, disequalities allow to express that an implication $p(t_0) \Leftarrow \alpha$ only holds if a given predicate q contains at least r elements. For that, we define:

$$p(t_0) \Leftarrow \alpha, q(Z_1), \ldots, q(Z_r), \phi_r$$

where $\phi_r \equiv \bigwedge\{Z_i \neq Z_j \mid 1 \leq i < j \leq r\}$. □

We were able to show in [SR11] that the normalization procedure for the transformation of finite sets of $\mathcal{H}_1$-clauses to automaton form can be adapted so as to work also in presence of disequalities of *terms*. It essentially consists of a saturation procedure which adds, to the original set $\mathcal{C}$ of clauses, simpler clauses that are implied by $\mathcal{C}$ according to three rules – until all clauses are implied by the subset of *normal clauses*. The rules are *splitting*, *resolution*, and *propagation*. For $\mathcal{H}_1$-clauses without constraints, it is not hard to prove that this procedure terminates if only clauses are added which are not implied yet, since only finitely many literals can possibly occur. In presence of disequalities, an extra subtle argument, though, is required in order to prove that only finitely many conjunctions of disequalities need to be taken into account. For that, we apply a compactness property of conjunctions of finite disjunctions of *term equalities* [MORS05]. The procedure for the splitting rule is based on an efficient method which decides, for a given set of automata clauses, a predicate p, and a number $k \geq 1$, whether p accepts at least k terms or not, and if not, enumerates all accepted terms.

Further extensions. Sometimes, the concept of disequality of terms may not be adequate for expressing more involved properties than freshness of keys or nonces, or the like. Consider, for instance, a voting protocol where for a submitted vote not freshness of the overall expression must be checked but that the voter has not submitted any vote so far [FOO93, BRS07]. Such checks can be realized by means of disequalities on subterms, specified by paths.

Example 4 Assume that we are given a list l of votes where each element of l is of the form $vote(p, v)$ for a constructor symbol $vote$, a person p who has voted, and his or her vote v. The next vote $vote(p', v')$ then should not only be not contained in the list l but should differ from all elements of l *in the first argument*. Reusing the scheme of Example 2, this can be expressed by the predicate $valid$ as defined by:

$$\begin{array}{lcl} valid(X) & \Leftarrow & q(X, [\,]) \\ q(X, Z) & \Leftarrow & q(X, :: (Y, Z)),\ X.1 \neq Y.1 \\ q(X, Y) & \Leftarrow & is\_vote(X), votes(Y) \end{array}$$

The second clause of this definition involves a comparison between the person trying to vote, identified by $X.1$, and the leftmost (direct) subterm of the first list entry, identified by $Y.1$, which denotes one of the persons who have already voted. Here, the second clause is equivalent to the following clause with a term disequality.

$$q(vote(X_1, X_2), Z) \Leftarrow q(vote(X_1, X_2), :: (vote(Y_1, Y_2), Z)),\ X_1 \neq Y_1$$

This clause, however, is not an $\mathcal{H}_1$-clause because the head is non-flat, i.e., contains more than one constructor: besides the $vote$ constructor, we must also count

an implicit constructor for the non-unary predicate q. (Otherwise, $\mathcal{H}_1$ would be undecidable.) □

Since it is known that automata clauses with path disequalities are decidable, we asked ourselves whether it would even be possible to extend $\mathcal{H}_1$-clauses with disequalities of paths. For tree automata, path disequalities are strictly more expressive than term disequalities (we prove this in Section 3.2.1). Consequently, if the normalization technique still worked, the extension would also strictly generalize the class of $\mathcal{H}_1$ with term disequalities. Quite surprisingly, we succeeded in proving $\mathcal{H}_1$ with path disequalities decidable in [SR12]. We had to apply a more intricate method for splitting in presence of paths since pigeon-hole principle based arguments, which we used for splitting in presence of term disequalities do not work any more for disequalities of paths. In order to make splitting work for path disequalities, we invented a novel algorithm that computes, for a given predicate p, an existentially quantified variable Y, and a conjunction ψ of path disequalities containing Y, a finite set T of ground terms covering all "useful effects" of substituting Y within ψ by an *arbitrary* term that is accepted by p. I.e., T covers all cases which satisfy ψ, although there are possibly infinitely many substitutions for Y. Also the termination proof for this extension was difficult. The problem is that resolution steps result in *growing paths*. Interestingly, a further extension of the constraint language to disequalities of *terms containing path expressions*, achieved the desired result. Instead of growing paths, this extension only leads to growing (general) terms – for which we could then reapply the aforementioned compactness property for conjunctions of finite disjunctions of (ordinary) term equalities.

Disequality on terms or subterms, however, may be too imprecise in presence of semantic interpretations because syntactically distinct terms may represent the same value. One of the simplest forms of such interpretations are homomorphisms. Homomorphisms allow, e.g., to relabel nodes, to select specific subtrees (depending on labels), or permute subtrees. In the term representation of a tuple of trees, a suitable tree homomorphism thus allows to select individual components. Perhaps most useful in the context of cryptographic protocols is the possibility of disequalities modulo homomorphisms to compare messages while disregarding irrelevant information such as random padding or session keys. Analyses of anonymity violation or non-interference [GM82, BR05, Cha07] may search for values which are independent of sender identities or secret subparts, respectively.

Example 5 Consider the following example where the predicates p_u, p_v model the set of states reaching stages u, v of a protocol. For simplicity, assume that the value at stage u is obtained from the value at stage v by combining the value at

v with a secret value under a data constructor f, where the secret value is taken from some set $input$. This can be formalized by the following clauses:

$$\begin{array}{lcl} p_u(f(Z,Y)) & \Leftarrow & high(Z), p_v(Y) \\ high(secret(X)) & \Leftarrow & input(X) \end{array}$$

where the value Y at stage v may contain secrets as well. Now assume that we want to verify that the *public view* of values at stage u is independent of the secrets included into the values. Here, the public view of a value is realized by a homomorphism H which maps the constructor $secret$ (along with its respective subtrees) to some constant $\perp$ and is the identity for the remaining constructors. Then a potential violation of the independence could be expressed by the clause

$$error \;\Leftarrow\; p_u(X), p_u(Y),\; X \neq_{\mathrm{H}} Y$$

where $\neq_{\mathrm{H}}$ applies disequality to the images of its arguments under H. □

We proved in [RS12] that normalization still works if homomorphic images of terms are compared instead of just terms. We call such disequalities between homomorphic images of terms *hom-disequalities*. In order to enable splitting in presence of hom-disequalities, we introduced *hom-disequality-automata* (HDA), i.e., finite sets of automata clauses with complex heads, auxiliary variables, and term disequalities, and proved the following two propositions. First, for every finite set of automata clauses with hom-disequalities, an HDA can be effectively computed whose predicates accept the homomorphic images of the terms that these predicates accept in the original automaton. Second, k-finiteness is decidable for HDA. (These proofs can be found in Section 3.3.1). We also had to adapt the termination proof appropriately, in order to account for the effects of homomorphic transformations (see Section 3.3.3). As is the case with path disequalities, hom-disequalities strictly generalize term disequalities. Both classes, however, are incomparable to each other, i.e., hom-disequalities are a different proper extension of term disequalities for $\mathcal{H}_1$.

Summary of main contributions. We extend $\mathcal{H}_1$-clauses with various kinds of disequality constraints and show that the satisfiability problem for these extensions is still decidable. In particular, $\mathcal{H}_1$-clauses are proved decidable when extended with disequalities of terms, disequalities of subterms (known as *path* disequalities), or *hom-disequalities*, i.e., disequalities of images of terms under a given tree homomorphism. The decidability proofs are provided through constructions which, for every finite set $\mathcal{C}$ of $\mathcal{H}_1$-clauses with disequalities, compute a finite set $\mathcal{A}$ of *automata clauses* with the same sort of disequalities which is equivalent to $\mathcal{C}$. The general construction relies on a normalization procedure

in the spirit of [NNS02, GL05], although this procedure has to be adapted in a different way for each kind of disequality.

The resulting set $\mathcal{A}$ of automata clauses can be considered as a particular form of bottom-up tree automaton with disequality constraints. Thus, (finite sets of) $\mathcal{H}_1$-clauses with disequalities are proved equally expressive as tree automata with disequalities. Since emptiness is decidable for these classes of extended automata, the reduction proves that $\mathcal{H}_1$-clauses extended with disequalities are decidable. W.r.t. the languages accepted by tree automata extended with disequalities, we show that disequalities of subterms and hom-disequalities are incomparable to each other, while both classes are more expressive than tree automata or $\mathcal{H}_1$-clauses with disequalities of terms. Tree automata with term disequalities, on the other hand, are proved strictly more expressive than finite sets of $\mathcal{H}_1$-clauses without constraints.

Further contributions and results on constrained automata are presented in Section 2.2, most notably an extension with equality (and disequality) term constraints for finite bottom-up tree automata, for which we prove decidability of the emptiness problem, and closure under Boolean operations. This extension strictly generalizes the known class of tree automata with constraints between brothers of Bogaert and Tison [BT92]. We also present *undecidability* proofs for other classes of tree automata extended with equality conditions (see Sections 2.3 and 2.4). These results as well as the undecidability of emptiness for automata with arbitrary equality constraints between subterms as shown in [Mon81] contrast with decidability of emptiness for our aforementioned new class of automata with equality and disequality term constraints.

The used proof techniques for our (un)decidability proofs are in part standard, but we also introduce novel techniques such as, e.g., a quantifier elimination algorithm (see the proof of Theorem 18 in Section 3.2.4).

Outline. In Chapter 2, we formally introduce Horn clauses and constraints. Section 2.1 reviews basic concepts and notations such as terms, *paths*, which identify subterms, and constraints. Also tree homomorphisms are introduced, which are used for the definition of *hom-constraints*, i.e., comparisons of homomorphic images of terms. Based on these concepts, Section 2.1 furthermore introduces the classes of constrained Horn clauses which we consider in this work, in particular the class $\mathcal{H}_1$ as well as useful subclasses of $\mathcal{H}_1$ such as simple, normal and automata clauses. We also establish two basic results: finite sets of normal clauses extended with term, path, or hom-constraints can be transformed to finite sets of automata clauses with the same sort of constraints, and the membership problem for finite sets of such clauses is decidable, i.e., for each variable-free term t and predicate p, it can be decided whether or not $p(t)$ holds in the least model

of the given set of clauses. Section 2.1 also provides an overview over known and new classes of Horn clauses (resp. tree automata) with constraints, with links to the respective sections where the classes are considered, and in case of a new class also the publication where we introduced the corresponding class. Finally, Boolean combinations of single constraints are discussed, and normal forms for term constraints are introduced.

In Section 2.2, tree automata with equalities and disequalities of terms (TCA) are introduced. The decidability proof for TCA is based on a saturation procedure that reduces TCA to *generalized* tree automata with disequalities only. This class is considered in Section 2.2.1. We relate the expressiveness of TCA to the known classes of tree automata with constraints between brothers and tree automata with path constraints in Section 2.2.2. We furthermore show that TCA can be determinized and that they are closed under Boolean operations. Finally, emptiness of TCA is proved decidable. Section 2.3 considers further extensions for automata with equalities and proves emptiness for these extensions undecidable. In particular, complex preconditions (i.e., literals $p(t)$ where t is not a variable but a term) and auxiliary variables not occurring in the head of a clause yield undecidability when combined with term equalities. Section 2.4 shows that term equalities alone already make the corresponding class of constrained automata undecidable if the equalities are interpreted modulo a given tree homomorphism, i.e., if images of terms under a given tree homomorphism are compared.

Chapter 3 is devoted to procedures for the normalization of finite sets $\mathcal{C}$ of Horn clauses from the class $\mathcal{H}_1$, effectively producing an equivalent tree automaton $\mathcal{A}$ (in the sense that non-emptiness of a predicate p within $\mathcal{A}$ corresponds to satisfiability of predicate p within $\mathcal{C}$). The general framework of $\mathcal{H}_1$-normalization, instantiated with term disequalities, is presented in Section 3.1.

Section 3.2 extends the framework to $\mathcal{H}_1$-clauses with path disequalities. We show in Section 3.2.1 that $\mathcal{H}_1$-clauses with path disequalities are strictly more expressive than $\mathcal{H}_1$-clauses with term disequalities. In Section 3.2.2, we illustrate the challenges that arise in the case of $\mathcal{H}_1$ with path disequalities w.r.t. splitting, the technique for the removal of auxiliary variables from preconditions. Section 3.2.3 then introduces *general terms*, an extension both of terms and paths, which enables to argue that the resolution steps also terminate in presence of paths. Section 3.2.3 also provides evidence that for the extension of tree automata, general term disequalities are equally expressive as disequalities between labeled or unlabeled paths. In Section 3.2.4, we present a method that enables splitting even in presence of paths.

A method for the normalization of $\mathcal{H}_1$-clauses with hom-disequalities is provided in Section 3.3. In order to enable splitting in presence of hom-disequalities, Section 3.3.1 introduces *hom-disequality-automata* (HDA), whose predicates accept homomorphic images of terms, and shows k-finiteness decidable for HDA.

Section 3.3.2 then compares the expressiveness of finite sets of automata clauses (or $\mathcal{H}_1$-clauses) extended with term, path, or hom-disequalities, and Section 3.3.3 constructs the normalization procedure for $\mathcal{H}_1$ with hom-disequalities based on the results of Section 3.3.1. Chapter 4 finally provides an outlook.

Chapter 2

Horn Clauses and Constraints

2.1 Basics

This section formally introduces Horn clauses — in particular, the subclass $\mathcal{H}_1$ as well as another subclass which corresponds to finite bottom-up tree automata. For these, extensions are defined, too, resulting from the addition of *constraints*. Furthermore, *paths* and tree homomorphisms are introduced.

A (definite) first-order Horn clause, written usually as a (reversed) implication, has the form:

$$A \Leftarrow B_1, \ldots, B_m \qquad m \geq 0$$

where the conclusion A as well as the preconditions $B_1, \ldots, B_m$ are *literals*, i.e., expressions $p(t_1, \ldots, t_k)$ consisting of a predicate p and terms $t_1, \ldots, t_k$, which may contain variables. Here, we only consider unary predicates, i.e., $k = 1$, to which the general case can be reduced. The variables are implicitly quantified by a universal quantifier. The terms t_i are made up of symbols (constructors with subterms, and constants) and variables. The precondition of such a clause can be extended by a constraint ϕ which allows to express (dis)equality conditions on terms or subterms, or the like. Then the clause has the form:

$$A \Leftarrow B_1, \ldots, B_m, \phi$$

where ϕ is a conjunction of expressions $l \circ r$ for terms or subterms l, r, and a given (set of) operator(s) $\circ$. The operator could for instance be a disequality.

A class $\mathcal{H}$ of (finite sets of) Horn clauses is said to be *decidable* if for every set $\mathcal{A} \in \mathcal{H}$, satisfiability (w.r.t. the least model) can be decided for every query $p(t)$, where p denotes a predicate, and t an arbitrary term. The decision procedures for all decidable classes considered in this work are known.

Terms, paths, and constraints. First-order Horn clauses with unary predicates define sets of trees. The trees are made up of symbols from a ranked alphabet (Σ, ar) where Σ denotes a finite set of *constructors* and $ar : \Sigma \to \mathbb{N} \cup \{0\}$ specifies each constructor's arity. If the arities of symbols are understood, then the ranked alphabet is denoted by Σ alone. Nullary constructors are also called atoms. In order to avoid trivial cases, let us assume that there is at least one atom and one constructor of arity greater than 0. For a ranked alphabet Σ and a set $\mathbf{V} = \{X_1, X_2, \ldots\}$ of *variables*, the set $\mathcal{T}_\Sigma(\mathbf{V})$ of (finite ordered ranked) trees over Σ and $\mathbf{V}$ consists of all terms t given by the grammar:

$$t \quad ::= \quad X_i \mid a \mid b(t_1, \ldots, t_k)$$

where $a, b \in \Sigma$, and a has arity 0, while b has arity $k > 0$. We also use the word *term* for tree. For an expression e, the expression $\mathrm{vars}(e)$ denotes the set of variables occurring in e. A tree t is called *ground* if $\mathrm{vars}(t) = \emptyset$, i.e., t contains no variable $X \in \mathbf{V}$. The set of ground terms (or ground trees) is denoted $\mathcal{T}_\Sigma$. Note that $\mathcal{T}_\Sigma$ is infinite according to our assumption on Σ. A (positive) *literal* A is an expression of the form $p(t)$ where p is a unary predicate and $t \in \mathcal{T}_\Sigma(\mathbf{V})$. Non-unary predicates can be integrated in our framework, too, by equipping their arguments with an *implicit* constructor of the same arity as the predicate.

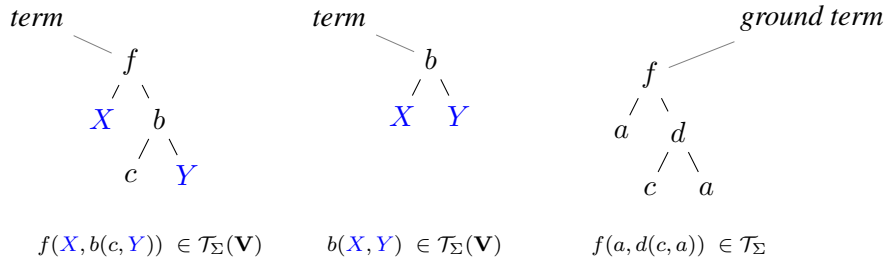

Figure 2.1: Two terms containing variables, and a variable-free term or tree

Subterms are identified by *paths*. A labeled path π is given by a finite sequence $(f_1, i_1).(f_2, i_2).\ldots.(f_n, i_n)$ where for every j, $f_j \in \Sigma$ and $1 \leq i_j \leq m_j$ if m_j is the arity of f_j. The empty sequence is denoted ϵ. An expression $X.\pi$ for a variable X and a path π is called *path expression*. We also write X instead of $X.\epsilon$.

A *substitution* θ is a mapping $\theta : \mathbf{V} \to \mathcal{T}_\Sigma(\mathbf{V})$. The mapping may be written as $\theta = [t_{i_1}/X_{i_1}, \ldots, t_{i_j}/X_{i_j}]$ or as $\theta = \{X_{i_1} \mapsto t_{i_1}, \ldots, X_{i_j} \mapsto t_{i_j}\}$. Then $X\theta = X$ for all variables X that are not mentioned. We write θe and $e\theta$ for the result of applying θ to the expression e. θ is called *ground* w.r.t. an expression e if θX_i is ground for all variables $X_i \in \mathrm{vars}(e)$.

Constraints extend the preconditions of clauses by additional conditions. Thus, the constraint ϕ of a clause is a *conjunction* of atomic constraints, for a given finite

set B of binary Boolean operators:

$$\phi = \bigwedge_{i=1,\ldots,m} l_i \circ_i r_i \qquad (m \geq 0, \circ_i \in B)$$

where the expressions l_i, r_i are compatible with the operator $\circ_i$. In most of the cases that we consider, B contains a disequality operator and possibly also the corresponding equality operator, i.e., either all constraints ϕ are of the form:

$$\bigwedge_{i=1,\ldots,m} l_i \neq r_i \qquad (m \geq 0)$$

or they all can be written as:

$$(\bigwedge_{i=1,\ldots,m} l_i = r_i) \wedge (\bigwedge_{i=m+1,\ldots,m+n} l_i \neq r_i) \qquad (m, n \geq 0)$$

A *term disequality constraint*, for instance, is a conjunction of disequalities $s \neq t$ for terms s, t. In this case, the type of the l_i and r_i is that of an ordinary term, and the operator $\circ$ is the disequality of terms. A path disequality constraint, on the other hand, is a conjunction of disequalities $X_i.\pi \neq X_j.\pi'$ of path expressions $X_i.\pi, X_j.\pi'$.

The substitution θ *satisfies* the constraint $l \circ r$, denoted by $\theta \models (l \circ r)$, if and only if $\theta l \circ \theta r$. An exception to this scheme is needed for path expressions $X.\pi$, since $\theta(l)$ and $\theta(r)$ may be undefined. Should either $\theta(l)$ or $\theta(r)$ be undefined, then $\theta \models l \circ r$ if and only if $\circ$ expresses disequality. (The intuition behind is that equality should be more forcing.) Consequently, an atomic equality is the logical negation of the corresponding disequality. Finally, satisfiability is extended to monotone Boolean combinations in the usual way:

$$\begin{array}{ll} \theta \models (\phi_1 \wedge \phi_2) & \text{iff} \quad (\theta \models \phi_1) \wedge (\theta \models \phi_2) \\ \theta \models (\phi_1 \vee \phi_2) & \text{iff} \quad (\theta \models \phi_1) \vee (\theta \models \phi_2) \end{array}$$

Tree homomorphisms. A tree homomorphism H maps each symbol $f \in \Sigma$ of arity $k \geq 0$ to a term $t(X_1, \ldots, X_k)$, i.e., a term containing only variables from the set $\{X_1, \ldots, X_k\}$. The mapping $\mathrm{H}^* : \mathcal{T}_\Sigma(\mathbf{V}) \mapsto \mathcal{T}_\Sigma(\mathbf{V})$ is then recursively defined as $\mathrm{H}^*(X) = X$ for $X \in \mathbf{V}$, and $\mathrm{H}^*(f(t_1, \ldots, t_k)) = \mathrm{H}(f)(\mathrm{H}^*(t_1), \ldots, \mathrm{H}^*(t_k)) = t[\mathrm{H}^*(t_1)/X_1, \ \ldots, \ \mathrm{H}^*(t_k)/X_k]$. Figure 2.2 shows an example of applying H^*. $\mathrm{H}^*(s)$ is ground whenever s is ground. The function H^{-1} reverses the effect of applying H^*, i.e., $\mathrm{H}^{-1}s = \{t \mid \mathrm{H}^*t = s\}$. We extend H^* and H^{-1} for sets of terms T by defining $\mathrm{H}^*T = \{\mathrm{H}^*t \mid t \in T\}$ and $\mathrm{H}^{-1}T = \{t \mid \mathrm{H}^*t \in T\}$. For notational convenience, we may also write $\mathrm{H}e$ or $\mathrm{H}(e)$ instead of H^*e, meaning that the mapping H^* is applied to the expression e.

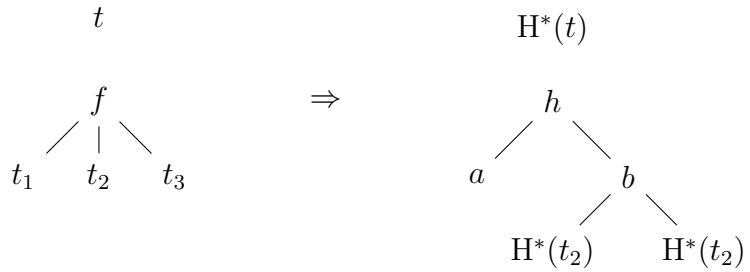

Figure 2.2: Transformation of the term $t = f(t_1, t_2, t_3)$ with $t_i \in \mathcal{T}_\Sigma(\mathbf{V})$ by the tree homomorphism H with $\mathrm{H}(f) = h(a, b(X_2, X_2))$

A *hom-disequality* is an expression $s \neq_\mathrm{H} t$ where $s, t \in \mathcal{T}_\Sigma(\mathbf{V})$. In case that both s and t are ground terms, $s \neq_\mathrm{H} t$ is equivalent to $\mathrm{H}^* s \neq \mathrm{H}^* t$. A ground substitution θ satisfies a hom-disequality $s \neq_\mathrm{H} t$ if (and only if) $\theta s \neq_\mathrm{H} \theta t$, i.e., $\mathrm{H}^*(\theta s) \neq \mathrm{H}^*(\theta t)$. For a substitution θ (not necessarily ground) and a homomorphism H, let θ_H denote the substitution given by: $\theta_\mathrm{H} X = \mathrm{H}^*(\theta X)$. Then it holds that $\mathrm{H}^* \circ \theta = \theta_\mathrm{H} \circ \mathrm{H}^*$, i.e., for every term t, we have: $\mathrm{H}^*(\theta t) = \theta_\mathrm{H}(\mathrm{H}^* t)$. As with path (dis)equalities, the *hom-equality* $s =_\mathrm{H} t$ is defined as the logical negation of the corresponding disequality $s \neq_\mathrm{H} t$, i.e., we have $s =_\mathrm{H} t \Leftrightarrow \neg(s \neq_\mathrm{H} t)$.
Tree homomorphisms are rather expressive. They may, e.g.,

- rename constructors: $\mathrm{H}^*(g(X_1)) = h(X_1)$
- delete constructors: $\mathrm{H}^*(g(X_1)) = X_1$
- delete subtrees: $\mathrm{H}^*(f(X_1, X_2)) = g(X_2)$
- add constructors: $\mathrm{H}^*(f(X_1, X_2)) = f(g(X_1), h(g(X_2)))$
- copy subtrees: $\mathrm{H}^*(g(X_1)) = f(X_1, X_1)$
- permute subtrees: $\mathrm{H}^*(f(X_1, X_2)) = f(X_2, X_1)$
- combine two or more of these features.

Horn clauses with constraints. A *constrained Horn clause* c is given by the implication

$$B_0 \Leftarrow B_1, \ldots, B_m, \phi$$

where $B_0, \ldots, B_m$ are literals and ϕ is a constraint. The left-hand side B_0 is the *head* of the clause c while the sequence $B_1, \ldots, B_m, \phi$ denotes the *body* or *precondition* of c. The constraint ϕ imposes an additional restriction on the applicability of the clause. A constraint ϕ which is always true can be omitted. With the constraint omitted, $B_0 \Leftarrow B_1, \ldots, B_m$ is called the *core clause* of c. The atomic constraints of ϕ may be separated by commas.

Let $\mathcal{C}$ denote a set of Horn clauses. Then the *least model* $\mathcal{M}_\mathcal{C}$ of $\mathcal{C}$ is the least set M such that $p(t\theta) \in M$ for a ground substitution θ whenever there is a clause

$p(t) \Leftarrow p_1(t_1), \ldots, p_k(t_k), \phi$ in $\mathcal{C}$ such that $p_i(t_i\theta) \in M$ for all $i = 1, \ldots, k$, and $\theta \models \phi$. The language $\{t \in \mathcal{T}_\Sigma \mid p(t) \in \mathcal{M}_\mathcal{C}\}$ of p is also denoted $[\![p]\!]_\mathcal{C}$. For convenience, we also consider the set $[\![p]\!]^i_\mathcal{C} = \{t \in \mathcal{T}_\Sigma \mid p(t) \in \mathcal{T}^i_\mathcal{C}(\emptyset)\}$ where the operator $\mathcal{T}_\mathcal{C}$ is defined as follows. Assume that M is any set of ground facts $p(t)$. Then $\mathcal{T}_\mathcal{C}(M)$ is the set of all ground facts θB_0 where θ is a ground substitution, $B_0 \Leftarrow B_1, \ldots, B_m, \phi$ is in $\mathcal{C}$, $\theta B_1, \ldots, \theta B_m \in M$, and $\theta \models \phi$. The set $[\![p]\!]^i_\mathcal{C}$ thus consists of all trees t where the fact $p(t)$ can be derived by at most i rounds of fixpoint iteration, and we have

$$\mathcal{M}_\mathcal{C} = \bigcup\{\mathcal{T}^i_\mathcal{C}(\emptyset) \mid i \geq 0\}$$

Consider as an example the set $\mathcal{A}$ consisting of two (non-constrained) Horn clauses:

$$p(f(X_1)) \Leftarrow p(X_1) \qquad \text{and} \qquad \mathrm{p(a)} \Leftarrow$$

Here, we have $[\![p]\!]^i_\mathcal{A} = \{f^j(a) \mid 0 \leq j \leq i-1\}$, the number of elements in $[\![p]\!]^i_\mathcal{A}$ is given by $|[\![p]\!]^i_\mathcal{A}| = i$, and the language *accepted* (or *recognized*) by p is given by

$$[\![p]\!]_\mathcal{A} = \bigcup\{[\![p]\!]^i_\mathcal{A} \mid i \geq 0\} = \{f^i(a) \mid i \geq 0\} = \{a, f(a), f(f(a)), ...\}$$

$\mathcal{H}_1$-clauses, normal clauses, automata clauses. Let us introduce the subclasses of Horn clauses which we consider here. A Horn clause is an $\mathcal{H}_1$-clause if the term t in the head $p(t)$ is *flat* and *linear*, i.e., t contains at most one constructor, and no variable occurs twice in t. For convenience, we adopt the convention that the variables in the heads of $\mathcal{H}_1$-clauses are enumerated $X_1, \ldots, X_k$, i.e., an $\mathcal{H}_1$-clause has one of these two forms:

$$p(X_1) \;\Leftarrow\; \alpha, \phi \qquad \textbf{or} \qquad p(f(X_1, \ldots, X_k)) \Leftarrow \alpha, \phi$$

for an arbitrary sequence of literals α and an arbitrary constraint ϕ — where the case of atoms:

$$p(a) \Leftarrow \alpha, \phi$$

is subsumed by choosing $k = 0$. Moreover for a distinction, variables that do *not* occur in the head will be denoted $Y, Y_1, \ldots$

An $\mathcal{H}_1$-clause is *simple* if no literal $q_j(t_j)$ in the body contains a constructor, i.e., each t_j is a variable. The Horn clause is a *normal clause* if it is of the form:

$$q(f(X_1, \ldots, X_k)) \Leftarrow p_1(X_{i_1}), \ldots, p_r(X_{i_r}), \phi \qquad \textbf{or} \qquad q(X_1) \Leftarrow \phi$$

where all variables occurring in the body of the clause also occur in the head. Note that in the notation of such clauses, the variable X_i is distinct from X_j for

$i \neq j$; predicates p_i, p_j however may coincide for $i \neq j$. Also the (universal) quantification of variables is implicit in each clause.

The clause is an *automata clause* if additionally each variable X_i occurring in the head occurs exactly once in the literals occurring in the body and the head contains exactly one constructor, i.e., the clause has the form:

$$q(f(X_1, \ldots, X_k)) \Leftarrow p_1(X_1), \ldots, p_k(X_k), \phi \ .$$

In particular, each normal clause as well as each automata clause is an $\mathcal{H}_1$-clause.

Example 6 Neither the clause

$$p(f(X, X)) \Leftarrow q(X)$$

nor the clause

$$p(f(X, g(Y))) \Leftarrow q(h(Y))$$

is $\mathcal{H}_1$: the first one is not linear while the second one has more than one constructor in the head. The clause:

$$p(f(X, Y)) \Leftarrow p(f(X, g(Y))), X \neq Y$$

on the other hand is an $\mathcal{H}_1$-clause with a disequality. Moreover, the following clauses are simple, normal, and automata clauses, respectively:

$$\begin{array}{lcll} p(X) & \Leftarrow & q_1(X), q_2(X), q(Y), & X \neq h(Y, a) \\ p(f(X, Y)) & \Leftarrow & q_1(X), q_2(X), q(Y), & X \neq h(Y, a) \\ p(f(X, Y)) & \Leftarrow & q_1(X), q_2(Y), & X \neq h(Y, a) \end{array}$$

□

Since Σ is fixed, it shall be assumed in the following that there always is a predicate $\top$ which is defined by a set of automata clauses so that it accepts every term. In particular, for $\Sigma = \{a_1, \ldots, a_n\}$ the following clauses are implicitly defined by this convention:

$$\top(a_i(X_1, \ldots, X_{k_i})) \Leftarrow \top(X_1), \ldots, \top(X_{k_i}) \qquad 1 \leq i \leq n$$

where $k_i = ar(a_i) \geq 0$ denotes the arity of the symbol a_i.

A small note on tree automata versus automata clauses: in the following, we do not differentiate between tree automata with constraints and finite sets of automata clauses. The predicates and clauses of a (finite) set of automata clauses correspond to the states and transition rules of the corresponding tree automaton with constraints and vice versa. For convenience, automata are usually defined together with a subset of distinguished *final* states. In this work, we consider all languages accepted by single predicates (and their properties such as emptiness) so that it is not necessary to define a distinguished set of final predicates.

Constraints — an overview. The described classes are generic in the kind of (atomic) constraints that are used. For the basic constraints, we distinguish

- equalities ($l = r$) and disequalities ($l \neq r$),

and the following three types of left-hand and right-hand sides that are compared:

- terms: *term (dis)equalities*,
- subterms: *path (dis)equalities*,
- homomorphic images of terms: *hom-(dis)equalities*.

The notations *term constraint*, *path constraint*, and *hom-constraint* denote either an equality or a disequality. For convenience, here is a table with all instantiations of (combinations of) constraints that we consider, together with the sections where they first occur. The framed entries indicate the novel classes introduced in the publications [RS10], [SR11], [SR12], [RS12].

Type of constraint	Example	Ext. TA	Ext. $\mathcal{H}_1$	D. TA	D. $\mathcal{H}_1$
Brother constraints	$X = Y$	4	4	Yes	Yes
Term disequalities	$X \neq f(a, Y)$	24	39	Yes	Yes
Term constraints	$X = g(Y)$	23	34	Yes	No
Path disequalities	$X.(f, 2) \neq g(a, b)$	51	51	Yes	Yes
Path constraints	$X.(f, 2) = Y.(g, 1)$	4	—	No	No
Hom-disequalities	$g(X) \neq_{\mathrm{H}} a$	66	65	Yes	Yes
Hom-constraints	$X =_{\mathrm{H}} f(Y, b)$	36	—	No	No

The column *Ext. TA* (*Ext.* $\mathcal{H}_1$) contains the page where the extension of tree automata ($\mathcal{H}_1$) with the corresponding type of constraint is introduced, while the column *D. TA* (*D.* $\mathcal{H}_1$) indicates whether emptiness (satisfiability) is decidable for the class of extended automata ($\mathcal{H}_1$-clauses). $\mathcal{H}_1$-clauses with path or hom-equalities are not directly considered since already the corresponding subsets of automata clauses with path or hom-equalities are undecidable classes. On the other hand, we also consider refinements of the tabled classes where needed (e.g., clauses in automata form but with complex heads and auxiliary variables are considered in Section 3.3.1).

Using dynamic programming, the following proposition is obtained:

Lemma 1 *Membership (of a ground term) is decidable for every finite set $\mathcal{A}$ of automata clauses extended with term, path, or hom-constraints.*

Proof. We observe that for a given ground substitution θ and a constraint ϕ, it is decidable whether $\theta \models \phi$. For a ground term t', let $\mathsf{valid}_{\mathcal{A}}(t')$ denote the set of all predicates p with $t' \in [\![p]\!]_{\mathcal{A}}$. The algorithm for testing membership of a term t determines for all subterms t' of t the set $\mathsf{valid}_{\mathcal{A}}(t')$. Then $t' \in [\![p]\!]_{\mathcal{A}}$ iff $p \in \mathsf{valid}_{\mathcal{A}}(t')$. The algorithm proceeds bottom-up over t. Assume that the algorithm has already determined the sets $\mathsf{valid}_{\mathcal{A}}(t')$ for all proper subterm t' of a term t_1. In order to determine $\mathsf{valid}_{\mathcal{A}}(t_1)$, the algorithm iterates over all clauses of $\mathcal{A}$. The set $\mathsf{valid}_{\mathcal{A}}(t_1)$ then consists of all predicates p for which there is a clause $p(s) \Leftarrow p_1(X_1), \ldots, p_k(X_k), \phi$ with $t_1 = s\theta$, $\theta = \{X_1 \mapsto s_1, \ldots, X_k \mapsto s_k\}$, such that for all $i = 1, \ldots, k$, $p_i \in \mathsf{valid}_{\mathcal{A}}(s_i)$ and $\theta \models \phi$. Thus, the set $\mathsf{valid}_{\mathcal{A}}(t')$ can be computed for all subterms t' of t, which completes the proof. □

Normal clauses versus automata clauses. Every set $\mathcal{N}$ of normal clauses without constraints can be transformed to an equivalent set $\mathcal{A}$ of automata clauses whose predicates correspond to conjunctions $p_1 \cap \ldots \cap p_j, j \geq 0$, of original predicates p_i from $\mathcal{N}$. This also holds for constrained Horn clauses.

Lemma 2 *For every finite set $\mathcal{N}$ of normal clauses, extended with term, path, or hom-constraints, a finite set $\mathcal{A}$ of automata clauses (with term, path, or hom-constraints) can be effectively constructed with $[\![p]\!]_{\mathcal{N}} = [\![p]\!]_{\mathcal{A}}$ for each predicate p of $\mathcal{N}$.*

Proof. The construction is a variant of the subset construction for finite tree automata. For every subset A of predicates occurring in $\mathcal{N}$, we introduce a new predicate $[A]$. Now consider a subset A of predicates of $\mathcal{N}$ and a family $p(f(X_1, \ldots, X_k)) \Leftarrow \alpha_p, \phi_p$ of clauses from $\mathcal{N}$ ($p \in A$). Let $B_{p,j}$ denote the set of all predicates q such that α_p contains the literal $q(X_j)$. Then the new set $\mathcal{A}$ has the clause:

$$[A](f(X_1, \ldots, X_k)) \Leftarrow [B_1](X_1), \ldots, [B_k](X_k), \psi$$

where $B_j = \bigcup_{p \in A} B_{p,j}$ and $\psi = \bigwedge_{p \in A} \phi_p$. And these are the only clauses of $\mathcal{A}$. Then for all subsets A,

$$[\![[A]]\!]_{\mathcal{A}} = \bigcap_{p \in A} [\![p]\!]_{\mathcal{N}}$$

Thus by renaming the predicates $[p]$ of $\mathcal{A}$ into p, we obtain the required result. □

Example 7 For terms t and constructors s, let $s^0(t) = t$, $s^{i+1}(t) = s(s^i(t))$. Consider the set $\mathcal{N}$ of normal clauses:

$$\begin{array}{lcl} adult(pers(X_1, X_2)) & \Leftarrow & age(X_1), name(X_2), old(X_1) \\ old(X_1) & \Leftarrow & \bigwedge_{i \leq 18} X_1 \neq s^i(0) \\ age(0) & \Leftarrow & \\ age(s(X_1)) & \Leftarrow & age(X_1) \end{array}$$

In order to construct a corresponding finite set of automata clauses, the variables X_1 occurring in the heads of normal clauses $p(X_1) \Leftarrow \phi$ are instantiated with all terms $c(X_1, \ldots, X_k)$, c a constructor of arity $k \geq 0$. Additionally, we introduce auxiliary predicates for all conjunctions of predicates, possibly occurring in preconditions. For the set $\mathcal{N}$ of normal clauses, we obtain the set $\mathcal{A}$ of clauses:

$$\begin{array}{lcl}
adult(pers(X_1, X_2)) & \Leftarrow & age_old(X_1), name(X_2) \\
old(s(X_1)) & \Leftarrow & \top(X_1), \bigwedge_{i \leq 17} X_1 \neq s^i(0) \\
age_old(s(X_1)) & \Leftarrow & age(X_1), \bigwedge_{i \leq 17} X_1 \neq s^i(0) \\
age(0) & \Leftarrow & \\
age(s(X_1)) & \Leftarrow & age(X_1) \\
\top(0) & \Leftarrow & \\
\top(s(X_1)) & \Leftarrow & \top(X_1) \\
\top(pers(X_1, X_2)) & \Leftarrow & \top(X_1), \top(X_2) \\
old(pers(X_1, X_2)) & \Leftarrow & \top(X_1), \top(X_2)
\end{array}$$

From the three possible new clauses defining the predicate old, we only kept the clauses for the constructors s and $pers$, since the precondition of the clause

$$old(0) \Leftarrow \bigwedge_{i \leq 18} 0 \neq s^i(0)$$

contains the disequality $0 \neq 0$ which is unsatisfiable. In order to illustrate the required conjunctions, we *explicitly* specify the new predicate $\top$ that denotes the empty conjunction of predicates, i.e., accepts all terms in $\mathcal{T}_\Sigma$ where $\Sigma = \{0, s, pers\}$ is given by the atoms and constructors occurring in $\mathcal{N}$. The new predicate age_old represents the conjunction of the predicates age and old. □

Boolean combinations of single constraints. As we allow the constraints ϕ to be finite conjunctions of atomic constraints, the *constraint language* is always closed under conjunction. In a certain sense, it is also closed under *disjunction*, because a constraint $\phi_1 \vee \phi_2$ of a clause

$$h \Leftarrow \alpha,\ \phi_1 \vee \phi_2$$

can be simulated by splitting the core clause $h \Leftarrow \alpha$, obtaining the two clauses:

$$h \Leftarrow \alpha,\ \phi_1 \qquad \text{and} \qquad h \Leftarrow \alpha,\ \phi_2$$

Thus, it is possible to realize any positive Boolean combination of atomic constraints by transforming it to DNF (disjunctive normal form) $\phi_1 \vee \ldots \vee \phi_l$ and defining clauses $h \Leftarrow \alpha, \phi_i$ for $1 \leq i \leq l$.

However, the constraint language is *not* automatically closed under negation. This is only the case if the set of Boolean operators itself is closed under negation, i.e., if for all $\circ \in B$ we have some $\circ' \in B$ such that $\neg(s \circ t)$ is logically equivalent to $s \circ' t$. As we define all considered equalities and disequalities in such a way that they are the negations of each other, the closure conditions can be simplified as follows. For a class of constraints of a certain kind (term, path, or hom-constraints), the constraint language *always* enjoys positive Boolean closure, and it enjoys full Boolean closure if both equalities and disequalities are allowed.

Normal forms of term constraints. If either only equalities or only disequalities are used, the constraint language is closed only under positive Boolean combinations (disjunction and conjunction). For term constraints, we observe that an equality $s = t$ of terms s, t is equivalent to a finite conjunction $X_{i_1} = t_1 \wedge \ldots \wedge X_{i_j} = t_j$ of equalities where all left-hand sides are variables — while a disequality $s \neq t$ is equivalent to a disjunction $X_{i_1} \neq s_1 \vee \ldots \vee X_{i_r} \neq s_r$ of disequalities.

Example 8 The following clause with one term equality and one term disequality:

$$h \;\Leftarrow\; p(X_1), q(X_2), r(X_3),\;\; g(X_1, a) = g(X_3, X_2), f(X_1, X_2) \neq f(X_2, X_3)$$

is equivalent to the clause:

$$\begin{array}{lll} h & \Leftarrow & p(X_1), q(X_2), r(X_3), \qquad (X_1 = X_3, X_2 = a, X_1 \neq X_2) \\ & & \qquad\qquad\quad \vee \;\; (X_1 = X_3, X_2 = a, X_2 \neq X_3) \end{array}$$

which in turn is equivalent to the two clauses:

$$\begin{array}{lll} h & \Leftarrow & p(X_1), q(X_2), r(X_3),\; X_1 = X_3, X_2 = a, X_1 \neq X_2 \\ h & \Leftarrow & p(X_1), q(X_2), r(X_3),\; X_1 = X_3, X_2 = a, X_2 \neq X_3 \end{array}$$

□

A term constraint ϕ is in *normal form* if it is either false or a conjunction of atomic constraints where all left-hand sides of equality constraints are variables X_i, and the following holds:

(1) If $X_i = X_j$ occurs in ϕ, then $i < j$; and

(2) If an equality constraint $X_i = t$ occurs in ϕ, then X_i does not occur in any other atomic constraint of ϕ.

The following fact is well-known.

Lemma 3 *For every term constraint ϕ, a disjunction $\phi' \equiv \phi_1 \vee \ldots \vee \phi_r$ of term constraints ϕ_i can be constructed such that each ϕ_i is in normal form and ϕ is equivalent to ϕ', i.e., for all ground substitutions θ,*

$$\theta \models \phi \quad \text{iff} \quad \theta \models \phi'$$

In order to construct ϕ', ϕ is first transformed to an equivalent positive Boolean formula over atomic constraints where all left-hand sides are variables. Then this formula is transformed to DNF $\phi'_1 \vee \ldots \vee \phi'_r$. After that, we further proceed with each ϕ'_i separately. Let $\phi'_i \equiv \psi_1 \wedge \psi_2$ where ψ_1 and ψ_2 consist of all equality constraints and all disequality constraints of ϕ'_i, respectively. If ψ_1 is unsatisfiable, we return false for ϕ_i, i.e., remove it from the disjunction. Otherwise, a most general unifier σ of ψ_1 can be constructed with the properties (1), (2). Assume that $\sigma = \{X_{j_1} \mapsto t_1, \ldots, X_{j_s} \mapsto t_s\}$. Then the corresponding conjunction ϕ_i is given by:

$$X_{j_1} = t_1, \ldots, X_{j_s} = t_s, \psi_2\sigma$$

□

Example 9 Consider $\phi \equiv X_1 = X_2, X_1 = f(X_4), X_2 = f(X_3)$. The normal form of this term constraint is $\phi' \equiv X_1 = f(X_4), X_3 = X_4, X_2 = f(X_4)$. □

In contrast to term constraints, atomic hom-constraints in general cannot so easily be split in such a way that all left-hand sides are variables. As a small example, consider the disequality

$$g(X) \neq_{\mathrm{H}} a$$

which would be vacuously true in case of a term constraint. If H is defined in such a way that it deletes the constructor g (i.e., $\mathrm{H} = \{g \mapsto X_1, \ldots\}$), then for the substitution $\theta = \{X \mapsto a\}$, the disequality is false (independent of the value of $\mathrm{H}a$ in this case).

2.2 Tree Automata with Term Constraints

In [RS10], we have introduced TCA (term-constrained automata), the class of tree automata with term equality and term disequality constraints. Such automata are strictly more expressive than the automata with constraints between brothers of [BT92]. In the Horn clause formalism, constraints between brothers are equalities and disequalities between *variables* while TCA allow comparisons between arbitrary terms over the variables that occur in the core clause. Since the *constraint language* of TCA includes both disequality and equality constraints, it is closed under complementation. It is also closed under intersection, and under disjunction – when representing disjunctions as alternative clauses that only differ

in their constraints. Therefore, the languages accepted by TCA are also closed under Boolean operations. As the main result for TCA we show that the standard decision problems for tree automata languages, and, in particular, emptiness are decidable for this class of automata with constraints. Roughly, the main idea for deciding emptiness of TCA is based on substitutions of variables X with terms t for equalities $X = t$ occurring in normal form term constraints. In the end, this yields a class of tree automata with complex heads and term disequalities, which we consider in the following section. We then return to TCA in Section 2.2.2.

2.2.1 Generalized Tree Automata with Term Disequalities

This class *TCA*$_{\neq}$ of *generalized* tree automata with term disequalities allows the heads of clauses to contain arbitrary terms t with $\mathrm{vars}(t) = \{X_1, \ldots, X_k\}$ instead of requiring the form $f(X_1, \ldots, X_k)$. Thus, these terms may be non-linear (multiple occurrences of the same variable) and of arbitrary depth.

Ordinary tree automata with term disequalities can be represented as finite sets of Horn clauses of the form:

$$p(X_1, \ldots, X_k) \Leftarrow p_1(X_1), \ldots, p_k(X_k), \phi$$

where ϕ is finite conjunction of term disequalities:

$$\phi = s_1 \neq t_1 \wedge \ldots \wedge s_n \neq t_n$$

for terms s_i and t_i over variables from the set $\{X_1, \ldots, X_k\}$.

TCA$_{\neq}$ constitute a generalization of this model that allows the heads to contain arbitrary terms over (all) the variables $X_1, \ldots, X_k$. Thus, the clauses of a TCA$_{\neq}$ $\mathcal{A}$ have the form:

$$p(t) \Leftarrow p_1(X_1), \ldots, p_k(X_k), \phi \qquad (k \geq 0)$$

where ϕ is a conjunction of disequalities $s_i \neq t_i$ with $\mathrm{vars}(\phi) \subseteq \{X_1, \ldots, X_k\}$, and t is a term with $\mathrm{vars}(t) = \{X_1, \ldots, X_k\}$. The number of disequalities in a clause $c \in \mathcal{A}$ is denoted $\mathrm{dc}(c)$.

With TCA$_{\neq}$, it is also possible to express certain kinds of *equalities*. This is not surprising as TCA$_{\neq}$ may be the result of our transformation of a TCA to a TCA$_{\neq}$ which, while not having *explicit* equalities, still is equivalent to the original automaton with equality constraints.

Example 10 The (non-regular) language $L = \{f(t, t) \mid t \in \mathcal{T}_\Sigma\}$ is expressible by the TCA$_{\neq}$

$$p(f(X_1, X_1)) \quad \Leftarrow$$

□

By Lemma 1, the membership problem is decidable for TCA by an algorithm using dynamic programming. In the exact same manner, membership can be decided for TCA$_{\neq}$. We obtain:

Lemma 4 *TCA$_{\neq}$-membership is decidable, i.e., for every TCA$_{\neq}$ $\mathcal{A}$, predicate p, and ground term t, it is decidable whether or not $t \in [\![p]\!]_{\mathcal{A}}$.* □

The remainder of this section contains a proof that also emptiness is efficiently decidable for TCA$_{\neq}$. Assume that the predicate q is defined by a TCA$_{\neq}$ $\mathcal{A}$. An algorithm for semi-deciding non-emptiness of q computes for $i \geq 1$ the sets $[\![p]\!]^i_{\mathcal{A}}$ for all predicates p until $[\![q]\!]^i_{\mathcal{A}} \neq \emptyset$. Thereby, the sets $[\![p]\!]^i_{\mathcal{A}}$ can be computed from the sets $[\![p']\!]^{i-1}_{\mathcal{A}}$ by applying the implications $c \in \mathcal{A}$, starting from $[\![p]\!]^0_{\mathcal{A}} = \emptyset$ for all predicates p.

For *automata without constraints*, each round $i \geq 1$ finds all recognized trees of *depth* at most i. Assume that n predicates occur in a given set $\mathcal{A}$ of automata clauses. Since each predicate p with $[\![p]\!]_{\mathcal{A}} \neq \emptyset$ recognizes at least one tree of depth $\leq n$, after a linear number of rounds (non-)emptiness is decided. Constraints, however, can *delay* the derivation process: if the term disequality constraint ϕ of a clause $c \equiv p(t) \Leftarrow p_1(X_1), \ldots, p_k(X_k), \phi$ is false for every substitution θ with $X_i\theta \in [\![p_i]\!]^j_{\mathcal{A}}, i \in \{1, \ldots, k\}$, then c cannot produce a tree in round $j+1$ even if $[\![p_i]\!]^j_{\mathcal{A}} \neq \emptyset$ for *all* i. Still, c may later produce a tree for a suitable combination of trees $t_i \in [\![p_i]\!]^m_{\mathcal{A}}, m > j$.

We establish an upper bound for the number of rounds which are needed in order to decide emptiness of generalized tree automata with term disequalities. By a counting argument, it suffices to increase the sets $[\![p]\!]^i_{\mathcal{A}}$ only up to a fixed number of trees. The claim is that for proving non-emptiness of a predicate q, at most $(\sum_{c \in \mathcal{A}} \mathrm{dc}(c)) + 1$ trees for each predicate $p \neq q$ suffice. This claim is based on the following lemma, which says that each term disequality constraint ϕ of a clause c "filters out" at most $\mathrm{dc}(c)$ trees.

Lemma 5 *Let $\mathcal{A}$ be a TCA$_{\neq}$ and $c \in \mathcal{A}$ a clause $q(t) \Leftarrow p_1(X_1), \ldots, p_k(X_k), \phi$. Assume that there is a number $d \geq 0$ such that $\exists\theta\, \forall i \in \{1, \ldots, k\}\ X_i\theta \in [\![p_i]\!]^d_{\mathcal{A}} \wedge \theta \models \phi$. Then it holds that $|[\![q]\!]^{d+1}_{\mathcal{A}}| \geq \max_{1 \leq i \leq k} \{|[\![p_i]\!]^d_{\mathcal{A}}| - \mathrm{dc}(c)\}$.* □

Based on this observation, the following theorem can be proved.

Theorem 1 *Let $\mathcal{A}$ be a TCA$_{\neq}$ which has n predicates. Then for all predicates p it holds that $[\![p]\!]_{\mathcal{A}} = \emptyset$ if and only if $[\![p]\!]^{n(d+1)}_{\mathcal{A}} = \emptyset$ where $d = \sum_{c \in \mathcal{A}} \mathrm{dc}(c)$.* □

Original proofs for both the lemma and the theorem can be found in [RS10]. Instead of repeating them here, more general versions of these results are proved in Section 3.3 — namely Lemma 19 and Theorem 22. Although the principal

proof methods agree, the more general versions additionally take care of auxiliary variables and decide k-finiteness instead of just emptiness.

In particular, by Theorem 1 emptiness is decidable for TCA$_{\neq}$.

Corollary 6 *For every TCA$_{\neq}$ $\mathcal{A}$, it can be effectively decided for each predicate p whether $[\![p]\!]_{\mathcal{A}} = \emptyset$.* □

2.2.2 Tree Automata with Term Equalities and Disequalities

We now return to automata with term constraints (TCA). There, atomic constraints are either disequalities $s \neq t$ or equalities $s = t$ for terms s and t. So the clauses are of this form:

$$p(b(X_1, \ldots, X_k)) \;\Leftarrow\; p_1(X_1), \ldots, p_k(X_k), \phi$$

where $b \in \Sigma$, and ϕ is a conjunction of equalities and disequalities which may only mention variables from $\{X_1, \ldots, X_k\}$.

Term constraints versus brother constraints. Let us call the automata of Bogaert and Tison with constraints between brothers *VCA* (variable-constrained automata). In addition to constraints on variables as in VCA, TCA in general also allow for comparisons like $X_i \neq f(X_j)$, which express relations between subtrees at different depths in the given trees. We prove the following theorem.

Theorem 2 *TCA are strictly more expressive than VCA.*

Proof. By definition, each VCA V can be considered as a TCA defining the same language. For the reverse direction, consider the TCA $\mathcal{A} = \{c_1, c_2, c_3\}$ with:

$$\begin{array}{lcll} c_1 & \equiv & p(b(X_1, X_2)) & \Leftarrow \quad q(X_1), q(X_2), X_1 = f(X_2) \\ c_2 & \equiv & q(f(X_1)) & \Leftarrow \quad q(X_1) \\ c_3 & \equiv & q(a) & \Leftarrow \end{array}$$

Here, $[\![p]\!]_{\mathcal{A}} = \{b(f^{i+1}(a), f^i(a)) \mid i \geq 0\}$. We show by contradiction that no VCA $\mathcal{A}'$ exists that defines a predicate p' with $[\![p']\!]_{\mathcal{A}'} = [\![p]\!]_{\mathcal{A}}$. Let us therefore assume that such a VCA exists. Since $[\![p]\!]_{\mathcal{A}}$ contains infinitely many elements, but $\mathcal{A}'$ has only finitely many clauses, $\mathcal{A}'$ must contain a clause c:

$$p'(b(X_1, X_2)) \Leftarrow q_1(X_1), q_2(X_2), \phi_c$$

such that c can produce infinitely many trees $t \in [\![p']\!]_{\mathcal{A}'} \subseteq [\![p]\!]_{\mathcal{A}}$. Let t_1, t_2 be two such trees with $t_1 = b(f^{i+1}(a), f^i(a))$ and $t_2 = b(f^{j+1}(a), f^j(a))$, $j > i \geq 0$. By construction, we have:

(1) $f^{j+1}(a) \in [\![q_1]\!]_{\mathcal{A}'}$, $f^i(a) \in [\![q_2]\!]_{\mathcal{A}'}$, but

(2) $b(f^{j+1}(a), f^i(a)) \notin [\![p']\!]_{\mathcal{A}'} \subseteq \{b(f^{k+1}(a), f^k(a)) \mid k \geq 0\}$, and

(3) $t_1, t_2 \in [\![p']\!]_{\mathcal{A}'}$ by application of c.

In the following we show that every combination of (dis)equality constraints between X_1, X_2 in ϕ_c leads to a contradiction:

(i) If either $X_1 \neq X_1$ or $X_2 \neq X_2$ occurs in ϕ_c, then ϕ_c is unsatisfiable and therefore (3) cannot hold.

(ii) If either $X_1 = X_2$ or $X_2 = X_1$ occurs in ϕ_c, neither t_1 nor t_2 can be generated by c, again contradicting (3).

(iii) Both $X_1 = X_1$ and $X_2 = X_2$ are tautologies. Therefore, assume now that ϕ_c either is a tautology or is equivalent to $X_1 \neq X_2$. Then $f^{j+1}(a) \neq f^i(a)$, and therefore $\theta \models \phi_c$ for $\theta(X_1) = f^{j+1}(a)$, $\theta(X_2) = f^i(a)$. We conclude that $b(f^{j+1}(a), f^i(a)) \in [\![p']\!]_{\mathcal{A}'}$, contradicting (2).

□

Path constraints can simulate term constraints. While TCA are strictly more expressive than VCA, they are strictly less expressive than automata with path constraints. This is indicated by the fact that emptiness is undecidable for automata with path equalities [Mon81]. Recall that paths are sequences of integers that identify subterms. E.g., the constraint $X_1.2 = X_2$ in the body of a clause $p(h(X_1, X_2)) \Leftarrow r(X_1), s(X_2), X_1.2 = X_2$ expresses that the second argument of X_1 must exist and equal X_2. Likewise, the constraint $X_1.2 \neq X_2$ in the body of a clause $p(h(X_1, X_2)) \Leftarrow r(X_1), s(X_2), X_1.2 \neq X_2$ expresses that either the path $X_1.2$ does not exist (i.e., the root symbol is of arity at most 1) or the second subterm of the value of X_1 is different from the value of X_2. It is not difficult to see that automata with path constraints can simulate TCA.

Lemma 7 *For every TCA $\mathcal{A}$, a tree automaton with path constraints $\mathcal{A}'$ can be constructed such that $[\![p]\!]_{\mathcal{A}} = [\![p]\!]_{\mathcal{A}'}$ for every predicate p.*

Proof sketch. Assume that the maximal depth of a term occurring in a constraint of $\mathcal{A}$ is k. The idea of the construction is that $\mathcal{A}'$ records the topmost constructors up to depth k of the current argument within the predicate and then enforces the required equalities or disequalities by means of path constraints. The clause

$$p(h(X_1, X_2)) \Leftarrow r(X_1), s(X_2), X_1 = f(X_2, X_2)$$

for instance, is simulated by the clauses:

$$p_{h(\_,\_)}(h(X_1, X_2)) \Leftarrow r_{f(\_,\_)}(X_1), s_t(X_2), X_2 = X_1.1, X_2 = X_1.2$$

for arbitrary patterns t of depth 1. □

Determinism, Closure. A finite set $\mathcal{A}$ of automata clauses with (term) constraints over a fixed ranked alphabet Σ is called *deterministic* if for each ground term $t \in \mathcal{T}_\Sigma$ there is at most one predicate p with $t \in [\![p]\!]_\mathcal{A}$. If *at least* one predicate p with $t \in [\![p]\!]_\mathcal{A}$ exists for each ground term, then $\mathcal{A}$ is called *total*. $\mathcal{A}$ is *total deterministic* if $\mathcal{A}$ is both total and deterministic, i.e, if for each $t \in \mathcal{T}_\Sigma$ there is exactly one predicate p with $t \in [\![p]\!]_\mathcal{A}$.

It shall now be shown that for every TCA $\mathcal{A}$, an equivalent total deterministic automaton $\mathcal{A}'$ can be constructed with $[\![p]\!]_{\mathcal{A}'} = [\![p]\!]_\mathcal{A}$ for all p by means of the powerset construction. The powerset construction is a standard automata technique. A general construction for constrained automata can, e.g., be found in [CDG$^+$07]. For TCA, a powerset automaton $\mathcal{A}'$ may be constructed as follows. The predicates of $\mathcal{A}'$ are taken from the powerset of $\mathcal{A}$'s predicates. For notational convenience, let us assume that clauses without explicit constraints occurring in $\mathcal{A}$ are replaced by corresponding clauses with an explicit constraint ϕ (equivalent to `true`). Then we define

$$P(b(X_1, \ldots, X_k)) \Leftarrow P_1(X_1), \ldots, P_k(X_k), \phi' \in \mathcal{A}'$$

if and only if P is a subset of the set

$$\{p \mid p(b(X_1, \ldots, X_k)) \Leftarrow p_1(X_1), \ldots, p_k(X_k), \phi \in \mathcal{A}, p_1 \in P_1, \ldots, p_k \in P_k\}$$

and

$$\begin{aligned} \phi' \;=\; & \bigwedge_{p \in P} \Big(\bigvee_{\substack{p(b(X_1,\ldots,X_k)) \Leftarrow p_1(X_1),\ldots,p_k(X_k),\phi'' \in \mathcal{A} \\ p_i \in P_i}} \phi'' \Big) \\ \wedge \; & \bigwedge_{p \notin P} \Big(\bigwedge_{\substack{p(b(X_1,\ldots,X_k)) \Leftarrow p_1(X_1),\ldots,p_k(X_k),\phi'' \in \mathcal{A} \\ p_i \in P_i}} \neg\phi'' \Big) \;. \end{aligned}$$

Note that in the construction of ϕ', an atomic constraint $s = t$ may result from a constraint $s \neq t$ and vice versa. The construction thus relies on the closure of the constraint language not only under *positive* Boolean operations but also under negation. Since each constraint occurring in $\mathcal{A}'$ can be transformed to DNF ψ, each clause of $\mathcal{A}'$ is equivalent to a finite set of clauses in TCA form. The sizes of the DNFs are exponentially bounded. We therefore conclude:

Theorem 3 *For every TCA, an equivalent total deterministic TCA exists and can be constructed in exponential time.* □

Example 11 The powerset construction for the set $\mathcal{A}$ consisting of the two clauses

$$\begin{array}{lcl} p(a) & \Leftarrow & \\ p(f(X_1)) & \Leftarrow & p(X_1), X_1 = a \end{array}$$

over $\Sigma = \{a, f\}$ results in the clauses:

$$\begin{array}{lcl} \{p\}(a) & \Leftarrow & \\ \{p\}(f(X_1)) & \Leftarrow & \{p\}(X_1), X_1 = a \\ \emptyset(f(X_1)) & \Leftarrow & \{p\}(X_1), X_1 \neq a \\ \emptyset(f(X_1)) & \Leftarrow & \emptyset(X_1) \end{array}$$

If Σ also contains a symbol b of arity 0, we additionally must add the clause $\emptyset(b) \Leftarrow$. □

Let $\mathcal{A}$ be a TCA with predicates $p_1, \ldots, p_n$, and $\mathcal{A}'$ the equivalent total deterministic automaton according to the presented powerset construction. Recall that $\mathcal{T}_\Sigma$ denotes the set of all trees constructible with symbols of Σ. Intersection, union, and complementation of languages defined by the predicates p_i of $\mathcal{A}$ can be obtained by the following three equivalences.

1. $[\![p_{i_1}]\!]_\mathcal{A} \cap \ldots \cap [\![p_{i_m}]\!]_\mathcal{A} = \bigcup\{[\![P]\!]_{\mathcal{A}'} \mid P \supseteq \{p_{i_1}, \ldots, p_{i_m}\}\}$
2. $[\![p_{i_1}]\!]_\mathcal{A} \cup \ldots \cup [\![p_{i_m}]\!]_\mathcal{A} = \bigcup\{[\![P]\!]_{\mathcal{A}'} \mid \exists j \in \{1, \ldots, m\} : p_{i_j} \in P\}$
3. $\mathcal{T}_\Sigma \setminus [\![p_i]\!]_\mathcal{A} = \bigcup\{[\![P]\!]_{\mathcal{A}'} \mid p_i \notin P\}$

Let $q_1, \ldots, q_{n'}$ denote the predicates of $\mathcal{A}'$. In order to determine the union of a subset $q_{i_1}, \ldots, q_{i_r}$ of those predicates, we introduce a new predicate q and add the following clauses to $\mathcal{A}'$.

$$q(f(X_1, \ldots, X_k)) \Leftarrow \alpha, \phi \qquad q_{i_j}(f(X_1, \ldots, X_k)) \Leftarrow \alpha, \phi \in \mathcal{A}', 1 \leq j \leq r$$

Then $[\![q]\!]_{\mathcal{A}'} = \bigcup\{[\![q_{i_j}]\!]_{\mathcal{A}'} \mid 1 \leq j \leq r\}$. Altogether, we obtain:

Theorem 4 *Tree automata with term equalities and disequalities (TCA) are closed under union, intersection, and complementation.* □

From Boolean closure and decidability of emptiness, it follows that also decision problems such as universality or inclusion are decidable for TCA.

Saturating deterministic TCA. Recall from Section 2.1 (page 22) that each term constraint ϕ can be transformed to normal form, i.e., a form where all left-hand sides of equality constraints are variables, and the following holds:

1. if $X_i = t$ occurs in ϕ, then X_i does not occur anywhere else in ϕ, and

2. if $X_i = X_j$ occurs in ϕ, then $i < j$;

Let $\mathcal{A}$ denote a deterministic TCA. Our goal is to construct an equivalent deterministic TCA $\mathcal{A}'$ which is *saturated*. This means that every clause c given by

$$H \Leftarrow p_1(X_1), \ldots, p_k(X_k), \phi$$

has the following properties. Let L and R denote the set of variables occurring in left-hand sides and right-hand sides of equality constraints in ϕ, respectively. Then

1. ϕ is in normal form;
2. For all ground substitutions $\theta : \mathbf{V} \to \mathcal{T}_\Sigma$ with $\theta \models \phi$ and $X_i\theta \in [\![p_i]\!]_{\mathcal{A}'}$ for all $X_i \in R$, $X_j\theta \in [\![p_j]\!]_{\mathcal{A}'}$ also for all $X_j \in L$.

In this case, the literals $p_j(X_j), X_j \in L$, are redundant and therefore can be removed. Let $\phi \equiv \bigwedge_{X_i \in L}(X_i = t_i) \wedge \phi'$ where ϕ' contains disequality constraints only. Let $\tau : L \to \mathcal{T}_\Sigma(R)$ denote the substitution $\tau = \{X_i \mapsto t_i \mid X_i \in L\}$. Let $X_{j_1}, \ldots, X_{j_r}$ be an enumeration of the variables of the clause which are not in L. Then the clause:

$$H\tau \Leftarrow p_{j_1}(X_{j_1}), \ldots, p_{j_r}(X_{j_r}), \phi'$$

has disequality constraints only and is equivalent to the clause c. Thus, we obtain:

Theorem 5 *For every saturated deterministic TCA $\mathcal{A}$, a TCA$_\neq$ $\mathcal{A}'$ can be constructed such that $[\![p]\!]_{\mathcal{A}} = [\![p]\!]_{\mathcal{A}'}$ for every predicate p.* □

In particular, this means that emptiness for saturated deterministic TCA is decidable.

Saturation steps replace (parts of) preconditions $p(X), X = t$ by constraints which express that if the leaves of t are accepted by the *enforced predicates*, then $p(t)$ also holds. The enforced predicates of variables X_i are given through the literals $p'(X_i)$ of the precondition while for atoms a there can be only one predicate p_a which accepts a in a deterministic automaton. In particular, p_a is given through a clause $p_a(a) \Leftarrow \phi$ where ϕ is equivalent to true.

Example 12 Consider the clause

$$q(f(X_1, X_2, X_3)) \quad \Leftarrow \quad q(X_1), p(X_2), q(X_3), \quad X_3 = g(f(X_2, X_1, a), X_2)$$

Assume that $a \in [\![q]\!]_{\mathcal{A}}$. Then Figure 2.3 shows one possible construction for the right-hand side of the constraint, using clauses

$$\begin{array}{lcl} r(f(X_1, X_2, X_3)) & \Leftarrow & p(X_1), q(X_2), q(X_3), \phi_1 \\ q(g(X_1, X_2)) & \Leftarrow & r(X_1), p(X_2), \phi_2 \end{array}$$

We have to construct a constraint ψ which expresses the conditions under which, for a given ground substitution $\theta = \{X_1 \mapsto t_1, X_2 \mapsto t_2\}$ with

$$t_1 \in [\![q]\!]_{\mathcal{A}}, \quad t_2 \in [\![p]\!]_{\mathcal{A}}$$

these two clauses can be applied. I.e., if $\psi \models \theta$, then the following must also hold:

$$f(t_2, t_1, a) \in [\![r]\!]_{\mathcal{A}} \text{ and } g(f(t_2, t_1, a), t_2) \in [\![q]\!]_{\mathcal{A}}$$

We choose:

$$\psi = \phi_1[X_2/X_1, X_1/X_2, a/X_3] \wedge \phi_2[f(X_2, X_1, a)/X_1, X_2/X_2]$$

Then the variable X_3 in the head can be substituted by the term $g(f(X_2, X_1, a), X_2)$, and $q(X_3), X_3 = g(f(X_2, X_1, a), X_2)$ can be replaced by ψ. Thus, the new clause

$$q(f(X_1, X_2, g(f(X_2, X_1, a), X_2))) \Leftarrow q(X_1), p(X_2), \quad \psi$$

is obtained and added to the current set of clauses. If ψ contains equality constraints, the resulting clause is not saturated yet. The constraint ψ, however, can now only contain X_1, X_2 while variable X_3 has been removed. Intuitively, this is the reason that saturation terminates. □

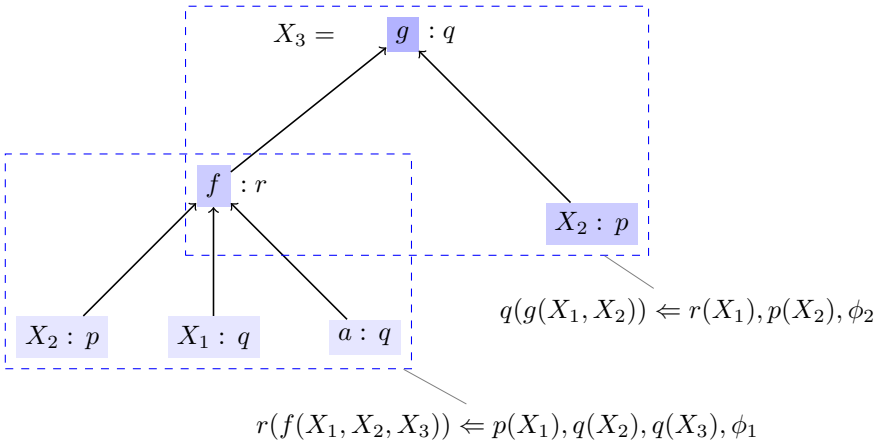

Figure 2.3: Building the right-hand side of $X_3 = g(f(X_2, X_1, a), X_2)$ where a, X_1, and X_2 are recognized by q, q, and p, respectively

Example 13 Consider the deterministic set of clauses $\mathcal{C} = \{c_1, c_2, c_3, c_4\}$ with

$$\begin{array}{lcll} c_1 & \equiv & p(h(X_1, X_2)) & \Leftarrow \quad p_1(X_1), p_2(X_2), \quad X_1 = f(X_2, X_2), X_1 \neq a \\ c_2 & \equiv & p_1(f(X_1, X_2)) & \Leftarrow \quad p_2(X_1), p_2(X_2), \quad X_2 = a \\ c_3 & \equiv & p_2(g(X_1)) & \Leftarrow \quad p_2(X_1) \\ c_4 & \equiv & p_2(a) & \Leftarrow \end{array}$$

over $\Sigma = \{a, h, f, g\}$. Here, clauses c_2, c_3 and c_4 are saturated, while clause c_1 is not. By adding the constraint $X_2 = a$, we obtain from c_1 the clause:

$$p(h(X_1, X_2)) \Leftarrow p_1(X_1), p_2(X_2), X_1 = f(X_2, X_2), X_2 = a, X_1 \neq a$$

or:

$$p(h(X_1, X_2)) \Leftarrow p_1(X_1), p_2(X_2), X_1 = f(a, a), X_2 = a, X_1 \neq a$$

which is saturated and equivalent to c_1. The set $\mathcal{C}'$ of an equivalent TCA$_{\neq}$ thus is given by $\mathcal{C}' = \{c_1', c_2', c_3, c_4\}$ with

$$\begin{array}{lcll} c_1' & \equiv & p(h(f(a,a),a)) & \Leftarrow \\ c_2' & \equiv & p_1(f(X_1,a)) & \Leftarrow \quad p_2(X_1) \end{array}$$

and the languages of the predicates p, p_1, p_2 are given by $[\![p]\!]_{\mathcal{C}} = \{h(f(a,a),a)\}$, $[\![p_1]\!]_{\mathcal{C}} = \{f(g^i(a), a) \mid i \geq 0\}$, and $[\![p_2]\!]_{\mathcal{C}} = \{g^i(a) \mid i \geq 0\}$. □

In the following, let $\mathcal{A}$ denote a fixed deterministic TCA. We observe:

Theorem 6 *Let* $\mathbf{V}$ *denote a set of variables. For every* $X_i \in \mathbf{V}$*, let* p_i *denote a predicate. Then for every term* $t \in \mathcal{T}_\Sigma(\mathbf{V})$ *and predicate* p*, a term constraint* $\Psi_{t,p}$ *can be constructed such that for all ground substitutions* $\theta : \mathbf{V} \to \mathcal{T}_\Sigma$ *with* $X_i\theta \in [\![p_i]\!]_{\mathcal{A}}$ *for all* $X_i \in \mathbf{V}$*,*

$$t\theta \in [\![p]\!]_{\mathcal{A}} \quad \text{iff} \quad \theta \models \Psi_{t,p}$$

Proof. We proceed by induction on the structure of t, with the two base cases:

(1) if $t = X_i$, $X_i \in \mathbf{V}$, then $\Psi_{t,p}$ is true if $p = p_i$, and false otherwise;

(2) if $t = a$, $a \in \Sigma$, then $\Psi_{t,p}$ is true if $p(a) \Leftarrow \phi \in \mathcal{A}$ where ϕ is equivalent to true, and false otherwise.

In both cases, the assertion of the theorem is satisfied. Note for case (2) that ϕ is variable-free, and therefore it is either equivalent to true or equivalent to false.

For the induction step, let $t = f(t_1, \ldots, t_k)$. Let θ' denote the substitution with $X_i\theta' = t_i$ for $i = 1, \ldots, k$. Then we define $\Psi_{t,p}$ as the disjunction:

$$\bigvee\{\phi\theta' \wedge \Psi_{t_1,p_1} \wedge \ldots \wedge \Psi_{t_k,p_k} \mid p(f(X_1, \ldots, X_k)) \Leftarrow p_1(X_1), \ldots, p_k(X_k), \phi \in \mathcal{A}\}$$

Now assume that θ is a ground substitution such that $t\theta \in [\![p]\!]_{\mathcal{A}}$. By definition, $t\theta = f(t_1\theta, \ldots, t_k\theta)$. Since $\mathcal{A}$ is deterministic, there exist predicates $p_1, \ldots, p_k$ and a clause $p(f(X_1, \ldots, X_k)) \Leftarrow p_1(X_1), \ldots, p_k(X_k), \phi$ such that the following holds:

1. $t_i\theta \in [\![p_i]\!]_{\mathcal{A}}$ for all $i = 1, \ldots, k$;
2. $\theta'\theta \models \phi$ where $X_i\theta'\theta = t_i\theta$.

By induction hypothesis, the first item implies that $\theta \models \Psi_{t_i,p_i}$ for all i. From the second item, we deduce that $\theta \models \phi\theta'$ as well, and therefore by definition $\theta \models \Psi_{t,p}$. For the reverse implication, assume that $\theta \models \Psi_{t,p}$. Then some clause $p(f(X_1, \ldots, X_k)) \Leftarrow p_1(X_1), \ldots, p_k(X_k), \phi$ exists such that

1. $\theta \models \Psi_{t_i,p_i}$ for all i, and
2. $\theta \models \phi\theta'$.

By induction hypothesis, we conclude from the first item that $t_i\theta \in [\![p_i]\!]_{\mathcal{A}}$ for all i. By the second item, $\theta'\theta \models \phi$. Overall, we therefore can apply the clause to the t_i to produce t, i.e., $t \in [\![p]\!]_{\mathcal{A}}$. □

Assume now that $H \Leftarrow p_1(X_1), \ldots, p_k(X_k), \phi$ is an automata clause with the constraint ϕ in normal form

$$(\bigwedge_{X_i \in L} X_i = t_i) \quad \wedge \quad \phi'$$

where ϕ' consists of disequalities only. (The set L consists of all left-hand sides of ϕ's equalities.) Let Ψ_{t_i,p_i} be the constraints as provided by Theorem 6 for the right-hand sides t_i of $X_i \in L$ in ϕ. Let $\overline{\phi}$ denote the constraint:

$$\phi \wedge \bigwedge \{\Psi_{t_i,p_i} \mid X_i \in L\}$$

Then the following holds:

Lemma 8 *For a ground substitution θ, the following statements are equivalent:*

1. *$\theta \models \phi$ and $X_i\theta \in [\![p_i]\!]_{\mathcal{A}}$ for all i;*
2. *$\theta \models \overline{\phi}$ and $X_i\theta \in [\![p_i]\!]_{\mathcal{A}}$ for all $X_i \notin L$.*

Proof. The constraint $\overline{\phi}$ need not be in normal form, but is equivalent to a (possibly empty) finite disjunction of constraints $\overline{\phi_1} \vee \ldots \vee \overline{\phi_s}$ where each $\overline{\phi_j}$ is in normal form. The clause $H \Leftarrow p_1(X_1), \ldots, p_k(X_k), \phi$ then is equivalent to the set of clauses $c_j, j = 1, \ldots, s$, given by

$$H \Leftarrow p_1(X_1), \ldots, p_k(X_k), \overline{\phi_j}$$

Let L_j denote the set of left-hand sides of equalities in $\overline{\phi_j}$. Then $L \subseteq L_j$ for all j. Whenever $L = L_j$, the clause c_j is saturated. Otherwise, we apply the same

procedure to the clause c_j. Since $L \subset L_j$, i.e., the set of variables occurring in left-hand sides of equalities has become strictly larger, this refinement may occur at most at k levels. □

Overall, we therefore have proved the following theorem:

Theorem 7 *For every deterministic TCA $\mathcal{A}$, a saturated deterministic TCA $\mathcal{A}'$ can be effectively constructed such that $[\![p]\!]_{\mathcal{A}} = [\![p]\!]_{\mathcal{A}'}$ for every predicate p.* □

By Theorem 3, for every TCA, an equivalent deterministic TCA can be constructed, which, by Theorem 7, can be saturated. By Theorem 5, we furthermore know that every saturated deterministic TCA can be transformed to an equivalent $\text{TCA}_{\neq}$ — to which we can apply the emptiness test from Corollary 6. In summary, we therefore have obtained:

Theorem 8 *For every TCA $\mathcal{A}$, it can be effectively decided for every predicate p whether $[\![p]\!]_{\mathcal{A}} = \emptyset$.* □

2.3 Complex Preconditions, Auxiliary Variables

In this section, it shall be proven that term equalities in combination with complex preconditions $p(t)$, where t is an arbitrary term, yield an undecidable class when extending automata clauses accordingly. The proof reduces the undecidable Post Correspondence Problem (PCP) to the problem of deciding emptiness of a predicate for a set $\mathcal{A}$ of automata clauses with term equalities and complex preconditions. The constructed set $\mathcal{A}$ will only contain one single term equality (in fact, a comparison of two variables suffices).

Let P $= ((a_1, b_1), \ldots, (a_m, b_m))$ be an instance of the PCP over a binary alphabet $\{a, b\}$. Let $a_i(t)$ and $b_i(t)$ be the corresponding codings of the words a_i and b_i, respectively, applied to the term t (e.g., if $a_i = bba$ then $a_i(t)$ is given by $b(b(a(t)))$). The alphabet Σ for the set $\mathcal{A}$ consists of the constant symbol 0, the unary symbols a and b, and a binary constructor h. First, a helper predicate $\top_{\neq 0}$ is defined which accepts every term with the exception of 0 as follows.

$$\begin{array}{lcl} \top_{\neq 0}(a(X_1)) & \Leftarrow & \top(X_1) \\ \top_{\neq 0}(b(X_1)) & \Leftarrow & \top(X_1) \\ \top_{\neq 0}(h(X_1, X_2)) & \Leftarrow & \top(X_1), \top(X_2) \end{array}$$

Then a predicate s is defined which accepts all terms $h(t, t)$ with $t \neq 0$:

$$s(h(X_1, X_2)) \Leftarrow \top_{\neq 0}(X_1), \top(X_2),\ X_1 = X_2 \tag{2.1}$$

The idea of the construction is that s will first accept the term $h(t,t)$ for the coding $t = a_{i_1}(\ldots(a_{i_k}(0))\ldots) = b_{i_1}(\ldots(b_{i_k}(0))\ldots)$ of a solution $i_1,\ldots,i_k, k \geq 1$, to P — which then is verified by the following clauses with a complex precondition:

$$s(h(X_1,X_2)) \Leftarrow \top(X_1), \top(X_2),\ s(h(a_i(X_1),b_i(X_2))) \qquad 1 \leq i \leq m$$

The condition for accepting a solution then is that the initial terms t could be simultaneously reduced to 0:

$$\mathrm{accept}(b(X_1)) \Leftarrow \top(X_1),\ s(h(0,0))$$

The predicate accept thus is empty if and only if P has no solution. Hence, the following lemma has been proved.

Lemma 9 *Emptiness is undecidable for automata clauses with term equalities when complex preconditions $p(t)$, where t is an arbitrary term, may occur in the preconditions of the clauses.* □

With the exception of the clause (2.1), the presented construction for TCA enhanced with complex preconditions contains no equality constraints. This clause, however, is equivalent to the following TCA$_{\neq}$-clause with a non-linear head:

$$s(h(X_1,X_1)) \Leftarrow \top_{\neq 0}(X_1), \top(X_1)$$

Consequently, TCA$_{\neq}$ extended with additional complex preconditions cannot be decidable either.

Lemma 10 *Emptiness is undecidable for TCA$_{\neq}$ extended with complex preconditions $p(t)$, where t is an arbitrary term.* □

Auxiliary variables and term equalities. Auxiliary variables Y not occurring in the head in combination with term equalities also yield an undecidable automata class. This is because complex preconditions can easily be simulated by term equalities if auxiliary variables are available. E.g., $Y = h(X_1, g(X_2))$ together with $p(Y)$ occurring in the precondition, for a variable Y which does not occur elsewhere in the clause, is logically equivalent to $p(h(X_1, g(X_2)))$.

Lemma 11 *Emptiness is undecidable for automata clauses extended with term equalities when auxiliary variables Y not occurring in the head are allowed in the preconditions.* □

Interestingly, auxiliary variables do not increase expressiveness if only disequalities are considered. An alternative undecidability proof for TCA with complex preconditions is based on the observation that *path constraints*, which are known to be undecidable according to Mongy [Mon81], can be expressed by term constraints extended with auxiliary variables. The constraint $X_1.2 = X_2$, for instance, can be expressed by the disjunction $\bigvee \phi_f$ over all $f \in \Sigma$ with $ar(f) = r \geq 2$ where ϕ_f is given by $X_1 = f(Y_1,\ldots,Y_r) \wedge X_2 = Y_2$.

2.4 Tree Automata with Hom-Equalities

As has been shown in Section 2.3, term equalities cannot be combined with either complex preconditions or auxiliary variables not occurring in the head as both extensions yield undecidable automata classes. It shall now be shown that term equalities alone already yield undecidability when they are interpreted modulo a given tree homomorphism.

The proof that is used here is based on the observation that tree homomorphisms are able to *delete* variables occurring in the terms of heads of clauses, effectively producing auxiliary variables. It may be seen as an instantiation of the argument for Lemma 11 that auxiliary variables can simulate complex preconditions and therefore yield an undecidable class of constrained automata. Consequently, a construction analogous to the one from the proof of Lemma 9 is used, reducing the Post Correspondence Problem (PCP) to the problem of deciding emptiness of a predicate for a set of automata clauses with hom-equalities.

Theorem 9 *Emptiness is undecidable for automata clauses extended with hom-equalities.*

Proof. Let $\mathrm{P} = ((a_1, b_1), \ldots, (a_m, b_m))$ be an instance of the PCP over an alphabet $\{a, b\}$. Consider the following set $\mathcal{A} = \{(a), (b), (c), (d), (1), \ldots, (m)\}$ of automata clauses with hom-equalities over $\Sigma = \{0, a, b, h, f\}$

$$
\begin{array}{lllll}
(a) & \top_{\neq 0}(a(X_1)) & \Leftarrow & \top(X_1) & \\
(b) & \top_{\neq 0}(b(X_1)) & \Leftarrow & \top(X_1) & \\
(c) & p(h(X_1, X_2)) & \Leftarrow & \top(X_1), \top_{\neq 0}(X_2), & X_1 =_{\mathrm{H}} X_2 \\
(d) & q(h(X_1, X_2)) & \Leftarrow & \top(X_1), \quad p(X_2), & X_2 =_{\mathrm{H}} h(0, 0) \\
(i) & p(f(X_1, X_2, X_3)) & \Leftarrow & \top(X_1), \top(X_2), \; p(X_3), & X_3 =_{\mathrm{H}} h(a_i(X_1), b_i(X_2)) \\
 & (\text{for } 1 \leq i \leq m) & & &
\end{array}
$$

together with the tree homomorphism $\mathrm{H} = \{f \mapsto h(X_1, X_2)\}$. The goal is now to show that the predicate q is empty (i.e., $[\![q]\!]_{\mathcal{A}} = \emptyset$) if and only if P has no solution.

Clauses (a) and (b) define $\top_{\neq 0}$ so that it accepts exactly all terms of either the form $a(t)$ or the form $b(t)$. Clause (c) defines a predicate p which accepts all terms $h(a(t), a(t))$ and $h(b(t), b(t))$ for arbitrary terms t. This predicate will also accept the term $h(t, t)$ for the coding $t = a_{i_1}(\ldots(a_{i_k}(0))\ldots) = b_{i_1}(\ldots(b_{i_k}(0))\ldots)$ of a solution $i_1, \ldots, i_k, k \geq 1$, to P — provided that such a solution exists. Only if this is the case, the solution can then be verified by the clauses (i).

To see this, assume that t is an arbitrary term. For a term s of the form $s = a_i(s')$, let $a_i^-(s) = s'$ (analogously for $s = b_i(s')$). The predicate p accepts the term $h(t, t)$ according to clause (c). (The predicate p also accepts terms $h(t, t')$ with $t \neq t'$ but $t =_{\mathrm{H}} t'$, containing the constructor f, for which the following

deductions, however, are not possible.) If, for some i_1, t is of the form $a_{i_1}(t_1)$ and also of the form $b_{i_1}(t_2)$, then the term $s_1 = f(a^-_{i_1}(t), b^-_{i_1}(t), h(t,t))$ is in $[\![p]\!]_{\mathcal{A}}$ according to clause (i_1). One of the clauses (i) may again be used if and only if there is some i_2 such that $a^-_{i_1}(t)$ is of the form $a_{i_2}(t'_1)$ and $b^-_{i_1}(t)$ is of the form $b_{i_2}(t'_2)$. Only in this case, the term $s_2 = f(a^-_{i_2}(a^-_{i_1}(t)), b^-_{i_2}(b^-_{i_1}(t)), s_1)$ is in $[\![p]\!]_{\mathcal{A}}$ according to clause (i_2). This derivation chain may continue until some term $s_l = f(a^-_{i_l}(\ldots(a^-_{i_1}(t))\ldots), b^-_{i_l}(\ldots(b^-_{i_1}(t))\ldots), s_{l-1}) = f(0,0,s_{l-1})$ is proved to be in $[\![p]\!]_{\mathcal{A}}$ — if and only if t is coding a solution for P of length $l \geq 1$, i.e. $t = a_{i_1}(\ldots(a_{i_l}(0))\ldots) = b_{i_1}(\ldots(b_{i_l}(0))\ldots)$ for a sequence of indexes $i_1, \ldots, i_l$.

Finally, if the initial terms t could be simultaneously reduced to 0 in the first two arguments below the constructor f through application of the clauses (i), the predicate q is proved non-empty through clause (d). Thus, the PCP has a solution if and only if q is non-empty. □

Chapter 3

Normalization of Horn Clauses

In this chapter, we present $\mathcal{H}_1$-*normalization*, a procedure transforming finite sets of Horn clauses of the class $\mathcal{H}_1$ to finite sets of automata clauses [NNS02, GL05], and prove that this procedure can be adapted in such a way that instead of pure $\mathcal{H}_1$-clauses, $\mathcal{H}_1$-clauses extended with disequality constraints are transformed to automata clauses with the same sort of constraints. In particular, it is shown that this is possible for disequalities of terms as well as for disequalities of paths. Moreover, the procedure can also be adapted so as to allow term disequalities to be interpreted modulo a tree homomorphism. For each of these extensions, finite sets of automata clauses with corresponding constraints are decidable. The normalization framework thus provides a method for the decision of finite sets of $\mathcal{H}_1$-clauses with disequality constraints.

Based on the result from Section 2.2 that tree automata extended with term *equalities* and disequalities are decidable, a natural question is whether $\mathcal{H}_1$-clauses extended with such constraints can be transformed to automata form by an appropriate adaptation of the normalization procedure. This, however, is not the case. Undecidability arises both from the feature of $\mathcal{H}_1$-clauses to provide auxiliary variables which do not occur in the head and also from the feature of $\mathcal{H}_1$-clauses to allow complex preconditions. In combination with term equalities, either of these $\mathcal{H}_1$-features yields undecidability already for automata clauses, as shown in Section 2.3. Thus, normalization is not possible since the class of automata with term constraints is decidable according to Theorem 8.

3.1 $\mathcal{H}_1$-Clauses with Term Disequalities

In this section, the general framework of $\mathcal{H}_1$-normalization (in presence of constraints) is presented, instantiated with term disequalities [SR11]. The normalization procedure repeatedly applies three rules until the set of clauses becomes

saturated. Each rule adds finitely many simpler clauses which are implied by the current set of clauses. Then the subset of normal clauses subsumes the whole set of clauses, and all non-normal clauses may be removed. The three rules are

- *Resolution*, which applies a resolution step with a normal clause,
- *Splitting*, which removes auxiliary variables, i.e., variables only occurring in the precondition, and
- *Propagation*, which resolves clauses $p(X_1) \Leftarrow q_1(X_1), \ldots, q_m(X_1), \phi$ where ϕ contains no variable other than X_1, with normal clauses for *all* predicates $q_1, \ldots, q_m$ "simultaneously".

For the Splitting rule, the (number of) terms accepted by a predicate must be computed w.r.t. the current subset of normal clauses, i.e., we have to decide k-*finiteness*. Recall from page 21 that the constraint language enjoys positive Boolean closure. (Disjunctions are provided by alternative clauses.) We can therefore bring a given set of clauses to a form where each satisfiable constraint ϕ of a single clause is a conjunction of disequalities $Z \neq t$ for variables Z and terms t. In order to remove an auxiliary variable Y, we consider the (maximal) part

$$\phi_Y = Z_1 \neq t_1 \wedge \ldots \wedge Z_m \neq t_m$$

of such a conjunction in which Y occurs within each disequality (i.e., Y either equals Z_i or occurs in t_i). By the pigeon-hole principle, ϕ_Y can be satisfied if enough, i.e., $m+1$, term are available for the substitution of Y. Hence, one either removes from the considered clause all literals and disequalities mentioning Y or, if only $n \leq m$ terms $s_1, \ldots, s_n$ are available, uses each of the terms s_i as a concrete substitution for Y — removing Y in both cases. In the first case, only one new clause is added, while the latter case results in $n \leq m$ added clauses.

3.1.1 Bounding the Number of Terms

Assume that the predicate p is defined by a finite set $\mathcal{A}$ of automata clauses. In this section it is established that for all $k \geq 1$, it can be efficiently determined whether p accepts less than k terms or not. Moreover, if p accepts less than k terms, then these terms can be effectively computed. The presented method is then applied in Section 3.1.2 for the removal of auxiliary variables in the preconditions of $\mathcal{H}_1$-clauses.

We recall a simplified version of Lemma 5 which shows that k disequalities can *filter out* no more than k terms of any predicate in the precondition of a clause. To be more precise, if a literal $p(X)$ occurs in the precondition of an applicable clause c whose constraint consists of k term disequalities, and n terms can be

deduced for p, then at least $n - k$ terms can be deduced for the predicate in c's head. Here, *applicable* w.r.t. the set of automata clauses $\mathcal{A}$ defining c means that for the sequence of literals $q_1(X_1), \ldots, q_m(X_m)$ in c's precondition, there exists a sequence of terms $t_1 \in [\![q_1]\!]_{\mathcal{A}}, \ldots, t_m \in [\![q_m]\!]_{\mathcal{A}}$ such that the ground substitution $\theta = \{X_1 \mapsto t_1, \ldots, X_m \mapsto t_m\}$ satisfies c's constraint.

In Lemma 5, the heads may be *complex*, i.e., the depth of the terms in the heads has no upper-bound. Here, that depth is always equal to one, which allows a less complex proof technique, based on the *depth* of an accepted term t rather than the number of rounds of fixpoint iteration needed to deduce a fact $p(t)$ [SR11]. For a predicate p, a set of automata clauses $\mathcal{A}$ and $d \geq 0$, let $[\![p]\!]_{\mathcal{A},d}$ denote the set of terms $t \in [\![p]\!]_{\mathcal{A}}$ of depth at most d. Then the following holds.

Lemma 12 *Let $\mathcal{A}$ be a finite set of automata clauses with term disequalities, and c a clause in $\mathcal{A}$ of the form $q(t) \Leftarrow p_1(X_1), \ldots, p_k(X_k), \phi$. Assume furthermore that there is a number $d \geq 0$ and a substitution θ such that $X_i\theta$ is of depth at most d, and $X_i\theta \in [\![p_i]\!]_{\mathcal{A}}$ for all i, and, moreover, $\theta \models \phi$. Then $|[\![q]\!]_{\mathcal{A},d+1}| \geq \max_{1 \leq i \leq k}\{|[\![p_i]\!]_{\mathcal{A},d}| - \mathrm{dc}(c)\}$.* □

Based on this lemma, the following theorem can be proven.

Theorem 10 *Let $\mathcal{A}$ be a finite set of automata clauses with term disequalities. Then for every $k > 0$ and predicate p it can be effectively determined whether $|[\![p]\!]_{\mathcal{A}}| < k$. Moreover if $|[\![p]\!]_{\mathcal{A}}| < k$, then $[\![p]\!]_{\mathcal{A}}$ can be effectively computed.* □

For the case of automata clauses, this theorem generalizes Theorem 1 where it is proved that *emptiness* for such clauses is decidable. Proofs of Lemma 12 and Theorem 10 are given in [SR11]. In this work we refer to slightly more complex but also more general versions of these proofs as provided in Section 3.3 for Lemma 19 and Theorem 22. There, k-finiteness is proved decidable even for the case of clauses with complex heads and auxiliary variables. The proof uses an algorithm which essentially iteratively derives facts $p(t)$, t a ground term, but for each predicate p, the number of facts $p(t)$ that *need* to be considered for deciding k-finiteness can be bounded by $d + k$, where d denotes the number of atomic disequalities occurring in the whole set of automata clauses.

Using Lemma 2, the algorithm of Theorem 10 can be applied to normal clauses as well, resulting in the theorem that enables Splitting:

Theorem 11 *Assume that $\mathcal{N}$ is a finite set of normal clauses with term disequalities. Then for every subset A of predicates and integer $k \geq 0$, it is decidable whether or not the intersection $\bigcap_{p \in A} [\![p]\!]_{\mathcal{N}}$ contains less than k elements, and if so, these elements can be effectively computed.* □

3.1.2 Normalizing Constrained $\mathcal{H}_1$-Clauses

Based on the results from the previous section, we shall prove in this section the following main theorem that claims the existence of a normalization procedure also in presence of (term) disequality constraints.

Theorem 12 *For every finite set $\mathcal{C}$ of $\mathcal{H}_1$-clauses with term disequalities, a finite set $\mathcal{N}$ of normal clauses with term disequalities can be effectively constructed which is equivalent to $\mathcal{C}$, i.e., has the same least model, i.e., $[\![p]\!]_{\mathcal{C}} = [\![p]\!]_{\mathcal{N}}$ for all predicates p.* □

Recall from Section 2.1 (page 21) that the constraint language of $\mathcal{H}_1$-clauses with constraints is always closed under positive Boolean operations and that each atomic term disequality $s \neq t$ is equivalent to a finite disjunction of disequalities of the form $X = t$ where X is a variable. W.l.o.g. we shall therefore assume in the following that each term constraint ϕ is in normal form

$$X_{i_1} \neq t_1 \wedge \ldots \wedge X_{i_r} \neq t_r$$

where the X_{i_j} are variables and the t_i are arbitrary terms. We furthermore assume that each term constraint is satisfiable.

To the set $\mathcal{C}$, we add clauses which are implied by $\mathcal{C}$. Let $\mathcal{S}$ denote the current set of implied clauses and $\mathcal{N}$ the subset of normal clauses in $\mathcal{S}$. Thus, $\mathcal{S}$ initially equals $\mathcal{C}$. Then we exhaustively add new clauses to $\mathcal{S}$ according to the following three rules.

Resolution: Assume that $\mathcal{S}$ contains a clause

$$p(t_0) \Leftarrow \alpha_1,\ q(f(s_1, \ldots, s_k)),\ \alpha_2,\ \phi$$

with $k \geq 0$. If $\mathcal{N}$ contains a clause

$$q(X_1) \Leftarrow \psi$$

then we add the new clause

$$p(t_0) \Leftarrow \alpha_1,\ \alpha_2,\ \phi,\ \psi'$$

where $\psi' = \psi[f(s_1, \ldots, s_k)/X_1]$. If $\mathcal{N}$ contains a clause

$$q(f(X_1, \ldots, X_k)) \Leftarrow \beta,\ \psi$$

then we add the clause

$$p(t_0) \Leftarrow \alpha_1,\ \beta',\ \alpha_2,\ \phi,\ \psi'$$

where $\beta' = \beta[s_1/X_1, \ldots, s_k/X_k]$ and $\psi' = \psi[s_1/X_1, \ldots, s_k/X_k]$.

Splitting: Assume that $\mathcal{S}$ contains a simple clause

$$p(t) \Leftarrow \alpha, \phi$$

where α, ϕ contains a variable Y which does not occur in the term t in the head. Let $q_1(Y), \ldots, q_m(Y)$ denote the subsequence of α containing Y, and assume that ϕ has r disequalities containing Y. If $L = [\![q_1]\!]_{\mathcal{N}} \cap \ldots \cap [\![q_m]\!]_{\mathcal{N}}$ contains more than r distinct elements, then we add the clause

$$p(t) \Leftarrow \alpha', \phi'$$

where α' and ϕ' are obtained from α and ϕ, respectively, by removing the literals and disequalities which contain Y. If L has at most r elements, we add all clauses

$$p(t) \Leftarrow \alpha', \phi[t/Y] \ , \quad t \in L$$

where α' is obtained from α by removing all literals containing Y.

Propagation: Assume that $\mathcal{S}$ contains a simple clause

$$p(X) \Leftarrow q_1(X), \ldots, q_m(X), \phi$$

where ϕ contains no variable other than X. If for each $i = 1, \ldots, m$, $\mathcal{N}$ contains a clause $q_i(t) \Leftarrow \beta_i, \phi_i$, then we add the clause

$$p(t) \Leftarrow \beta_1, \ldots, \beta_m, \phi', \phi_1, \ldots, \phi_m$$

where $\phi' = \phi[t/X]$.

These rules generalize the corresponding rules for $\mathcal{H}_1$ as considered, e.g., in [NNS02], by a dedicated treatment of disequalities. Forgetting about disequalities, the rules can best be understood as a simple form of ordered resolution steps with splitting [GL05]. The ordering enforces that only heads of normal clauses may be resolved with literals in preconditions. Such resolution steps are applied in the Resolution and the Propagation rule. The Splitting rule allows to eliminate variables from a clause which do not occur in the head. This corresponds to splitting in resolution proofs.

Example 14 Consider the following set $\mathcal{C}$ of $\mathcal{H}_1$-clauses:

$$\begin{array}{lcll} p(a) & \Leftarrow & q(f(X,Y)), r(g(Y)), & X \neq Y \\ q(f(X,Y)) & \Leftarrow & p(Z), & Z \neq g(Y) \\ q(X) & \Leftarrow & r(X), & X \neq f(a,a) \\ r(X) & \Leftarrow & & X \neq g(a) \end{array}$$

where, for better readability, we use X, Y, Z as canonical names of variables occurring in heads. Applying Propagation to the last clause, we obtain:

$$\begin{array}{lll} r(a) & \Leftarrow & a \neq g(a) \\ r(g(X)) & \Leftarrow & g(X) \neq g(a) \\ r(f(X,Y)) & \Leftarrow & f(X,Y) \neq g(a) \end{array}$$

Since the precondition of $r(X)$ contains an empty sequence of literals, application of Propagation only amounts to substituting variable X with $a, g(X)$ and $f(X,Y)$, respectively. The constraints of the resulting clauses can be simplified, obtaining:

$$\begin{array}{lll} r(a) & \Leftarrow & \\ r(g(X)) & \Leftarrow & X \neq a \\ r(f(X,Y)) & \Leftarrow & \end{array}$$

Using these clauses, the Propagation rule furthermore gives us:

$$\begin{array}{lll} q(a) & \Leftarrow & a \neq f(a,a) \\ q(g(X)) & \Leftarrow & X \neq a, g(X) \neq f(a,a) \\ q(f(X,Y)) & \Leftarrow & f(X,Y) \neq f(a,a) \end{array}$$

Simplifying the constraints of these clauses, we obtain:

$$\begin{array}{lll} q(a) & \Leftarrow & \\ q(g(X)) & \Leftarrow & X \neq a \\ q(f(X,Y)) & \Leftarrow & f(X,Y) \neq f(a,a) \end{array}$$

These new clauses allow to deduce by Resolution from the first clause of the example set, the clause:

$$p(a) \Leftarrow f(X,Y) \neq f(a,a), Y \neq a, X \neq Y$$

Applying Splitting for variable Y then removes all disequality constraints. Using Splitting again, the new fact $p(a) \Leftarrow$ together with the second clause of the example set allow to deduce

$$q(f(X,Y)) \Leftarrow$$

Now none of the Resolution, Splitting, and Propagation rules yields a new clause. The resulting set $\mathcal{N}$ of normal clauses (with simplified constraints) thus consists

of:

$$\begin{array}{lll} r(a) & \Leftarrow & \\ r(g(X)) & \Leftarrow & X \neq a \\ r(f(X,Y)) & \Leftarrow & \\ \\ q(a) & \Leftarrow & \\ q(g(X)) & \Leftarrow & X \neq a \\ q(f(X,Y)) & \Leftarrow & f(X,Y) \neq f(a,a) \\ q(f(X,Y)) & \Leftarrow & \\ \\ p(a) & \Leftarrow & \end{array}$$

□

Correctness of the normalization procedure is proven analogously to the case of non-constrained $\mathcal{H}_1$-clauses, while the termination proof is less standard and makes use of a compactness property of disjunctive sequences of conjunctions of term disequalities (Lemma 13).

Theorem 13 *Let $\mathcal{S}$ and $\mathcal{N}$ denote the set of all derived and all derived normal clauses, respectively, which are constructed by exhaustively applying the normalization rules. Then for every predicate p,*

$$[\![p]\!]_{\mathcal{N}} = [\![p]\!]_{\mathcal{S}} = [\![p]\!]_{\mathcal{C}} \tag{3.1}$$

Proof. Since $\mathcal{N} \subseteq \mathcal{S}$ as well as $\mathcal{C} \subseteq \mathcal{S}$, $[\![p]\!]_{\mathcal{N}} \subseteq [\![p]\!]_{\mathcal{S}}$ and also $[\![p]\!]_{\mathcal{C}} \subseteq [\![p]\!]_{\mathcal{S}}$ for all predicates p. Since every clause c in $\mathcal{S}$ is implied by $\mathcal{C}$, the least model $\mathcal{M}_{\mathcal{S}}$ of $\mathcal{S}$ is still included in the least model of $\mathcal{C}$, i.e., $[\![p]\!]_{\mathcal{S}} = [\![p]\!]_{\mathcal{C}}$ for every predicate p. Now we claim that for every ground fact $p(t) \in \mathcal{M}_{\mathcal{S}}$ with a deduction of length m, there exists a deduction of $p(t)$ of length at most m using clauses from $\mathcal{N}$ only. In particular, this implies that $p(t) \in \mathcal{M}_{\mathcal{N}}$, and the statement (3.1) follows.

For a contradiction, assume that the latter claim is wrong. Let m be minimal such that there exists some ground fact $p(t) \in \mathcal{M}_{\mathcal{S}}$ with a deduction of length m where $p(t)$ cannot be deduced by normal clauses with at most m steps.

Let c denote the last clause which is applied in the deduction of $p(t)$. Then c itself cannot be contained in $\mathcal{N}$. We distinguish three cases.

1. If any of the literals on the right-hand side of c contains a constructor, there is a shorter deduction in $\mathcal{S}$ using a derived implication — implying that there is also a shorter deduction w.r.t. $\mathcal{N}$.

   Therefore, now assume that no literal on the right-hand side of c contains a constructor.

2. If c contains a literal $q(Y)$ where Y does not occur in the head, then again there is a shorter deduction with derived clauses.

   Since every disequality in c containing a variable which does not occur in the core clause, simply can be removed, we henceforth may assume that every variable of c also occurs in the head of c.

3. If the head does not contain a constructor, i.e., is of the form $p(X)$, then again a shorter deduction with normal clauses can be found.

But then c must be a normal clause, contradicting our assumption. □

Termination. It remains to prove that normalization eventually terminates. In order to achieve this, we prove that for each core clause $p(t_0) \Leftarrow q_1(t_1), \ldots, q_m(t_m)$ only finitely many distinct constraints ϕ must be distinguished. The key idea is not to add clauses which are implied by the current set of clauses.

Example 15 The set of normalized clauses which have been deduced in Example 14, contains the clauses:

$$\begin{array}{lll} q(f(X,Y)) & \Leftarrow & f(X,Y) \neq f(a,a) \\ q(f(X,Y)) & \Leftarrow & \end{array}$$

which agree in their core clauses.
As the first clause is subsumed by the second clause, it can be omitted. □

In general, we consider subsumption not just between two clauses, but between *all* clauses which agree in their core clauses. Note that the application of Resolution preserves the number of auxiliary variables (variables not occurring in the head), while Splitting strictly decreases that number and no auxiliary variables are involved in the Propagation rule. Hence the number of variables occurring in any newly generated clause is bounded. For the termination proof we now consider *families* of clauses which (semantically) only differ in their constraints, and conceptually replace them by single clauses whose constraints are disjunctions of conjunctions of disequalities. Two clauses $h \Leftarrow \alpha_1, \phi_1$ and $h \Leftarrow \alpha_2, \phi_2$ belong to the same family if α_1 and α_2 contain the same set of literals.

For the termination of $\mathcal{H}_1$-normalization it suffices not to add clauses that are already *subsumed* by the current set of clauses. A clause $h \Leftarrow \alpha_0, \phi_0$ is subsumed by a set of clauses $h \Leftarrow \alpha_i, \phi_i, i \geq 1$, if all clauses belong to the same family and ϕ_0 implies the disjunction $\bigvee_{i \geq 1} \phi_i$.

Example 16 The two clauses from Example 15 belong to the same family, and they are logically equivalent to the single clause:

$$q(f(X,Y)) \Leftarrow f(X,Y) \neq f(a,a) \vee \mathsf{true}$$

□

So for each family of clauses, the normalization steps yield a sequence

$$\phi_0 \vee \phi_1 \vee \phi_2 \vee \ldots$$

of conjunctions ϕ_i of disequalities which constitutes the *condition* under which at least one clause from that family is applicable (in the current set of implied clauses). In order to prove that these sequences become stable, we recall Theorem 3 from [MORS05] in a slightly simplified form:

Theorem 14 *Let $d_j, j \geq 0$, be a sequence of disjunctions of equalities over k variables. Then the sequence $\psi_j = \bigwedge_{i=0}^{j} d_i$ is ultimately stable, i.e., there is some $m \in \mathcal{N}$ such that for all $m' \geq m$, $\psi_m \Leftrightarrow \psi_{m'}$.*

Proof. For convenience, a proof of the theorem is included which essentially follows the proof given in [MORS05]. The first observation is that, if any of the ψ_j is unsatisfiable, i.e., equivalent to false, then all $\psi_{j'}$ for $j \leq j'$ also must be unsatisfiable, and the assertion of the theorem follows. Accordingly, let us henceforth assume that all ψ_j are satisfiable. Recall that a conjunction of term equalities $(s_1 = t_1) \wedge \ldots \wedge (s_r = t_r)$ is in *normal form* if

- (i) the left-hand sides s_i all are distinct variables,
- (ii) no variable occurring on the left-hand side may occur in a right-hand side t_i, and
- (iii) if both s_i and t_i are variables X_j and $X_{j'}$, then $j > j'$.

We note that every satisfiable conjunction of equalities is equivalent to a conjunction in normal form which is unique. Let us call a finite disjunction of such conjunctions a *normal DNF*.

For the sequence $\psi_j, j \geq 0$, we now successively construct a corresponding sequence Γ_j, $j \geq 0$, where each Γ_j is a normal DNF. The sequence is constructed in such a way that Γ_j is equivalent to ψ_j. First, consider an arbitrary disjunction $d_i = (s_1 = t_1) \vee \ldots \vee (s_r = t_r)$ where each $s_i = t_i$ is satisfiable. Then each such equality $s_i = t_i$ is equivalent to a conjunction c_i in normal form. Therefore, d_i is equivalent to the normal DNF $c_1 \vee \ldots \vee c_r$.

Now for $j = 0$, we choose Γ_0 as a normal DNF equivalent to d_0. For $j > 0$, assume that $c_1 \vee \ldots \vee c_r$ is a normal DNF for d_j. Then Γ_j is obtained from Γ_{j-1} and that normal DNF by constructing a normal DNF for $\Gamma_{j-1} \wedge (c_1 \vee \ldots \vee c_r)$. For each normal DNF Γ, we further maintain a vector $v[\Gamma] \in \mathbb{N}^k$ where the i-th component of $v[\Gamma]$ counts the number of conjunctions in Γ with exactly i equalities. On $\mathbb{N}^k$ we consider the lexicographical ordering "$\leq$" which is given by: $(n_1, \ldots, n_k) \leq (n'_1, \ldots, n'_k)$ iff either $n_l = n'_l$ for all l, or there is some $1 \leq i \leq k$ such that $n_l = n'_l$ for all $l < i$, and $n_i < n'_i$. Recall that this ordering is a *well-ordering*, i.e., it does not admit infinite strictly decreasing sequences.

Now assume that Γ_{j-1} equals $\bar{c}_1 \vee \ldots \vee \bar{c}_m$ for normal conjunctions $\bar{c}_i$. Then by distributivity, $\Gamma_{j-1} \wedge d_j$ is equivalent to $\bigvee_{i=1}^m \bar{c}_i \wedge (c_1 \vee \ldots \vee c_r)$. First, assume that for a given i, $\bar{c}_i \wedge c_l$ is equivalent to $\bar{c}_i$ for some l. Then also $\bar{c}_i \wedge (c_1 \vee \ldots \vee c_r)$ is equivalent to $\bar{c}_i$. Let V denote the subset of all i with this property. Thus for all $i \notin V$, $\bar{c}_i$ is *not* equivalent to any of the conjunctions $\bar{c}_i \wedge c_l$. Let $J[i]$ denote the set of all l such that $\bar{c}_i \wedge c_l$ is satisfiable. For every $l \in J[i]$, we can construct a non-empty normal conjunction $\bar{c}_{il}$ equivalent to $\bar{c}_i \wedge c_l$. Note that every variable occurring on the left-hand side of an equality in $\bar{c}_i$ also occurs on the left-hand side of an equality in $\bar{c}_{il}$, while at least one variable on the left-hand side of an equality in $\bar{c}_{il}$ does not occur on a left-hand side in $\bar{c}_i$. Therefore, $\bar{c}_{il}$ contains strictly more equalities than $\bar{c}_i$. Summarizing, we construct a normal DNF Γ_j equivalent to $\Gamma_{j-1} \wedge d_j$ as:

$$\left(\bigvee_{i \in V} \bar{c}_i \right) \vee \left(\bigvee_{i \notin V} \bigvee_{l \in J[i]} \bar{c}_{il} \right)$$

According to this construction, we always have $v[\Gamma_{j-1}] \geq v[\Gamma_j]$. Moreover, $v[\Gamma_{j-1}] = v[\Gamma_j]$ implies that $V = \{1, \ldots, m\}$ and therefore that Γ_{j-1} is equivalent to Γ_j. Therefore, if Γ_{j-1} is not equivalent to Γ_j, then $v[\Gamma_{j-1}] > v[\Gamma_j]$. Accordingly, if the sequence $\Gamma_j, j \geq 0$, is not ultimately stable, we obtain an infinite sequence of strictly decreasing vectors — contradiction. □

From that theorem, the following corollary is obtained.

Corollary 13 *For every sequence $c_i, i \geq 1$, of finite conjunctions of term disequalities, there exists some $m \geq 1$ such that $\bigvee_{i=1}^m c_i$ is implied by c_j for every $j \geq 1$.*

Proof. Consider the sequence $d_i, i \geq 1$, where $d_i \equiv \neg c_i$ is the *complement* of c_i, i.e., a disjunction of equalities. By Theorem 14, there exists some $m \geq 1$ such that $\bigwedge_{i=1}^m d_i$ implies d_j for all $j \geq 1$. Then for all $j \geq 1$, $\neg d_j \equiv c_j$ implies $\neg(\bigwedge_{i=1}^m d_i) \equiv \bigvee_{i=1}^m c_i$, and the assertion follows. □

Example 17

$$X \neq a \vee X \neq g(Y, b)$$

is equivalent to true and therefore implied by any conjunction of term disequalities. The disjunction

$$(X \neq a \wedge X \neq g(Y, b)) \vee Y \neq h(X)$$

is not yet a tautology and not implied by $X \neq b$. But the disjunction

$$(X \neq a \wedge X \neq g(Y, b)) \vee Y \neq h(X) \vee X \neq b$$

is equivalent to

$$(X \neq a \vee Y \neq h(X) \vee X \neq b) \ \wedge \ (X \neq g(Y, b) \vee Y \neq h(X) \vee X \neq b)$$

which again is a tautology and therefore implied by any further conjunction of disequalities. □

Now assume that $\mathcal{S}'$ is the set of clauses which is obtained if new clauses $p(t_0) \Leftarrow \alpha, \phi$ are added according to the normalization rules only if ϕ does *not* imply the disjunction $\bigvee\{\phi' \mid (p(t_0) \Leftarrow \alpha, \phi') \in \mathcal{S}'\}$. Note that the latter is decidable. Let $\mathcal{N}'$ denote the set of normal clauses contained in $\mathcal{S}'$. Then $\mathcal{S}' \subseteq \mathcal{S}$, and $\mathcal{N}' \subseteq \mathcal{N}$. Moreover, for every core clause $p(t_0) \Leftarrow \alpha$, the set $R = \{\phi' \mid (p(t_0) \Leftarrow \alpha, \phi') \in \mathcal{S}'\}$ is finite, and $\bigvee R$ is implied by every constraint ϕ for which $p(t_0) \Leftarrow \alpha, \phi$ is contained in $\mathcal{S}$. Therefore, $\mathcal{S}$ and $\mathcal{S}'$ as well as $\mathcal{N}$ and $\mathcal{N}'$ are equivalent.

By Lemma 13, $\mathcal{S}'$ has for each core clause c only finitely many clauses whose core is c. It remains to show that the number of possible cores is finite, in order to conclude that $\mathcal{S}'$ and hence also $\mathcal{N}'$ is finite. In order to enforce that, we assume that duplicates of literals in preconditions are removed. Then the following holds.

Theorem 15 *The number of* core *clauses occurring during normalization of finite sets of $\mathcal{H}_1$-clauses with term disequalities is finite.*

Proof. Since the number of predicates and constructors is finite, there are only finitely many distinct heads of clauses. The number of literals occurring in preconditions is bounded since new literals $p(t)$ are only added for *subterms* t of terms already present in the original set $\mathcal{C}$ of clauses. Therefore, the number of families of clauses occurring during normalization is finite. For each family f let $\psi_{\mathcal{C},f}$ denote the (possibly empty) disjunction of constraints of clauses of $\mathcal{C}$ which belong to f. Each clause that is added to $\mathcal{C}$ extends one of the finitely many constraints $\psi_{\mathcal{C},f}$ to $\psi_{\mathcal{C},f} \vee \phi$ for a conjunction of disequalities ϕ. The number of variables in each constraint $\psi_{\mathcal{C},f}$ is bounded since neither Resolution with normal clauses nor Splitting does introduce new variables, while Propagation may introduce fresh variables, but directly produces normal clauses. □

In particular, we have effectively constructed a finite set of normal clauses which is equivalent to $\mathcal{C}$, constituting a proof of Theorem 12. Summarizing, every finite set $\mathcal{C}$ of $\mathcal{H}_1$-clauses with term disequalities can be transformed to an equivalent finite set $\mathcal{N}$ of normal clauses with term disequalities. By Lemma 2, $\mathcal{N}$ can in turn be transformed to an equivalent finite set of automata clauses with term disequalities for which we have shown the availability of algorithms for the standard decision problems in Section 2.2.1. Overall we therefore have proved decidability of the extension of $\mathcal{H}_1$-clauses with term disequalities:

Theorem 16 *Satisfiability is decidable for a finite set of $\mathcal{H}_1$-clauses with term disequalities.* □

3.2 $\mathcal{H}_1$-Clauses with Path Disequalities

The concept of disequalities of terms shall now be generalized to disequalities of *subterms* — specified by *paths*. The generalization is strict for the extension of (finite sets of) automata clauses (see Section 3.2.1) — and consequently also for the extension of $\mathcal{H}_1$-clauses. In order to make the framework for $\mathcal{H}_1$-normalization work also in presence of path disequalities, a completely new construction for Splitting is required (sections 3.2.2 and 3.2.4). Also the argument for termination must be appropriately generalized (Section 3.2.3).

There are two versions for the specification of subterms by means of paths: labeled paths as introduced in Section 2.1 (page 14), which are primarily considered in this work, and *unlabeled* paths $i_1.i_2.\ldots.i_n$. For the extension of (finite sets of) automata clauses, though, both versions have the same expressive power when the alphabet Σ is fixed. Unlabeled paths are obtained from labeled paths $(f_1, i_1).\ldots.(f_n, i_n)$ by omitting the labels $f_1, \ldots, f_n$. The constructors occurring in the labeled paths can instead be recorded by means of specialized predicates. E.g., a clause

$$p(f(X_1, X_2)) \Leftarrow q(X_1), r(X_2),\ X_1.(f, 1) \neq X_2$$

is replaced by the clauses

$$\begin{array}{lcll} p_{f(\_,\_)}(f(X_1, X_2)) & \Leftarrow & q_{f(\_,\_)}(X_1), r_t(X_2), & X_1.1 \neq X_2 \\ p_{f(\_,\_)}(f(X_1, X_2)) & \Leftarrow & q_{g(\_,\ldots,\_)}(X_1), r_t(X_2) & \end{array}$$

for arbitrary patterns t and all $g \neq f$. In this construction, only those patterns are enumerated which are made up from suffixes of paths occurring in the given set of automata clauses. For the reverse direction, we observe that every unlabeled-path disequality is equivalent to a finite conjunction of labeled-path disequalities.

For convenience, we assume in the following that right-hand sides of path disequalities occurring in automata clauses may also be ground terms, since it can also be recorded by means of specialized predicates whether a constraint $X.\pi \neq t$ for a ground term t holds or not.

3.2.1 Increased Expressiveness

Tree automata with disequalities of subterms have been intensively investigated by Comon and Jacquemard [CJ94]. While decidability of emptiness is retained, allowing disequalities of subterms (instead of terms only) increases the expressiveness of the resulting class of extended tree automata. One indication for this is that universality is decidable [RS10] for tree automata with term disequalities

while it is undecidable for tree automata with path disequalities — since emptiness for automata with path *equalities* only [GGRÀ10] is undecidable [Mon81].

Let us define here a specific language T which can be expressed as $[\![p]\!]_{\mathcal{C}}$ for a finite set $\mathcal{C}$ of automata clauses with labeled-path disequalities, but which cannot be characterized by means of finite sets of automata clauses with term disequalities only. Let $T = [\![p]\!]_{\mathcal{C}}$ for the following set $\mathcal{C}$ of automata clauses:

$$\begin{array}{lcl} p(f(X_1, X_2)) & \Leftarrow & \top(X_1), \top(X_2), X_1 \neq X_2.(f, 1) \\ \top(f(X_1, X_2)) & \Leftarrow & \top(X_1), \top(X_2) \\ \top(a) & \Leftarrow & \end{array}$$

Lemma 14 *There is no finite set $\mathcal{C}'$ of automata clauses with term disequalities which define a predicate p such that $[\![p]\!]_{\mathcal{C}} = [\![p]\!]_{\mathcal{C}'}$.*

Proof. Assume for a contradiction that there is such a finite set $\mathcal{C}'$ which defines such a predicate p. Then we construct a finite set $\mathcal{N}$ of normal clauses for $\mathcal{C}'$ using equality term constraints only such that for every predicate q of $\mathcal{C}'$, $\mathcal{N}$ has a predicate $\bar{q}$ with $[\![\bar{q}]\!]_{\mathcal{N}}$ is the complement of $[\![q]\!]_{\mathcal{C}'}$. This set is constructed as follows. Assume that $q(f(X_1, \ldots, X_k)) \Leftarrow l_{i1}, \ldots, l_{ir_i}$ for $i = 1, \ldots, m$ are the clauses for q and constructor f where each l_{ij} either is a literal of the form $p'(X_s)$ or a single disequality. Then the predicate $\bar{q}$ for constructor f is defined by the set of all clauses

$$\bar{q}(f(X_1, \ldots, X_k)) \Leftarrow \bigwedge_{1 \leq i \leq m} \bar{l}_{ij_i}$$

where for each i, $1 \leq j_i \leq r_i$. For a literal l_{ij_i} of the form $p'(X_s)$, $\bar{l}_{ij_i}$ is given by $\bar{p}'(X_s)$, and if l_{ij_i} equals a disequality $t_1 \neq t_2$, $\bar{l}_{ij_i}$ is given by $t_1 = t_2$. By this construction the resulting clauses contain equality constraints only. As in Lemma 2, a finite set $\mathcal{A}$ of automata clauses with term equality constraints can be computed such that for all predicates $\bar{q}$ of $\mathcal{N}$, $[\![\bar{q}]\!]_{\mathcal{N}} = [\![\bar{q}]\!]_{\mathcal{A}}$.

A similar construction for automata without constraints has been described in [SN99]. Another variant has been presented in [GGRÀ10].

Let $\overline{T}$ denote the complement of T, i.e., the set $\mathcal{T}_\Sigma \setminus T$. $\overline{T}$ consists of all elements of the form

$$f(t, f(t, s))$$

for arbitrary terms s, t. By construction, $[\![\bar{p}]\!]_{\mathcal{A}} = \bar{T}$. Then $\mathcal{A}$ must contain a clause

$$\overline{p}(f(X_1, X_2)) \Leftarrow q_1(X_1), q_2(X_2), \phi$$

where ϕ is a finite conjunction of term equalities, such that there are distinct ground terms $t_1, t_2 \in [\![q_1]\!]_{\mathcal{A}}$ and for $i = 1, 2$ there are two distinct terms t_{i1}, t_{i2}

such that $f(t_i, t_{ij}) \in [\![q_2]\!]_{\mathcal{A}}$ and $f(t_i, f(t_i, t_{ij})) \in [\![\overline{p}]\!]_{\mathcal{A}}$ by application of this clause. This means for $\theta_{ij}(X_1) = t_i$ and $\theta_{ij}(X_2) = f(t_i, t_{ij})$, that $\theta_{ij} \models \phi$. In order to see this, we first convince ourselves that for every term t there must be some clause c_t by which for two distinct terms s_1, s_2, facts $\overline{p}(f(t, f(t, s_1)))$ and $\overline{p}(f(t, f(t, s_2)))$ can be derived. Assume for a contradiction, this were not the case. Then for some t and every clause c for predicate $\overline{p}$, there is at most one term t_c such that a fact $\overline{p}(f(t, f(t, t_c)))$ is derived by means of c. Accordingly, for this t, the set $\{s \mid f(t, f(t, s)) \in \bar{T}\}$ were finite — which is not the case. Consequently, for every t we can find a clause c_t by which for two distinct terms s_1, s_2, facts $\overline{p}(f(t, f(t, s_1)))$ and $\overline{p}(f(t, f(t, s_2)))$ can be derived. Since the number of terms is infinite while the number of clauses is finite, we conclude that there must be two distinct terms t_1, t_2 for which the clauses c_{t_1} and c_{t_2} coincide.

Now recall that each finite conjunction of term equalities $s_i = s_j$ between arbitrary terms s_i, s_j can be expressed as a finite conjunction of term equalities of the form $Z = s$ for variables Z. W.l.o.g. let ϕ be of this form. Furthermore, recall that a term equality $Z = s$ where s contains Z either is trivially true or trivially false. In addition to equalities $X_1 = X_1$ and $X_2 = X_2$, the satisfiable constraint ϕ therefore can only contain equalities of one of the following forms:

(1) $X_1 = g$ or $X_2 = g$, where g denotes a ground term;

(2) $X_1 = s$, where s contains variable X_2;

(3) $X_2 = s$, where s contains variable X_1

In the following, we show that an equality of any of these forms leads to a contradiction.

Case 1. Assume that there is an equality $X_r = g$ for some ground term g. If $r = 1$, then either $t_1 \neq g$ or $t_2 \neq g$ implying that $\theta_{ij} \not\models (X_r = g)$ for some i, j. If on the other hand, $r = 2$, then either $f(t_1, t_{11}) \neq g$ or $f(t_1, t_{12}) \neq g$, and hence also $\theta_{ij} \not\models (X_r = g)$ for some i, j.

Case 2. Assume that there is an equality $X_1 = s$ where s contains X_2. If $\theta_{11} \models (X_1 = s)$, then t_1 would contain the term $f(t_1, t_{11})$ — which is impossible.

Case 3. Finally, assume that there is an equality $X_2 = s$ where s contains X_1. Then consider the two substitutions θ_{11} and θ_{12}. If $\theta_{11} \models (X_2 = s)$, we conclude that $f(t_1, t_{11}) = \theta_{11}(X_2) = s[t_1/X_1]$. Then $\theta_{12} \models (X_2 = s)$ implies that $s[t_1/X_1]$ also equals $f(t_1, t_{12})$, and therefore, $f(t_1, t_{11}) = f(t_1, t_{12})$. This however, implies that $t_{11} = t_{12}$ — in contradiction to our assumption.

We conclude that the conjunction ϕ is equivalent to true. But then $[\![\overline{p}]\!]_{\mathcal{A}}$ must also contain the term $f(t_1, f(t_2, t_{21}))$ — which is not contained in $\bar{T}$. This completes the proof. □

3.2.2 Challenges in Computing with Subterms

In Section 3.1.2, the Splitting rule is applied based on a method that determines the number of terms accepted by a predicate w.r.t. the current subset of normal clauses. The idea of using this method is straight-forward. Things are, however, completely different if we are given a conjunction of *path* disequalities. The problem is that the number of terms is now irrelevant. Even the number of terms that are (pairwise) distinct at a specific path (i.e., terms $t_1, \ldots, t_m$ with $t_i.\pi \neq t_j.\pi$ for all $i < j$) is insufficient information for the goal of removing auxiliary variables.

Example 18 A small example that illustrates the difficulties is given by the following set $\mathcal{C}$ of $\mathcal{H}_1$-clauses with unlabeled-path disequalities.

$$\begin{array}{lcl} q_b(b) & \Leftarrow & \\ r(f(a,X)) & \Leftarrow & \\ r(f(b,b)) & \Leftarrow & \\ q(f(X,Y)) & \Leftarrow & r(X), r(Y),\ X.1 \neq Y.1 \\ p(h(X)) & \Leftarrow & q_b(X), q(Y),\ Y.1.2 \neq X, Y.2.2 \neq X \end{array}$$

Note that for the second and third clause, an equivalent finite set of automata clauses exists. For simplicity, we use the $\mathcal{H}_1$ form and variable names X, Y. The last clause is the one where Splitting for the variable Y should be applied w.r.t. the other clauses. There are infinitely many terms $t \in [\![q]\!]_{\mathcal{C}}$, and, moreover, the set of terms accepted by q even has infinitely many different subterms both at the path $\pi_1 = 1.2$ and at the path $\pi_2 = 2.2$ as q accepts all terms of the forms

$$f(f(a,t), f(b,b)) \quad \text{and} \quad f(f(b,b), f(a,t))$$

where t is arbitrary. Still p is empty in this example due to the constraint which enforces the disequality $Y.1.2 \neq b$ *in combination with* the disequality $Y.2.2 \neq b$. □

So what has to be considered is different *subterm combinations*. Essentially, the question is whether we can *cover* infinitely many possible substitutions $\theta(Y)$ for the auxiliary variable Y by only finitely many representing terms. What makes matters even more complicated is the fact that different subterms of the term in the head of a clause may originate from different predicates of the literals in the precondition. A straight-forward Splitting solution for paths therefore seems infeasible. Moreover, it must be dealt with the problem that each subterm combination may occur in a possibly infinite number of terms. The latter problem may be solved by avoiding endless recurrences of the same combinations. Thus, we have to exclude at paths π ground terms t which occur at π (i.e., $s.\pi = t$ for some term s that is available for the substitution of Y) — but then certain combinations

might be missed in the ongoing calculation, which is not acceptable. Therefore we additionally consider different orders of excluding such ground terms at different paths — a "blind exploration" technique which is able to account for *any* possible *combination* of subterms occurring in an infinite number of terms, and which is described in detail in Section 3.2.4.

3.2.3 Termination-sensitive Resolution in Presence of Paths

Another interesting problem arises when trying to adapt the normalization framework for constrained $\mathcal{H}_1$-clauses to the case of path disequalities. While a compactness property (Lemma 13) of disjunctive sequences of conjunctions of term disequalities can be exploited to guarantee termination of normalization in the case of term disequalities, the paths occurring in path disequalities may *grow* by the application of normalization steps. E.g., the Resolution rule leads to substitutions of variables by terms. Thus, a constraint

$$X_1 \neq X_2.\pi$$

may be transformed by a substitution $\theta = \{X_1 \mapsto g(X_1), X_2 \mapsto X_2, \ldots\}$ to

$$g(X_1) \neq X_2.\pi \quad \equiv \quad X_1 \neq X_2.\pi.(g,1)$$

resulting in a grown path $\pi.(g,1)$. Growing data, of course, must be avoided or at least bounded in saturation procedures. But it is unclear how to specify a maximal depth of paths in constraints resulting from resolution steps.

Therefore, we implement a different method. The idea is to reuse the compactness property of Lemma 13. To do so, the constraint language is *extended* from disequalities of path expressions $X.\pi$ to disequalities of terms containing such path expressions. Instead of growing paths this concept only results in growing terms over a finite set of possible path expressions (since paths may only shrink by normalization steps). Thus, in the above example, the constraint $g(X_1) \neq X_2.\pi$ is *not* transformed to the constraint $X_1 \neq X_2.\pi.(g,1)$ but kept. We call such terms possibly containing path expressions *general terms*. Note that disequalities of general terms subsume both disequalities of (ordinary) terms and disequalities of paths. Furthermore, general term disequalities allow to conveniently express that a specified subterm should be distinct from a given ground term. E.g., the clause

$$p(X) \Leftarrow X.(g,1) \neq a$$

defines a predicate p which accepts all terms of the forms $g(f(s_1,\ldots,s_k)), f \neq a$ and $h(t_1,\ldots,t_l), h \neq g$.

For a general term t, the subterm at path π, denoted $t.\pi$, is recursively defined by:

- $t.\epsilon = t$;
- $(Y.\pi_1).\pi_2 = Y.(\pi_1.\pi_2)$;
- $t.(f,i).\pi' = t_i.\pi'$ if $t = f(t_1,\ldots,t_m)$ and $1 \leq i \leq m$. In particular, for $g \neq f$, $t.(g,i).\pi'$ is *undefined*.

Thus, for a variable X, it depends on the respective substitution whether the expression $X.(g,i)$ will be defined or not.

Example 19 For an atom a and a binary constructor b, the following expressions t_i with

$$t_1 = a \qquad t_2 = b(b(a,a),X) \qquad t_3 = b(a,X.(b,1).(b,2))$$

all are general terms, and the following identities hold.

$$\begin{array}{lcllcl} t_2.(b,1).(b,1) & = & a & t_3.(b,2) & = & X.(b,1).(b,2) \\ t_2.(b,2).(b,2) & = & X.(b,2) & t_3.(b,2).(b,1) & = & X.(b,1).(b,2).(b,1) \end{array}$$

□

Consider now a general term t and a *ground* substitution θ. If $\theta(X).\pi$ is defined for each path expression $X.\pi$ occurring in t, then $t\theta$ is obtained from t by replacing each occurrence of $X.\pi$ with the ground term $\theta(X).\pi$. Otherwise, $t\theta$ is undefined.

Satisfiability of general term constraints. As with (dis)equalities of path expressions $X.\pi$, a refinement is needed for the definition of satisfiability w.r.t. a ground substitution θ, since for a general term t containing such path expressions, $\theta(t)$ may be undefined. Consider a general term disequality $t_1 \neq t_2$. As with path expressions, should either $\theta(t_1)$ or $\theta(t_2)$ be undefined, then $\theta \models (t_1.\pi_1 \neq t_2.\pi_2)$.

General term disequalities can be split into a disjunction of path disequalities.

Lemma 15 *For every disequality $t_1 \neq t_2$ between general terms t_1, t_2, a finite disjunction ϕ of path disequalities can be constructed such that $t_1 \neq t_2$ is equivalent to ϕ, i.e., for every substitution θ, it holds that $\theta \models (t_1 \neq t_2)$ iff $\theta \models \phi$.*

Proof. In the first step, we observe that the disequality $t_1 \neq t_2$ is equivalent to a finite disjunction of disequalities $X.\pi \neq t$ for suitable path expressions $X.\pi$ and subterms t occurring in t_1 or t_2. Now let Π denote the set of all labeled paths π' such that $t.\pi'$ either is a path expression or a ground term. Let Π_0 denote the minimal elements in Π, i.e., the set of all paths $\pi' \in \Pi$ where Π does not contain a proper prefix of π'. The elements in Π_0 denote labeled paths in t reaching maximal

ground subterms or path expressions contained in t. Therefore, the subset Π_0 is finite. Then the disequality $X.\pi \neq t$ is equivalent to the disjunction

$$\bigvee_{\pi' \in \Pi_0} X.\pi.\pi' \neq t.\pi'$$

□

Using this lemma, the following simulations can be proven.

Theorem 17 *Assume that $\mathcal{A}$ is a finite set of automata clauses with general term disequalities. Then the following holds:*

1. *A finite set $\mathcal{A}'$ of automata clauses with labeled-path disequalities can be effectively constructed such that for every predicate p, $[\![p]\!]_{\mathcal{A}} = [\![p]\!]_{\mathcal{A}'}$.*

2. *A finite set $\mathcal{A}''$ of automata clauses with unlabeled-path disequalities can be effectively constructed such that for all predicates p, $[\![p]\!]_{\mathcal{A}} = [\![p]\!]_{\mathcal{A}''}$.*

□

Since unlabeled-path disequalities can be expressed by means of labeled-path disequalities, and labeled-path disequalities are a special case of general term disequalities, all three classes of automata clauses compared in Theorem 17 are equally expressive. It furthermore follows that automata clauses with term disequalities can be simulated by means of automata clauses with labeled-path or unlabeled-path disequalities. In [CJ94], it has been proven that emptiness for finite tree automata with unlabeled-path disequalities is decidable. Therefore, emptiness is also decidable for automata clauses with general term disequalities:

Corollary 16 *Given a finite set $\mathcal{A}$ of automata clauses with general term disequalities and a predicate p, it is decidable whether or not $[\![p]\!]_{\mathcal{A}} = \emptyset$. Moreover, in case that $[\![p]\!]_{\mathcal{A}} \neq \emptyset$, a witness $t \in [\![p]\!]_{\mathcal{A}}$ can effectively be computed.* □

The following subsections and paragraphs provide the normalization rules for finite sets of $\mathcal{H}_1$-clauses with path disequalities. We refer to the current set of all implied clauses (whether originally present or added during normalization) as $\mathcal{C}$, while $\mathcal{N} \subseteq \mathcal{C}$ denotes the subset of normal clauses in $\mathcal{C}$.

Resolution. Recall that the Resolution rule simplifies a complicated clause from $\mathcal{C}$ by applying a resolution step with a normal clause. The principle approach is the same as with the Resolution rule for normalization of $\mathcal{H}_1$ with term disequalities. Assume that $\mathcal{C}$ contains a clause $h \Leftarrow \alpha_1, p(t), \alpha_2, \psi$.

If $\mathcal{N}$ contains a clause $p(X_1) \Leftarrow \phi$, then we add the clause $h \Leftarrow \alpha_1, \alpha_2, \psi, \psi'$ where $\psi' = \phi[t/X_1]$.

If $\mathcal{N}$ has a clause $p(f(X_1, \ldots, X_k)) \Leftarrow \beta, \phi$, and $t = f(t_1, \ldots, t_k)$, then we add the clause:

$$h \Leftarrow \alpha_1, \alpha', \alpha_2, \psi, \psi'$$

where $\alpha' = \beta[t_1/X_1, \ldots, t_k/X_k]$ and likewise, $\psi' = \phi[t_1/X_1, \ldots, t_k/X_k]$. These resolution steps may introduce new disequalities. The new constraints are obtained from already available constraints by substitution of terms for variables. We remark, though, that after simplification of queries $t.\pi$ with t not a variable, the new constraints only contain path expressions for paths which are *suffixes* of paths already occurring in constraints of $\mathcal{C}$.

Example 20 Consider the labeled-path variant of the voting protocol Example 4:

$$\begin{array}{lcl} valid(X_1) & \Leftarrow & q(X_1, [\,]) \\ q(X_1, X_2) & \Leftarrow & q(X_1, :: (Y, X_2)),\ X_1.(vote, 1) \neq Y.(vote, 1) \\ q(X_1, X_2) & \Leftarrow & is\_vote(X_1), votes(X_2) \end{array}$$

enhanced with the normal clauses:

$$\begin{array}{rcl} empty([\,]) & \Leftarrow & \\ age_{15}(15) & \Leftarrow & \\ age_{25}(25) & \Leftarrow & \\ n_{alice}(alice) & \Leftarrow & \\ n_{bob}(bob) & \Leftarrow & \end{array} \qquad \begin{array}{rcl} h_b(pers(X_1, X_2)) & \Leftarrow & n_{bob}(X_1), age_{25}(X_2) \\ h_a(pers(X_1, X_2)) & \Leftarrow & n_{alice}(X_1), age_{15}(X_2) \\ p_{bob}(vote(X_1, X_2)) & \Leftarrow & h_b(X_1) \\ p_{alice}(vote(X_1, X_2)) & \Leftarrow & h_a(X_1) \\ votes(:: (X_1, X_2)) & \Leftarrow & p_{bob}(X_1), v'(X_2) \\ v'(:: (X_1, X_2)) & \Leftarrow & p_{alice}(X_1), empty(X_2) \end{array}$$

to fill the list of already submitted votes with the two entries $vote(pers(alice, 15), \_)$ and $vote(pers(bob, 25), \_)$ where _ is intended to represent one of the atoms yes or no ($yes, no \in \Sigma$). Resolving the first two clauses of this example with the third one for the substitution $X_2 \mapsto [\,]$ and $X_2 \mapsto :: (Y, X_2)$, respectively, yields the clauses:

$$\begin{array}{lcl} valid(X_1) & \Leftarrow & is\_vote(X_1), votes([\,]) \\ q(X_1, X_2) & \Leftarrow & is\_vote(X_1), votes(:: (Y, X_2)), X_1.(vote, 1) \neq Y.(vote, 1) \end{array}$$

With the clause $votes(:: (X_1, X_2)) \Leftarrow p_{bob}(X_1), v'(X_2)$, the second new clause can further be resolved to obtain

$$q(X_1, X_2) \Leftarrow is\_vote(X_1), p_{bob}(Y), v'(X_2), X_1.(vote, 1) \neq Y.(vote, 1)$$

3.2.4 Splitting Paths

Recall that the Splitting rule removes variables not contained in the head of a clause. Assume that $\mathcal{C}$ contains a simple clause $h \Leftarrow \alpha, \psi$ and Y is a variable occurring in α, ψ but not in h. We rearrange the precondition α into a sequence $\alpha', q_1(Y), \ldots, q_r(Y)$ where α' does not contain the variable Y. Then we construct a finite sequence $t_1, \ldots, t_l$ of ground terms such that, w.r.t. $\mathcal{N}$,

$$\psi[t_1/Y] \vee \ldots \vee \psi[t_l/Y]$$

is equivalent to $\exists\, Y\, q_1(Y), \ldots, q_r(Y), \psi$, and add the clauses

$$h \Leftarrow \alpha', \psi[t_j/Y], \quad j = 1, \ldots, l$$

to the set $\mathcal{C}$.

According to this construction, Splitting may introduce new disequalities. As in the case of Resolution, new constraints are obtained from already available constraints by substitution of (ground) terms. This means that, after simplification of queries $t.\pi$ with t not a variable, the new constraints only contain path expressions for paths which are *suffixes* of paths already occurring in constraints of $\mathcal{C}$.

The remainder of this section provides a proof that the finite sequence $t_1, \ldots, t_l$ of ground terms for the removal of Y exists together with an effective construction of such a sequence. For notational convenience, let us assume that instead of a finite sequence $q_1, \ldots, q_r$ of predicates we are just given a single predicate p which is defined by means of a finite set of automata clauses $\mathcal{A}$. By Lemma 2, the results can then be applied to the case of a finite conjunction $q_1(Y) \wedge \ldots \wedge q_r(Y)$ (and intersection $[\![q_1]\!]_{\mathcal{N}} \cap \ldots \cap [\![q_r]\!]_{\mathcal{N}}$, respectively) which is defined by an equivalent finite set $\mathcal{N}$ of normal clauses.

Theorem 18 *Let $\mathcal{A}$ be a finite set of automata clauses with general term disequalities, p a predicate, and Y a variable. For every conjunction of labeled-path disequalities ψ, a finite sequence of ground terms $t_1, \ldots, t_l$ can be effectively constructed such that, with respect to $\mathcal{A}$, the disjunction $\phi = \psi[t_1/Y] \vee \ldots \vee \psi[t_l/Y]$ is logically equivalent to the expression $\exists\, Y\, p(Y), \psi$.*

Proof. W.l.o.g. we assume that the variable Y does not occur on both sides within the same disequality in ψ. Otherwise, we modify the set $\mathcal{A}$ of clauses in such a way that only those terms of the (original) set $[\![p]\!]_{\mathcal{A}}$ are accepted by the predicate p which satisfy those disequalities. (For this construction, we may need to introduce an auxiliary predicate in order to take care of production cycles, since terms of $[\![p]\!]_{\mathcal{A}}$ may also contain subterms that are accepted by p.)

Now let Π denote the set of path expressions $Y.\pi$ occurring in ψ, and m the total number of occurrences of such expressions in ψ. We construct a finite sequence of terms $t_1, \ldots, t_l$ of $[\![p]\!]_{\mathcal{A}}$ such that for each ground substitution θ not mentioning Y, the following holds: if $\theta \oplus \{Y \mapsto t\} \models \psi$ for some $t \in [\![p]\!]_{\mathcal{A}}$, then also $\theta \oplus \{Y \mapsto t_i\} \models \psi$ for some i. Each of the terms t_i is generated during one possible run of the following nondeterministic algorithm. The algorithm starts with one term $s_0 \in [\![p]\!]_{\mathcal{A}}$. If no such term exists, the empty sequence is returned. Otherwise, the algorithm adds s_0 to the output sequence and proceeds according to one permutation of the occurrences of path expressions $Y.\pi$ occurring in ψ. Then it iterates of the path expressions in the permutation. In the round i for the path expression $Y.\pi$, the current set $\mathcal{A}$ of automata clauses is modified in such a way that all terms t with $t.\pi = s_{i-1}.\pi$ are excluded from $[\![p]\!]_{\mathcal{A}}$. Let $\mathcal{A}'$ be the resulting set of automata clauses. If $[\![p]\!]_{\mathcal{A}'}$ is empty, the algorithm terminates. Otherwise, a term $s_i \in [\![p]\!]_{\mathcal{A}'}$ is selected and added to the output sequence.

For the correctness of the approach, consider an arbitrary ground substitution θ defined for all variables occurring in ψ with the exception of Y. First, assume that $\exists\, Y\, p(Y), \psi\theta$ is not satisfiable. Then for no $s \in [\![p]\!]_{\mathcal{A}}$, $\psi\theta[s/Y]$ is true. Hence, also the finite disjunction provided by our construction cannot be satisfiable for such a θ, and therefore is equivalent to $\exists\, Y\, p(Y), \psi\theta$. Now assume that $\theta \models \exists\, Y\, p(Y), \psi$, i.e., some $s \in [\![p]\!]_{\mathcal{A}}$ exists with $\theta \models \psi[s/Y]$. Then we claim that there exists some s' occurring during one run of the nondeterministic algorithm with $\theta \models \psi[s'/Y]$. We construct this run as follows. Let $s_0 \in [\![p]\!]_{\mathcal{A}}$ denote the start term of the algorithm. If s_0 satisfies all disequalities, we are done. Otherwise, we choose one disequality $Y.\pi \neq t$ in $\psi\theta$ which is not satisfied. This means that $s_0.\pi = t$. Accordingly, we choose $Y.\pi$ as the first occurrence of a path expression selected by the algorithm. In particular, this means that all further terms s_i output by the algorithm will satisfy the disequality $Y.\pi \neq t$. After each round, one further disequality is guaranteed to be satisfied – while still s is guaranteed to be accepted by p in the resulting set $\mathcal{A}'$ of automata clauses. □

Corollary 17 *Let $\mathcal{N}$ be a finite set of normal clauses with general term disequalities, $q_1, \ldots, q_r$ a finite sequence of predicates, and Y a variable. For every conjunction of general term disequalities ψ, a finite sequence of ground terms $s_1, \ldots, s_l$ can be constructed such that with respect to $\mathcal{N}$, the disjunction $\phi = \psi[s_1/Y] \vee \ldots \vee \psi[s_l/Y]$ is equivalent to the expression $\exists\, Y\, (q_1(Y), \ldots, q_r(Y), \psi)$.*

Proof. First, recall that we can construct a finite set $\mathcal{A}$ of automata clauses together with a predicate p such that $[\![p]\!]_{\mathcal{A}} = [\![q_1]\!]_{\mathcal{N}} \cap \ldots \cap [\![q_r]\!]_{\mathcal{N}}$. Clearly, $\exists\, Y\, p(Y), \psi$ is implied by every constraint $\psi[s/Y]$ with $s \in [\![p]\!]_{\mathcal{A}}$.

By Lemma 15, every disequality $t_1 \neq t_2$ is equivalent to a disjunction of disequalities of the form $X.\pi \neq Z.\pi'$ or $X.\pi \neq t$ for variables X, Z, paths π, π',

and ground terms t. Accordingly, ψ is equivalent to a disjunction $\psi_1 \vee \ldots \vee \psi_k$ for suitable conjunctions ψ_i of labeled-path disequalities. By Theorem 18, each conjunction $p(Y), \psi_i$ is equivalent to a disjunction $\psi_i[s_{i1}/Y] \vee \ldots \vee \psi_i[s_{il_i}/Y]$. Since $p(Y), \psi$ is equivalent to the disjunction

$$p(Y), \psi_1 \vee \ldots \vee p(Y), \psi_k$$

we conclude that it is equivalent to the disjunction:

$$\psi_1[s_{11}/Y] \vee \ldots \vee \psi_k[s_{kl_k}/Y]$$

The latter, on the other hand, implies the disjunction

$$\psi[s_{11}/Y] \vee \ldots \vee \psi[s_{kl_k}/Y]$$

Therefore, the sequence $s_{11}, \ldots, s_{kl_k}$ satisfies the requirements of the corollary. □

Example 21 The resolution steps of Example 20 produced the clause

$$q(X_1, X_2) \Leftarrow \mathit{is\_vote}(X_1), p_{\mathit{bob}}(Y), v'(X_2), X_1.(\mathit{vote}, 1) \neq Y.(\mathit{vote}, 1)$$

In order to decide satisfiability of $\exists Y p_{\mathit{bob}}(Y), X_1.(\mathit{vote}, 1) \neq Y.(\mathit{vote}, 1)$, the algorithm of Theorem 18 starts with the term $\mathit{vote}(\mathit{pers}(\mathit{bob}, 25), \mathit{yes}) \in [\![p_{\mathit{bob}}]\!]_{\mathcal{N}}$ for the subset $\mathcal{N}$ of normal clauses. Then it enforces the disequality $s.(\mathit{vote}, 1) \neq \mathit{pers}(\mathit{bob}, 25)$ for terms $s \in [\![p_{\mathit{bob}}]\!]_{\mathcal{N}}$ and finds out by a successful emptiness-test that no such term exists. Therefore only the *normal* clause:

$$q(X_1, X_2) \Leftarrow \mathit{is\_vote}(X_1), v'(X_2), X_1.(\mathit{vote}, 1) \neq \mathit{pers}(\mathit{bob}, 25)$$

is added. This new clause in turn enables two more resolution steps for the first two clauses of this voting protocol example, yielding

$$\begin{array}{lcll} \mathit{valid}(X_1) & \Leftarrow & \mathit{is\_vote}(X_1), v'([\,]), & X_1.(\mathit{vote}, 1) \neq \mathit{pers}(\mathit{bob}, 25) \\ q(X_1, X_2) & \Leftarrow & \mathit{is\_vote}(X_1), v'(:: (Y, X_2)), & X_1.(\mathit{vote}, 1) \neq \mathit{pers}(\mathit{bob}, 25), \\ & & & X_1.(\mathit{vote}, 1) \neq Y.(\mathit{vote}, 1) \end{array}$$

for the substitution $X_2 \mapsto [\,]$ and $X_2 \mapsto :: (Y, X_2)$, respectively. Another resolution step with the clause $v'(:: (X_1, X_2)) \Leftarrow p_{\mathit{alice}}(X_1), \mathit{empty}(X_2)$ now yields:

$$\begin{array}{ll} q(X_1, X_2) \Leftarrow \mathit{is\_vote}(X_1), p_{\mathit{alice}}(Y), \mathit{empty}(X_2), & X_1.(\mathit{vote}, 1) \neq \mathit{pers}(\mathit{bob}, 25), \\ & X_1.(\mathit{vote}, 1) \neq Y.(\mathit{vote}, 1) \end{array}$$

with substitution $X_1 \mapsto Y$. To the last clause, the Splitting rule can again be applied in order to replace the precondition $p_{\mathit{alice}}(Y), X_1.(\mathit{vote}, 1) \neq Y.(\mathit{vote}, 1)$

with a disjunction of constraints not containing Y. As is the case with p_{bob}, the algorithm of Theorem 18 finds one term for predicate p_{alice}, for instance $vote(pers(alice, 15), no)$. Then for the path $(vote, 1)$ the term $pers(alice, 15)$ is excluded, and p_{alice} becomes empty. Thus, the new normal clause

$$q(X_1, X_2) \Leftarrow is\_vote(X_1), empty(X_2), \quad X_1.(vote, 1) \neq pers(bob, 25), \; X_1.(vote, 1) \neq pers(alice, 15)$$

is added. Again the obtained normal clause enables two resolution steps for the first two clauses of the voting protocol, yielding

$$\begin{aligned} valid(X_1) &\Leftarrow is\_vote(X_1), empty([\,]), \phi \\ q(X_1, X_2) &\Leftarrow is\_vote(X_1), empty(:: (Y, X_2)), X_1.(vote, 1) \neq Y.(vote, 1), \phi \end{aligned}$$

where ϕ abbreviates the conjunction $X_1.(vote, 1) \neq pers(bob, 25), X_1.(vote, 1) \neq pers(alice, 15)$. Finally, a last resolution step with the clause $empty([\,]) \Leftarrow$ for the first of these two clauses achieves the result

$$valid(X_1) \Leftarrow is\_vote(X_1), \phi$$

where ϕ again equals $X_1.(vote, 1) \neq pers(bob, 25), X_1.(vote, 1) \neq pers(alice, 15)$, stating that a vote is valid if it is submitted by a person who is different from the two persons as stored in the given list. □

In the example, $\mathcal{N}$ grows, but the set $[\![p_{bob}]\!]_{\mathcal{N}}$ is not affected. Therefore, the constraint $X_1.(vote, 1) \neq Y.(vote, 1)$ cannot be completely removed but only translated to $X_1.(vote, 1) \neq pers(bob, 25)$. However, if later $\mathcal{N}$ changed in such a way that $[\![p_{bob}]\!]_{\mathcal{N}}$ contains two or more elements, the constraint would be removed by a splitting step.

Propagation. The Propagation rule is concerned with clauses that are of the form $p(X_1) \Leftarrow q_1(X_1), \ldots, q_r(X_1), \psi$ where ψ only contains the variable X_1 (or none). The principal procedure is the same as with Propagation in case of normalization of $\mathcal{H}_1$-clauses with term disequalities as described in Section 3.1. Assume that $r > 0$, and $\mathcal{N}$ contains normal clauses $q_j(f(X_1, \ldots, X_k)) \Leftarrow \alpha_j, \psi_j$ for $j = 1, \ldots, r$. Then we add the normal clause:

$$p(f(X_1, \ldots, X_k)) \Leftarrow \alpha_1, \ldots, \alpha_r, \psi_1, \ldots, \psi_r, \psi'$$

where $\psi' = \psi[f(X_1, \ldots, X_k)/X_1]$.

This rule may create new disequalities, too. Again, however, after simplification of queries $t.\pi$ where t is not a variable, the new constraints only contain path expressions for paths which are suffixes of paths already occurring in disequalities of the original set $\mathcal{C}$.

Example 22 Consider the following variant of the voting protocol example

$$\begin{array}{lcl} person(pers(X_1, X_2)) & \Leftarrow & name(X_1), age(X_2) \\ adult(X_1) & \Leftarrow & person(X_1),\ \bigwedge_{0\leq i\leq 17} X_1.(pers, 2) \neq i \\ valid(vote(X_1, Y)) & \Leftarrow & q(X_1, [\,]) \\ q(X_1, X_2) & \Leftarrow & q(X_1, :: (Y, X_2)), X_1 \neq Y.(vote, 1) \\ q(X_1, X_2) & \Leftarrow & adult(X_1), votes(X_2) \end{array}$$

intending to only allow adults to submit a vote. Here, the first argument of q is a person instead of a vote. Then the second clause is instantiated for constructor $pers$ with substitution $X_1 \mapsto pers(X_1, X_2)$. Together with the first clause, we obtain:

$$adult(pers(X_1, X_2)) \Leftarrow name(X_1), age(X_2), \bigwedge_{0\leq i\leq 17} pers(X_1, X_2).(pers, 2) \neq i$$

or simplified: $adult(pers(X_1, X_2)) \Leftarrow name(X_1), age(X_2), \bigwedge_{0\leq i\leq 17} X_2 \neq i$ □

Theorem 19 *Let $\mathcal{C}$ denote a finite set of $\mathcal{H}_1$-clauses with path disequalities. Let $\overline{\mathcal{C}}$ denote the set of all clauses obtained from $\mathcal{C}$ by exhaustively adding clauses according to the normalization rules. Then the subset $\mathcal{N}$ of all normal clauses in $\overline{\mathcal{C}}$ is equivalent to $\mathcal{C}$, i.e., $[\![p]\!]_{\mathcal{C}} = [\![p]\!]_{\mathcal{N}}$ for every predicate p occurring in $\mathcal{C}$.*

Proof. The proof follows the same lines as the proof of Theorem 13: every clause added to $\mathcal{C}$ by means of Resolution, Splitting or Propagation is implied by the set of clauses in $\mathcal{C}$. Therefore, for every predicate p, $[\![p]\!]_{\mathcal{C}} = [\![p]\!]_{\overline{\mathcal{C}}}$. In the second step, it is verified that every fact $p(t)$ which can be deduced by means of the clauses in $\overline{\mathcal{C}}$ can also be deduced in at most as many steps with clauses from $\mathcal{N}$ alone. □

The proof of termination of $\mathcal{H}_1$-normalization in presence of path disequalities is more specific and makes use of the fact that the paths occurring in general term constraints do not grow by resolution steps. Note that the proposition of Theorem 15 that only finitely many core clauses occur still holds, as the resolution steps are done with normal clauses, resulting only in literals $p(t)$ for subterms t of terms that are already present. In order to prove termination, however, we additionally must show that the occurring disjunctive sequences of finite conjunctions of disequalities also become stable if *general* terms are used instead of ordinary terms.

Theorem 20 *Let $\mathcal{C}$ denote a finite set of $\mathcal{H}_1$-clauses with path disequalities. Let $\overline{\mathcal{C}}$ denote the set of clauses obtained from $\mathcal{C}$ by adding all clauses according to the Resolution, Splitting and Propagation rules that are not subsumed by the current set of clauses. Then $\overline{\mathcal{C}}$ is finite.*

Proof. The difference w.r.t. the corresponding proof for term disequalities concerns the argumentation that only finitely many constraints can occur. Concerning the core clauses, we recall that the number of cores is bounded, i.e., only finitely many distinct *families* of clauses can occur during normalization. For each family f, let $\psi_{\mathcal{C},f}$ denote the disjunction of constraints of clauses of $\mathcal{C}$ which belong to f. Each clause that is added to $\mathcal{C}$ extends one of the finitely many constraints $\psi_{\mathcal{C},f}$ to $\psi_{\mathcal{C},f} \vee \phi$ for a conjunction of disequalities ϕ. In order to show that the resulting disjunctions eventually are implied, we recall that in every normalization step, the terms in constraints may grow — but the lengths of paths in path expressions remain bounded. Therefore, the number of all possibly occurring path expressions is finite.

By Theorem 14, for each sequence ψ_i of conjunctions of term disequalities over finitely many variables, the disjunction $\bigvee_{i=1}^{m} \psi_i$, $m \geq 1$, eventually becomes *stable*, i.e., there exists some M such that $\bigvee_{i=1}^{m} \psi_i = \bigvee_{i=1}^{M} \psi_i$ for all $m \geq M$. This also holds true if we use finitely many path expressions instead of variables. Therefore a disjunction $\bigvee_{i=1}^{m} \psi_i, m \geq 1$, also eventually becomes stable if each ψ_i is a conjunction of general term disequalities and the set of occurring path expressions is finite.

We conclude that eventually, every newly added clause is subsumed — implying that the modified normalization procedure terminates with a finite set of clauses $\overline{\mathcal{C}}$. □

By Theorem 19 and Theorem 20, the modified normalization rules for path disequalities constitute a sound and complete procedure which constructs for every finite set $\mathcal{C}$ of $\mathcal{H}_1$-clauses with path disequalities an equivalent finite set $\mathcal{N}$ of normal clauses within finitely many steps. By Lemma 2, $\mathcal{N}$ can then be transformed to an equivalent finite set $\mathcal{A}$ of automata clauses with path disequalities, for which emptiness can be decided for every predicate according to [CJ94]. Altogether this proves our main result of this section:

Theorem 21 *Assume that $\mathcal{C}$ is a finite set of $\mathcal{H}_1$-clauses with general term disequalities. Then a finite set $\mathcal{A}$ of automata clauses can be effectively constructed such that for every predicate p of $\mathcal{C}$, $[\![p]\!]_{\mathcal{C}} = [\![p]\!]_{\mathcal{A}}$. In particular, it is decidable whether or not $[\![p]\!]_{\mathcal{C}}$ is empty.* □

3.3 $\mathcal{H}_1$-Clauses with Hom-Disequalities

Here, we extend the concept of term disequalities by transforming the compared terms by a tree homomorphism H prior to the test for disequality. The disequality $s \neq_{\mathrm{H}} t$, as defined in Section 2.1 on page 16, holds true if the images of s and t under H are distinct. In case of $s = t$, it also holds that $s =_{\mathrm{H}} t$ – and therefore the hom-disequality $s \neq_{\mathrm{H}} t$ is false. If, on the other hand, s and t are distinct terms, $s \neq_{\mathrm{H}} t$ may still be false. This may not only be the case for terms containing distinct variables such as $f(X_1)$ and $f(X_2)$, but even for terms whose topmost symbols are different.

Example 23 Let $\mathrm{H} = \{a \mapsto b, g \mapsto f(b, X_1), b \mapsto b, f \mapsto f(X_1, X_2)\}$. Then the hom-disequality

$$g(a) \neq_{\mathrm{H}} f(b, Y)$$

is not satisfied by the substitution $\theta = \{Y \mapsto b\}$ since both $\mathrm{H}(\theta(f(b, Y))) = \mathrm{H}(f(b, b)) = f(b, b)$ and also $\mathrm{H}(\theta(g(a))) = \mathrm{H}(g(a)) = f(b, \mathrm{H}a) = f(b, b)$ yield the same term. □

At the end of Section 2.1 on page 23, we have seen that in contrast to ordinary term disequalities, hom-disequalities cannot easily be split in disjunctions of the form $X \neq_{\mathrm{H}} t$ where X is a variable and t is a term which does not contain X. For the Splitting rule, however, a form where auxiliary variables Y do not occur on both the left- and right-hand side of a disequality, is desirable.

Example 24 Consider the constraint

$$f(X_1, Y) \neq_{\mathrm{H}} g(X_2, Y)$$

In case that $\mathrm{H}g = \mathrm{H}f = g(X_1, X_2)$, this constraint is equivalent to the disequality $X_1 \neq_{\mathrm{H}} X_2$ – and thus *independent* of the substitutions for Y. If, on the other hand, $\mathrm{H}g = g(X_1, X_2)$ but $\mathrm{H}f = g(X_2, X_2)$, then it is equivalent to the disequality $Y \neq_{\mathrm{H}} X_2$ and therefore depends on the substitutions for Y. □

In order to simplify the hom-disequality constraint $\phi = \phi_1 \wedge \ldots \wedge \phi_n$ of a clause $h \Leftarrow \alpha, \phi$, we proceed as follows. In the first step, we apply H to each disequality $\phi_i = (s_i \neq_{\mathrm{H}} t_i)$. In the second step, the topmost symbols of the resulting terms $\mathrm{H}s_i$ are *matched* with those of the terms $\mathrm{H}t_i$, as in the case of ordinary term disequalities. If there is a mismatch, ϕ_i is vacuously true and therefore is removed from the conjunction ϕ. Otherwise, this step results in a disjunction of disequalities of the form $X_{ij} \neq t_{ij}$ where X_{ij} is a variable. Then we undo the effect of applying H to both sides of the disequality by computing arbitrary pre-images of the t_{ij} under H. The result then is a disjunction of hom-disequalities

$$d_i = X_{i1} \neq_{\mathrm{H}} \mathrm{H}^{-1}(t_{i1}) \vee \ldots \vee X_{im} \neq_{\mathrm{H}} \mathrm{H}^{-1}(t_{im})$$

which is equivalent to ϕ_i. Therefore, the conjunction $\phi' = d_1 \wedge \ldots \wedge d_n$ is equivalent to the given constraint ϕ. We then transform ϕ' to DNF $\psi_1 \vee \ldots \vee \psi_l$ and replace the clause $h \Leftarrow \alpha, \phi$ by clauses $h \Leftarrow \alpha, \psi_i$ for $i = 1, \ldots, l$.

Outline. We show in Section 3.3.1 that for finite sets $\mathcal{A}$ of automata clauses with hom-disequalities, it can be efficiently decided for every predicate p whether at least k terms are contained in the homomorphic image of $[\![p]\!]_{\mathcal{A}}$. Moreover, if only $m < k$ such terms are contained, a witness of terms $t_1, \ldots, t_m \in [\![p]\!]_{\mathcal{A}}$ can be computed whose homomorphic images are pairwise distinct. This construction is used in Section 3.3.3 for the Splitting rule.

In Section 3.3.2, we then show that finite sets of automata clauses with hom-disequalities are more expressive than $\mathcal{H}_1$ with term disequalities, and that they are incomparable to finite sets of automata clauses with path disequalities.

For the proof that normalization terminates also for hom-disequalities, we have to take care of syntactically distinct disequality constraints which are semantically equivalent. E.g., the two atomic constraints $X \neq_{\mathrm{H}} a$ and $X \neq_{\mathrm{H}} g(a)$ may express the same if H deletes the constructor g. In order to account for such cases, we replace, for the decision of subsumption, the topmost constructors of the left-hand sides and right-hand sides of disequalities according to the given tree homomorphism, as described in Section 3.3.3 on page 75.

3.3.1 Tree Automata with Hom-Disequalities

In this section we first show that to every finite set of automata clauses $\mathcal{A}$ with hom-disequalities $s \neq_{\mathrm{H}} t$, a finite set of *generalized* automata clauses with term disequalities $\mathcal{A}_{\mathrm{H}}$ can be effectively constructed such that $\mathrm{H}[\![p]\!]_{\mathcal{A}} = [\![p]\!]_{\mathcal{A}_{\mathrm{H}}}$. *Generalized* here means that the heads may contain complex terms and the preconditions may contain auxiliary variables. Secondly, we show that it is decidable for $\mathcal{A}_{\mathrm{H}}$, every predicate p, and and number k whether $|[\![p]\!]_{\mathcal{A}_{\mathrm{H}}}| < k$ — and thus, whether $|\mathrm{H}[\![p]\!]_{\mathcal{A}}| < k$.

The class HDA(Hom-Disequality-Automata) of general automata with term disequalities consists of finite sets of clauses of the form:

$$p(t) \Leftarrow p_1(X_1), \ldots, p_k(X_k), \phi \qquad (k \geq 0)$$

where $p, p_1, \ldots, p_k$ are unary predicates, t is a term with $\mathrm{vars}(t) \subseteq \{X_1, \ldots, X_k\}$, and ϕ is a conjunction of disequalities $t_i \neq t_j$ which may only mention variables from $\{X_1, \ldots, X_k\}$. The set $\mathrm{vars}(t)$ of variables occurring in the head of such a clause c is denoted $\mathrm{hv}(c)$ while its complement with respect to $\{X_1, \ldots, X_k\}$, i.e., the set of (auxiliary) variables occurring only in the body of c, is denoted $\mathrm{bv}(c)$. The number of disequalities in a clause c is denoted $\mathrm{dc}(c)$.

Note that in this definition of HDA auxiliary variables are named X_i for some $i \geq 1$ — instead of Y, Y_i. Also for each auxiliary variable X_i there is some literal $p(X_i)$ in the precondition of the clause. However, this is not a restriction on the preconditions of clauses since the predicate p may be chosen as $\top$ so that only the occurrences of X_i within ϕ are *relevant*. The following lemma provides the construction which transforms a finite set of automata clauses with disequalities modulo a given tree homomorphism H to an HDA. The construction also applies for equalities modulo H — which is used for an undecidability result in Section 2.4.

Lemma 18 *Let $\mathcal{A}$ be a finite set of automata clauses with hom-equalities $t_1 =_\mathrm{H} t_2$ and hom-disequalities $t_1 \neq_\mathrm{H} t_2$ for an arbitrary tree homomorphism H. Then an HDA $\mathcal{A}_\mathrm{H}$ with additional term equalities in the preconditions can be constructed such that for every predicate p, $[\![p]\!]_{\mathcal{A}_\mathrm{H}} = \mathrm{H}[\![p]\!]_\mathcal{A}$. If $\mathcal{A}$ only contains (dis)equalities, then $\mathcal{A}_\mathrm{H}$ only contains (dis)equalities as well.*

Proof. Note that an analogous construction has been provided by Godoy et al. for tree automata with *path disequalities* [GGRÀ10]. For simplicity, we provide the construction for automata clauses with hom-*dis*equalities only. If hom-equalities occur, too, the construction yields additional term equalities in the resulting generalized automaton $\mathcal{A}_\mathrm{H}$. $\mathcal{A}_\mathrm{H}$ is obtained from $\mathcal{A}$ essentially by applying H to each clause, i.e., by transforming each clause c of the form:

$$p(f(X_1, \ldots, X_k)) \Leftarrow p_1(X_1), \ldots, p_k(X_k), \phi$$

with $\phi = l_1 \neq_\mathrm{H} r_1 \wedge \ldots \wedge l_m \neq_\mathrm{H} r_m$, $m \geq 0$ to the new clause c':

$$p(\mathrm{H}^*(f(X_1, \ldots, X_k))) \Leftarrow p_1(X_1), \ldots, p_k(X_k), \phi'$$

with $\phi' = \mathrm{H}^* l_1 \neq \mathrm{H}^* r_1 \wedge \ldots \wedge \mathrm{H}^* l_m \neq \mathrm{H}^* r_m$. Instead of $[\![q]\!]_{\mathcal{A}_\mathrm{H}} = \mathrm{H}[\![q]\!]_\mathcal{A}$ for all predicates q, we prove by induction that $[\![q]\!]^i_{\mathcal{A}_\mathrm{H}} = \mathrm{H}[\![q]\!]^i_\mathcal{A}$ for all $i \geq 0$ and all q, with the base case $[\![q]\!]^0_{\mathcal{A}_\mathrm{H}} = \mathrm{H}[\![q]\!]^0_\mathcal{A} = \emptyset$. Recall that $\mathrm{H}^*\theta s = \theta_\mathrm{H}\mathrm{H}^* s$ for terms s, where $\theta_\mathrm{H} = \{X_i \mapsto \mathrm{H}^*\theta X_i\}$. For a collection of terms $t_1, \ldots, t_k$, let $\theta = \{X_i \mapsto t_i\}$ (hence $\theta_\mathrm{H} = \{X_i \mapsto \mathrm{H}^* t_i\}$). Now assume that $f(t_1, \ldots, t_k) \in [\![p]\!]^{i+1}_\mathcal{A}$ by application of the clause c, i.e., $t_j \in [\![p_j]\!]^i_\mathcal{A}$ for all $1 \leq j \leq k$ and $\theta \models l_j \neq_\mathrm{H} r_j$ for all $1 \leq j \leq m$. By induction hypothesis, the first condition holds iff $\mathrm{H}^* t_j \in [\![p_j]\!]^i_{\mathcal{A}_\mathrm{H}}$ $\forall 1 \leq j \leq k$. The latter condition means that for all $1 \leq j \leq m$ we have $\mathrm{H}^*\theta l_j \neq \mathrm{H}^*\theta r_j$, which is equivalent to $\theta_\mathrm{H}\mathrm{H}^* l_j \neq \theta_\mathrm{H}\mathrm{H}^* r_j$ (for all j). Thus, $f(t_1, \ldots, t_k) \in [\![p]\!]^{i+1}_\mathcal{A}$ through application of c if and only if $\mathrm{H}^* t_j \in [\![p_j]\!]^i_{\mathcal{A}_\mathrm{H}}$ $\forall 1 \leq j \leq k$ and $\theta_\mathrm{H} \models \mathrm{H}^* l_j \neq \mathrm{H}^* r_j$ $\forall 1 \leq j \leq m$, which is equivalent to $\mathrm{H}^* f(t_1, \ldots, t_k) \in [\![p]\!]^{i+1}_{\mathcal{A}_\mathrm{H}}$ through application of c', with the ground substitution θ_H, since $\mathrm{H}(f)(\mathrm{H}^* t_1, \ldots, \mathrm{H}^* t_k) = \mathrm{H}^* f(t_1, \ldots, t_k)$. □

Example 25 Consider the tree homomorphism:

$$\mathrm{H} = \{b \mapsto a, f(X_1, X_2) \mapsto g(X_1, g(X_1, a))\}$$

where all other constructors are preserved. Then the set of automata clauses:

$$\begin{array}{lcl} p(b) & \Leftarrow & \\ p(f(X_1, X_2)) & \Leftarrow & p(X_1), p(X_2), X_1 \neq_{\mathrm{H}} f(X_2, X_2) \end{array}$$

is transformed into the following set of clauses:

$$\begin{array}{lcl} p(a) & \Leftarrow & \\ p(g(X_1, g(X_1, a))) & \Leftarrow & p(X_1), p(X_2), X_1 \neq g(X_2, g(X_2, a)) \end{array}$$

Note that the variable X_2 no longer occurs in the head of the second clause, while the variable X_1 occurs more than once. □

Deciding k-Finiteness. In the following we assume that an HDA $\mathcal{A}$ is given. Let us furthermore assume that we are given some predicate p occurring in $\mathcal{A}$ and some number $k > 0$. The problem to be solved, is:

1. to decide whether p accepts at least k terms in the least model of $\mathcal{A}$, and
2. if only $m < k$ terms are in $[\![p]\!]_{\mathcal{A}}$, to provide a witness of m such terms.

An algorithm based on emptiness decision for a model of tree automata for homomorphic images of regular tree languages, extended with path disequalities, could be derived from [GGRÀ10]. The time complexity, however, would be a tower of several exponentials in the worst case. Here, we proceed along the lines of efficiently deciding k-finiteness of automata with term disequalities in [SR11]. This base algorithm, though, must be extended as now heads are no longer just single constructor applications. Moreover, not all variables occurring in preconditions necessarily also occur in the head of a clause. Again, we start with a semi-algorithm that decides for a given HDA $\mathcal{A}$, a predicate p, and a number $k \geq 1$ whether $|[\![p]\!]_{\mathcal{A}}| \geq k$, by computing in every round i, $i \geq 1$, the sets $[\![q]\!]^i_{\mathcal{A}}$ for all predicates q until $|[\![p]\!]^i_{\mathcal{A}}| \geq k$ after some round i. Here, the sets $[\![q]\!]^i_{\mathcal{A}}$ can be computed from the sets $[\![q']\!]^{i-1}_{\mathcal{A}}$ by applying the implications $c \in \mathcal{A}$ — starting with $[\![q]\!]^0_{\mathcal{A}} = \emptyset$ for all q.

In order to obtain an algorithm, we establish an upper bound for the number of rounds which are needed for deciding HDA-k-finiteness. By a counting argument (analogous to [SR11] as described in Section 3.1.1), it suffices to increase the sets $[\![q]\!]^i_{\mathcal{A}}$ only up to $k + \sum_{c \in \mathcal{A}} \mathrm{dc}(c)$ trees for each predicate $q \neq p$. The claim is again based on a lemma which states that each term constraint ϕ of a clause c "filters out"

no more than $\mathrm{dc}(c)$ trees. This property of finite conjunctions of term disequalities also applies for HDA. More precisely, if a clause $q_1(t) \Leftarrow \alpha_1, q_2(X), \alpha_2, \phi$ can produce a tree for predicate q_1 in round i, and $X \in \mathrm{vars}(t)$, then the clause can produce at least $|[\![q_2]\!]^{i-1}_{\mathcal{A}}| - |\phi|_X$ trees until round i, where $|\phi|_X$ denotes the number of disequalities in ϕ which mention X.

Lemma 19 *Let $\mathcal{A}$ be an HDA, and $c \in \mathcal{A}$ a clause $q(t) \Leftarrow p_1(X_1), \ldots, p_k(X_k), \phi$. Assume that we are given a ground substitution $\theta \models \phi$ with $X_i\theta \in [\![p_i]\!]^d_{\mathcal{A}}$ $\forall\, i \in \{1, \ldots, k\}$ for some $d \geq 0$. Then $|[\![q]\!]^{d+1}_{\mathcal{A}}| \geq \max\{|[\![p_i]\!]^d_{\mathcal{A}}| - \mathrm{dc}(c) \mid X_i \in \mathrm{hv}(c)\}$.*

Proof. Let $X_j \in \mathrm{hv}(c)$ and $\phi \equiv C_1 \wedge \ldots \wedge C_m, m = \mathrm{dc}(c)$. Reorder the C_i s. t. X_j is mentioned exactly in $C_1, \ldots, C_l$, $0 \leq l \leq m$. Choose θ s.t. $\theta \models C_{l+1} \wedge \ldots \wedge C_m$ and $X_i\theta \in [\![p_i]\!]^d_{\mathcal{A}}$ for all $i \in \{1, \ldots, k\} \setminus \{j\}$. Making $C_1, \ldots, C_l$ true by choosing $\theta(X_j)$ can be considered as an instance of the pigeonhole principle implying that there are at least $|[\![p_j]\!]^d_{\mathcal{A}}| - l \geq |[\![p_j]\!]^d_{\mathcal{A}}| - m$ different trees in $[\![p_j]\!]^d_{\mathcal{A}}$ which satisfy all C_i, $1 \leq i \leq l$. Since $X_j \in \mathrm{hv}(c)$, each of them can be used in combination with the trees $X_i\theta, i \neq j$, to produce one tree for $[\![q]\!]^{d+1}_{\mathcal{A}}$. □

The main theorem is based on a procedure which iteratively constructs all facts $p(t)$ with a proof depth (i.e., the number of rounds that the procedure needs in order to find this fact) less than or equal to some $m \geq 0$ — which depends only on the number k, the number of predicates occurring in $\mathcal{A}$, and the total number of disequalities in $\mathcal{A}$. The theorem generalizes Theorem 10 in that now variables have to be taken into consideration which only occur in the precondition of a clause but not in the head, and the heads of clauses are not restricted to terms with exactly one constructor.

Theorem 22 *Let $\mathcal{A}$ be an HDA with n predicates and $d = \sum_{c \in \mathcal{A}} \mathrm{dc}(c)$ disequality constraints. Let k be a positive number. Then for all predicates p, it holds that $|[\![p]\!]_{\mathcal{A}}| < k$ if and only if $|[\![p]\!]^{n(d+k)}_{\mathcal{A}}| < k$.*

Proof. Direction "⇒" is trivial. For a proof of "⇐", consider the smallest number r such that $\forall p\ (|[\![p]\!]^r_{\mathcal{A}}| < d + k \Rightarrow [\![p]\!]^{r+1}_{\mathcal{A}} = [\![p]\!]^r_{\mathcal{A}})$. We have $r \leq n(d + k)$, since for all $1 \leq i \leq r$ at least one set $[\![p]\!]^{i-1}_{\mathcal{A}}$ with $|[\![p]\!]^{i-1}_{\mathcal{A}}| < d + k$ increases (i.e., $|[\![p]\!]^i_{\mathcal{A}}| > |[\![p]\!]^{i-1}_{\mathcal{A}}|$), which is possible only $n(d + k)$ times. Assume for a contradiction that $|[\![p]\!]_{\mathcal{A}}| \geq k$ but $|[\![p]\!]^r_{\mathcal{A}}| < k$ for some predicate p.

Let j be minimal such that $j > r+1$ and for some predicate p, $|[\![p]\!]^{j-1}_{\mathcal{A}}| < k$ but $|[\![p]\!]^j_{\mathcal{A}}| \geq k$ by application of some clause c_j (possibly among other clauses). Then for all $r + 1 \leq m < j - 1$ and predicates q, either $[\![q]\!]^{m+1}_{\mathcal{A}} = [\![q]\!]^m_{\mathcal{A}}$ or $|[\![q]\!]^m_{\mathcal{A}}| \geq k$. Consider a chain of clauses $c_{r+1}, \ldots, c_j$ where for all $i \in \{r + 1, \ldots, j - 1\}$

$$c_i \equiv q_i(t_i) \Leftarrow p_{i_1}(X_{i_1}), \ldots, p_{i_m}(X_{i_m}), \phi_i$$

is chosen such that for some variable X

(1) $[\![q_i]\!]^i_{\mathcal{A}} \neq [\![q_i]\!]^{i-1}_{\mathcal{A}}$ and $q_i(X)$ occurs in the body of c_{i+1}, and,

(2) if $X \in \mathrm{bv}(c_{i+1})$ then $|[\![q_i]\!]^{i-1}_{\mathcal{A}}| \leq \mathrm{dc}(c_{i+1})$.

Such a chain exists since $[\![q]\!]_i\mathcal{A} = [\![q]\!]^{i-1}_{\mathcal{A}}$ for all predicates q in the precondition of a clause c_{i+1} would imply that c_{i+1} cannot contribute to the set $[\![q_{i+1}]\!]^{i+1}_{\mathcal{A}} \setminus [\![q_{i+1}]\!]^i_{\mathcal{A}}$, and concerning (2), if $|[\![q_i]\!]^{i-1}_{\mathcal{A}}| > \mathrm{dc}(c_{i+1})$, the exact cardinality of $[\![q_i]\!]^m_{\mathcal{A}}$, $m \geq i$ is irrelevant for c_{i+1} since any ground substitution θ can be made to satisfy all disequalities in ϕ_{i+1} mentioning X by changing only $\theta(X)$ to an appropriate term of $[\![q_i]\!]^m_{\mathcal{A}}$.

Case 1. First assume that for all occurrences of $q_i(X)$ in the body of c_{i+1} we have $X \in \mathrm{hv}(c_{i+1})$. Then by Lemma 19, we have $|[\![q_i]\!]^{i-1}_{\mathcal{A}}| \leq |[\![q_{i+1}]\!]^i_{\mathcal{A}}| + \mathrm{dc}(c_{i+1})$ for $i = r+1, \ldots, j-1$. Since $|[\![q_j]\!]^{j-1}_{\mathcal{A}}| = |[\![p]\!]^{j-1}_{\mathcal{A}}| < k$, it follows that $|[\![q_{r+1}]\!]^r_{\mathcal{A}}| < \sum_{i=r+2}^{j} \mathrm{dc}(c_i) + k$. If $q_i \neq q_l$ for all $r+1 \leq i < l \leq j$, then $\sum_{i=r+2}^{j} \mathrm{dc}(c_i) \leq \sum_{c\in\mathcal{A}} \mathrm{dc}(c) = d$ — a contradiction because $|[\![q_{r+1}]\!]^r_{\mathcal{A}}| < d + k$ but $[\![q_{r+1}]\!]^{r+1}_{\mathcal{A}} \neq [\![q_{r+1}]\!]^r_{\mathcal{A}}$, violating the assumption in the definition of r. Otherwise, let $q_i = q_l$ such that $l - i$ is maximal. Then $[\![q_l]\!]^m_{\mathcal{A}} = [\![q_i]\!]^m_{\mathcal{A}} \forall m$, and $[\![q_i]\!]^{i-1}_{\mathcal{A}} \subseteq [\![q_i]\!]^{l-1}_{\mathcal{A}}$, hence $|[\![q_i]\!]^{i-1}_{\mathcal{A}}| \leq |[\![q_l]\!]^{l-1}_{\mathcal{A}}|$. Therefore, $|[\![q_{r+1}]\!]^r_{\mathcal{A}}| < k + \sum_{m=r+2}^{i} \mathrm{dc}(c_m) + \sum_{m=l+1}^{j} \mathrm{dc}(c_m)$. From maximality of $l - i$, it follows that $q_{r+1}, \ldots, q_i$ are pairwise disjoint with $q_{l+1}, \ldots, q_j$, and so also $c_{r+1}, \ldots, c_i$ are pairwise disjoint with $c_{l+1}, \ldots, c_j$. Multiple occurrences of clauses c_m therefore can now be recursively removed in the same way within both sums, proving that $|[\![q_{r+1}]\!]^r_{\mathcal{A}}| < k + \sum_{c\in\mathcal{A}} \mathrm{dc}(c)$. Again the contradiction follows.

Case 2. Assume $X \in \mathrm{bv}(c_{i+1})$ for some occurrence of $q_i(X)$ in the body of c_{i+1}. Instead of $|[\![q_j]\!]^{j-1}_{\mathcal{A}}| = |[\![p]\!]^{j-1}_{\mathcal{A}}| < k$ as in Case 1 we now have $|[\![q_i]\!]^{i-1}_{\mathcal{A}}| \leq \mathrm{dc}(c_{i+1})$ for some $i < j$, given by the definition of the chain $c_{r+1}, \ldots, c_j$. By choosing the minimal i, we get $|[\![q_{r+1}]\!]^r_{\mathcal{A}}| \leq \sum_{m=r+2}^{i+1} \mathrm{dc}(c_m) < \sum_{m=r+2}^{i+1} \mathrm{dc}(c_m) + k$, as in Case 1 (with $i+1 \leq j$ playing the role of j), completing the proof. □

Corollary 20 *Let $\mathcal{A}$ be an HDA. Then for all predicates p and numbers $k \geq 0$, it can be effectively decided whether $|[\![p]\!]_{\mathcal{A}}| \leq k$. Moreover, if $|[\![p]\!]_{\mathcal{A}}| \leq k$, then $[\![p]\!]_{\mathcal{A}}$ can be effectively computed.* □

From Lemma 18 and Corollary 20, we conclude:

Corollary 21 *Let $\mathcal{A}$ be a finite set of automata clauses with hom-disequalities. Then for all predicates p it can be effectively decided whether $[\![p]\!]_{\mathcal{A}} = \emptyset$.* □

Corollary 22 *Let $\mathcal{A}$ be a finite set of automata clauses with hom-disequalities. Then for all predicates p and numbers $k > 0$ it can be effectively decided whether $|\mathrm{H}[\![p]\!]_{\mathcal{A}}| < k$. Moreover, in case that $|\mathrm{H}[\![p]\!]_{\mathcal{A}}| = m < k$, a sequence $t_1, \ldots, t_m \in [\![p]\!]_{\mathcal{A}}$ can be effectively constructed such that the terms $\mathrm{H}t_i$, $i = 1, \ldots, m$, are pairwise distinct.* □

3.3.2 Expressiveness

This section compares automata classes extended with term, path, and hom-disequalities w.r.t. their expressiveness. First we show that hom-disequalities cannot be simulated by path disequalities, by presenting a specific language defined through a set of automata clauses with hom-disequalities which cannot be defined by a finite set of automata clauses with path disequalities only. Intuitively, path constraints can only express disequalities between subterms of at most a certain depth d as specified as part of the corresponding path expression. Hom-disequalities, however, may disregard an unbounded number of constructors on top of the tree.

Let $\Sigma = \{a, s, f\}$ where a, s, and f have arities 0, 1, and 2, respectively. Let H be the homomorphism defined by: $\mathrm{H}s = X_1$ while terms rooted a or f are not changed by H. Consider the language L with:

$$L = \{f(t_1, t_2) \mid t_1 \neq_{\mathrm{H}} t_2\}$$

The following automaton with hom-disequalities accepts L through predicate p.

$$p(f(X_1, X_2)) \;\Leftarrow\; \top(X_1), \top(X_2), X_1 \neq_{\mathrm{H}} X_2$$

Lemma 23 *There is no tree automaton for L with path disequalities only.*

Proof. Assume for a contradiction that an automaton $\mathcal{A}$ with path disequalities exists which accepts L through a predicate p. It is known [SN99, GGRÀ10, SR12] that to $\mathcal{A}$, a complement automaton $\mathcal{B}$ with *path equalities* only can be constructed such that there is a predicate $\overline{p}$ which accepts the complement language $\overline{L}$ given by: $\overline{L} = \mathcal{T}_\Sigma \setminus L = \{f(t_1, t_2) \mid t_1 =_{\mathrm{H}} t_2\} \cup \{a\} \cup \{s(t) \mid t \in \mathcal{T}_\Sigma\}$. Let d be the maximal depth of a path occurring in $\mathcal{B}$. For a ground term t, let $r_{t1}, r_{t2}, \ldots$ denote the infinite sequence of terms defined by: $r_{ti} = s^d(f(t, s^i(a)))$. Then $\mathrm{H}r_{ti} = f(\mathrm{H}t, a)$. Therefore, for $t \neq_{\mathrm{H}} t'$, it holds that for all i, j, $r_{ti} =_{\mathrm{H}} r_{tj}$ and $r_{ti} \neq_{\mathrm{H}} r_{t'j}$, but for all paths π occurring in $\mathcal{B}$, $r_{ti}.\pi \neq r_{tj}.\pi$ if $i \neq j$. As there are infinitely many sequences (r_{ti}) but only finitely many clauses, $\mathcal{B}$ has a clause

$$\overline{p}(f(X_1, X_2)) \;\Leftarrow\; q_1(X_1), q_2(X_2), \phi$$

such that for two terms t, t' with $t \neq_{\mathrm{H}} t'$, there are two terms t_1, t_2 from the sequence (r_{ti}) and two terms t_3, t_4 from the sequence $(r_{t'i})$ such that both $f(t_1, t_2)$ and $f(t_3, t_4)$ are in $[\![\overline{p}]\!]_{\mathcal{B}}$ by application of this clause. Especially, $t_1, t_3 \in [\![q_1]\!]_{\mathcal{B}}$ and $t_2, t_4 \in [\![q_2]\!]_{\mathcal{B}}$. Since ϕ is a conjunction of path equalities, ϕ must be equivalent to true because $t_1.\pi \neq t_2.\pi$ for all paths π occurring in $\mathcal{B}$. But then the clause also accepts the term $f(t_1, t_4)$ for $\overline{p}$ — contradiction. □

Now consider the set $\mathcal{C}$:

$$\begin{array}{lcl} \top(a) & \Leftarrow & \\ \top(f(X_1, X_2)) & \Leftarrow & \top(X_1), \top(X_2) \\ p(f(X_1, X_2)) & \Leftarrow & \top(X_1), \top(X_2), X_1 \neq X_2.1 \end{array}$$

for $\Sigma = \{a, f\}$. The language $[\![p]\!]_{\mathcal{C}}$ is not accepted by any automaton with hom-disequalities since hom-disequalities cannot directly access arbitrary subtrees independent of the labels in the tree. Let us denote by T the language $[\![p]\!]_{\mathcal{C}}$ of p w.r.t. $\mathcal{C}$. We have $T = \{f(t, a) \mid t \in \mathcal{T}_\Sigma\} \cup \{f(t, f(t_1, t_2)) \mid t, t_1, t_2 \in \mathcal{T}_\Sigma, t \neq t_1\}$.

Lemma 24 *There is no tree automaton $\mathcal{A}$ with hom-disequalities only that defines a predicate p with $[\![p]\!]_{\mathcal{A}} = T$.*

Proof. This example is the *unlabeled-path* variant of the corresponding example from [SR12] which provides the language $[\![p]\!]_{\mathcal{C}}$ that is accepted by an automaton with path disequalities but not by any automaton with term disequalities only.

Assume for a contradiction that an automaton $\mathcal{A}$ with hom-disequalities exists which accepts T through a predicate p. As in Lemma 23 we construct the complement automaton $\mathcal{B}$ with *hom-equalities* only, containing a predicate $\overline{p}$ which accepts the language $[\![\overline{p}]\!]_{\mathcal{B}} = \overline{T} = \mathcal{T}_\Sigma \setminus T = \{a\} \cup \{f(t, f(t, s)) \mid s, t \in \mathcal{T}_\Sigma\}$.
Case 1: If both X_1 and X_2 occur in $\mathrm{H}f$, then it holds for each equality of $\mathcal{B}$ that $l =_{\mathrm{H}} r$ if and only if $l = r$, and we refer to the proof in [SR12] that for $\overline{T}$ no automaton with term equalities exists.
Case 2: If $\mathrm{H}f \in \{X_1, X_2, g\}$ for a ground term g, then each equality $l =_{\mathrm{H}} r$ either is vacuously true or false (in case $g \neq \mathrm{H}a$), so that $\mathcal{B}$ may be considered an automaton with term equalities only, and as in Case 1, the proof in [SR12] applies.
Case 3: Assume therefore that $\mathrm{H}f$ equals a term $t = f(t_1, t_2)$ where t only contains the variable X_1 (respectively X_2). Then we define $d(t)$ to be the maximal $i \geq 0$ so that the path 1^i (respectively 2^i) is defined for t. Then $l =_{\mathrm{H}} r$ iff $d(l) = d(r)$. The contradiction now follows from an argument analogous to Lemma 23 based on the fact that there are infinitely many values $d(t)$ but only finitely many clauses in $\mathcal{B}$. □

Tree automata with term disequalities can be simulated by tree automata with path disequalities according to Theorem 17. Choosing the trivial tree homomorphism, on the other hand, it is clear that hom-disequalities can simulate term disequalities as well. Thus, we conclude that automata clauses with hom-disequalities are incomparable to automata clauses with path disequalities, while both classes are more expressive than automata with term disequalities only. Figure 3.1 shows a summary of classes of normalizable $\mathcal{H}_1$-extensions concerning their expressiveness.

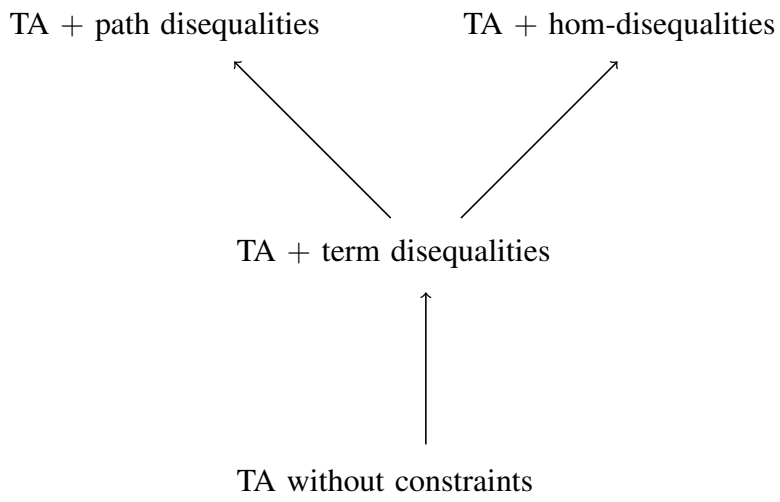

Figure 3.1: Relative expressiveness of tree automata (TA) extended with different kinds of disequality constraints. The transitive reduction of all proper inclusions is indicated by arrows.

3.3.3 $\mathcal{H}_1$-Normalization modulo Tree Homomorphism

In this section, we describe the adaptation of the normalization procedure to construct for every finite set $\mathcal{C}$ of $\mathcal{H}_1$-clauses with hom-disequalities a finite set $\mathcal{N}$ of normal clauses with hom-disequalities which is equivalent to $\mathcal{C}$. The following paragraphs collect the three normalization rules for the case of $\mathcal{H}_1$ extended with hom-disequalities. A significant modification w.r.t. ordinary term disequalities (as described in Section 3.1) is required for Splitting. Again, we refer to the current set of all implied clauses (whether originally present or added during normalization) as $\mathcal{C}$, while $\mathcal{N} \subseteq \mathcal{C}$ is the current subset of normal clauses in $\mathcal{C}$.

Resolution. Complex queries in preconditions are simplified by a resolution step with a *normal clause*. Assume that $\mathcal{C}$ contains a clause $h \Leftarrow \alpha_1, p(t), \alpha_2, \psi$. If $\mathcal{N}$ contains a clause $p(X_1) \Leftarrow \phi$, then we add the clause $h \Leftarrow \alpha_1, \alpha_2, \psi, \psi'$ where $\psi' = \phi[t/X_1]$. If $\mathcal{N}$ has a clause $p(f(X_1, \ldots, X_k)) \Leftarrow \beta, \phi$, and $t = f(t_1, \ldots, t_k)$, then

$$h \Leftarrow \alpha_1, \alpha', \alpha_2, \psi \wedge \psi'$$

is added with $\alpha' = \beta[t_1/X_1, \ldots, t_k/X_k]$ and likewise, $\psi' = \phi[t_1/X_1, \ldots, t_k/X_k]$.

Splitting. Splitting removes (or replaces) variables that are not contained in the head of a clause. For better comparability w.r.t. the Splitting step for term disequalities and w.l.o.g. we will assume here that all occurring hom-disequalities are of

the form $X \neq_{\mathrm{H}} t$ for a variable X and a term t not containing X, which can be obtained by the construction described at the beginning of Section 3.3. Assume that $\mathcal{C}$ contains a simple clause $h \Leftarrow \alpha, \psi$ and Y is a variable which occurs in the precondition α, ψ but neither occurs in h nor in any literal $q(t)$ with $t \neq Y$ within α. Then we can rearrange α into a sequence $\alpha', q_1(Y), \ldots, q_r(Y)$ where α' does not contain Y. Let ψ contain n disequalities involving Y.

In the case of term disequalities, the key issue is to decide whether the intersection $L = [\![q_1]\!]_{\mathcal{N}} \cap \ldots \cap [\![q_r]\!]_{\mathcal{N}}$ contains less than $n+1$ terms — and if so, to provide all terms of this set. In presence of the homomorphism H however, this is no longer sufficient. Instead, we must refer to the number of *images* of terms from L under H. In order to do so, we apply Lemma 18 from Section 3.3.1. Using Lemma 2, we can construct for $\mathcal{N}$ an HDA $\mathcal{A}$ such that $[\![p]\!]_{\mathcal{A}} = \mathrm{H}[\![p]\!]_{\mathcal{N}}$ for all predicates p of $\mathcal{N}$. By Corollary 22, we can decide k-finiteness (choosing $k = n+1$) of the conjunction of the q_i with respect to this automaton. If only $n' < n+1$ terms are in the set $\mathrm{H}(L)$, the corollary provides us with n' witnesses in the set L whose images under H are pairwise distinct.

Let $\mathrm{H}(L)$ contain m terms. If $m > n$, then we add the clause $h \Leftarrow \alpha', \psi'$ to the set $\mathcal{C}$ where ψ' is obtained from ψ by removing all disequalities that mention Y. If $m \leq n$, let $t_1, \ldots, t_m$ be the terms as provided by Corollary 22. Then we add to $\mathcal{C}$ all clauses

$$h \Leftarrow \alpha', \psi[t_i/Y], \quad i = 1, \ldots, m$$

Example 26 Consider again the clause

$$\mathit{error} \;\Leftarrow\; p_u(X), p_u(Y),\ X \neq_{\mathrm{H}} Y$$

from Example 5 (here, $X, Y, Z \ldots$ are used as variable names), and assume that $\mathrm{H}[\![p_u]\!]_{\mathcal{N}} = \{t\}$ for some ground term $t = f(\bot, b)$, where $\mathcal{N}$ denotes the whole (current) subset of normal clauses. One potential pre-image of t then is the term $t' = f(\mathit{secret}(a), b)$. Applying Splitting for variable Y (and assuming $t' \in [\![p_u]\!]_{\mathcal{N}}$), we obtain the new clause

$$\mathit{error} \;\Leftarrow\; p_u(X),\ X \neq_{\mathrm{H}} f(\mathit{secret}(a), b)$$

Now applying Splitting for variable X results in

$$\mathit{error} \;\Leftarrow\; f(\mathit{secret}(a'), b) \neq_{\mathrm{H}} f(\mathit{secret}(a), b)$$

for some possibly different pre-image $f(\mathit{secret}(a'), b) \in [\![p_u]\!]_{\mathcal{N}}$ of t. The disequality of the clause turns out to be false, which is due to the fact that p_u does not accept two or more terms that are *different modulo* H. □

Propagation. Propagation considers clauses $p(X_1) \Leftarrow q_1(X_1), \ldots, q_r(X_1), \psi$ ($r > 0$) where ψ only contains the variable X_1 (or none). If $\mathcal{N}$ contains normal clauses $q_j(f(X_1, \ldots, X_k)) \Leftarrow \alpha_j, \psi_j$ for $j = 1, \ldots, r$, then the normal clause

$$p(f(X_1, \ldots, X_k)) \Leftarrow \alpha_1, \ldots, \alpha_r, \psi_1 \wedge \ldots \wedge \psi_r \wedge \psi'$$

is added with $\psi' = \psi[f(X_1, \ldots, X_k)/X_1]$.

The correctness of the whole construction for hom-disequalities can be proven along the lines of the corresponding proof of Theorem 13.

Theorem 23 *Let $\mathcal{C}$ denote a finite set of $\mathcal{H}_1$-clauses with hom-disequalities. Let $\overline{\mathcal{C}}$ denote the set of clauses obtained from $\mathcal{C}$ by adding all clauses according to the normalization rules. Then the subset $\mathcal{N}$ of all normal clauses in $\overline{\mathcal{C}}$ is equivalent to $\mathcal{C}$, i.e., $[\![p]\!]_{\mathcal{C}} = [\![p]\!]_{\mathcal{N}}$ for every predicate p occurring in $\mathcal{C}$.* □

Termination. As is the case with term or path disequalities, termination can be achieved by avoiding to add clauses that are subsumed by the current set of clauses. However, in presence of a (non-trivial) tree homomorphism H, we need to apply H to the topmost constructors of the left-hand sides and right-hand sides of hom-disequalities in order to decide subsumption. Otherwise, the constraints may grow unnecessarily.

Example 27 Consider the tree homomorphism $\mathrm{H} = \{f \mapsto g(X_2), g \mapsto X_1, a \mapsto b, b \mapsto a\}$ over the alphabet $\Sigma = \{a, b, g, f\}$. Then for an *arbitrary* number $i \geq 0$, the disjunction of (conjunctions of) hom-disequalities

$$X_1 \neq_{\mathrm{H}} X_2 \vee X_1 \neq_{\mathrm{H}} g(X_2) \vee, \ldots, \vee X_1 \neq_{\mathrm{H}} g^i(X_2) \tag{3.2}$$

by definition is equivalent to:

$$\mathrm{H}X_1 \neq \mathrm{H}X_2 \vee, \ldots, \vee \mathrm{H}X_1 \neq \mathrm{H}X_2 \tag{3.3}$$

with the convention that a ground substitution θ satisfies the constraint if and only if $\mathrm{H}(\theta(X_1)) \neq \mathrm{H}(\theta(X_2))$, i.e., the substitution first applies, and the resulting terms are then transformed according to H. Then the constraint (3.2) is equivalent to the single disequality

$$X_1 \neq_{\mathrm{H}} X_2 \tag{3.4}$$

Especially, i is not bounded here; hence, formulas of type (3.2) may *grow* arbitrarily although all these formulas are equivalent to just one single hom-disequality. □

Example 28 Let H be defined as in Example 27, i.e., $\mathrm{H} = \{f \mapsto g(X_2), g \mapsto X_1, a \mapsto b, b \mapsto a\}$. The disjunction

$$X_1 \neq_{\mathrm{H}} a \vee X_1 \neq_{\mathrm{H}} g(b) \vee X_1 \neq_{\mathrm{H}} f(X_2, X_2) \tag{3.5}$$

is equivalent to:

$$\mathrm{H}X_1 \neq b \vee \mathrm{H}X_1 \neq a \vee \mathrm{H}X_1 \neq g(\mathrm{H}X_2) \tag{3.6}$$

which in turn is equivalent to

$$X_1 \neq_{\mathrm{H}} a \vee X_1 \neq_{\mathrm{H}} b \vee X_1 \neq_{\mathrm{H}} f(a, X_2) \tag{3.7}$$

where we choose a, b, and $f(a, X_2)$ as pre-images under H of b, a, and $g(X_2)$, respectively. Now consider the case that $\mathrm{H}a$ is defined as a (instead of b), i.e., the symbol a is not changed by H. Then the disjunction (3.5) is equivalent to

$$\mathrm{H}X_1 \neq a \vee \mathrm{H}X_1 \neq g(\mathrm{H}X_2) \tag{3.8}$$

since both a and $g(b)$ are mapped to the ground term a. □

For the decision of subsumption, it is sufficient to consider the formulas (3.3), (3.6), and (3.8), where the topmost constructors are replaced according to H — so that the effects of H only apply to the symbols occurring in later substitutions of the variables X_i.

Theorem 24 *Let $\mathcal{C}$ denote a finite set of $\mathcal{H}_1$-clauses with hom-disequalities. Let $\overline{\mathcal{C}}$ denote the set of clauses obtained from $\mathcal{C}$ by adding all clauses according to Resolution, Splitting, and Propagation that are not subsumed by the current set of clauses. Then $\overline{\mathcal{C}}$ is finite.*

Proof. Concerning the core clauses, the proof agrees with that of Theorem 15 for normalization of $\mathcal{H}_1$-clauses with term disequalities. With the interpretation of disequalities modulo tree homomorphism, there still are only finitely many cores.

Now consider a sequence $\psi_i, i \geq 1$, of conjunctions of hom-disequalities. It remains to show that the disjunction $\bigvee_{i=1}^{m} \psi_i, m \geq 1$, eventually becomes stable, i.e., there exists some M such that $\bigvee_{i=1}^{m} \psi_i = \bigvee_{i=1}^{M} \psi_i$ for all $m \geq M$. In order to construct such an M consider the sequence $\psi_{\mathrm{H},i}, i \geq 1$, of ordinary term disequalities where $\psi_{\mathrm{H},i}$ is obtained from ψ_i by replacing each hom-disequality $s \neq_{\mathrm{H}} t$ with $\mathrm{H}(s) \neq \mathrm{H}(t)$. Then θ is a solution to ψ_i, iff $\mathrm{H} \circ \theta$ is a solution to $\psi_{\mathrm{H},i}$. In Section 3.1 we have shown that disjunctions of sequences of conjunctions of ordinary term disequalities are ultimately stable. Therefore, there exists an M'

such that $\bigvee_{i=1}^{m} \psi_{\mathrm{H},i} = \bigvee_{i=1}^{M'} \psi_{\mathrm{H},i}$ for all $m \geq M'$. Then we choose the constant M as M'. In order to prove that the sequence $\bigvee_{i=1}^{m} \psi_i$ for $m \geq M'$ is implied by $\bigvee_{i=1}^{M'} \psi_i$, assume that θ is a solution of $\bigvee_{i=1}^{m} \psi_i$ for some $m \geq M'$. Then $\mathrm{H} \circ \theta$ is a solution of $\bigvee_{i=1}^{m} \psi_{\mathrm{H},i}$ and therefore also of $\bigvee_{i=1}^{M'} \psi_{\mathrm{H},i}$. Consequently, θ must also be a solution of $\bigvee_{i=1}^{M'} \psi_i$. Therefore, we conclude that also $\bigvee_{i=1}^{m} \psi_i = \bigvee_{i=1}^{M'} \psi_i$ for all $m \geq M$. This implies that eventually all clauses that can be added are subsumed. Therefore, the normalization procedure terminates. □

According to Theorem 23 and Theorem 24, for every finite set $\mathcal{C}$ of $\mathcal{H}_1$-clauses with hom-disequalities an equivalent finite set $\mathcal{N}$ of normal clauses can be constructed. By Lemma 2, $\mathcal{N}$ can then be transformed to an equivalent finite set $\mathcal{A}$ of automata clauses. Finally, by Corollary 21, emptiness is decidable for every predicate defined by $\mathcal{A}$. Altogether, we obtain:

Theorem 25 *To every finite set $\mathcal{C}$ of $\mathcal{H}_1$-clauses with hom-disequalities, a finite set $\mathcal{A}$ of automata clauses with hom-disequalities can be effectively constructed such that for every predicate p of $\mathcal{C}$, $[\![p]\!]_{\mathcal{C}} = [\![p]\!]_{\mathcal{A}}$. In particular, emptiness is decidable for the sets $[\![p]\!]_{\mathcal{C}}$.* □

Chapter 4

Perspectives

Let us briefly recap the described work and collect some ideas for future work. We have presented several extensions of first-order Horn clauses with constraints. An extension of finite sets of automata clauses – a particular class of Horn clauses corresponding to finite bottom-up tree automata – with equality and disequality term constraints was presented in Section 2.2. The languages accepted by this new class of *term-constrained automata* (TCA) are closed under Boolean operations for tree languages, i.e., union, intersection, and complementation. TCA are strictly more expressive than tree automata with constraints between brothers of [BT92], but less expressive than tree automata with path constraints [Mon81]. While emptiness is undecidable for the latter class of automata, we have shown that emptiness for TCA is decidable.

When adding *equality* term constraints, however, the class $\mathcal{H}_1$ becomes undecidable. In fact, adding either of the two main $\mathcal{H}_1$ features – auxiliary variables, or complex preconditions – to automata clauses with term equalities already results in an undecidable class of Horn clauses. For $\mathcal{H}_1$ extended with term *disequalities*, on the other hand, we have shown that finite sets of such clauses can still be transformed to tree automaton form (extended with disequalities), and the resulting automata with term disequalities are decidable.

While for tree automata with equality constraints, both the extension to path equalities and the extension to equalities of images of terms under a given tree homomorphism yield undecidability, the disequalities for $\mathcal{H}_1$ can be further extended. Path disequalities enable to express disequality relations between subterms at different levels while disregarding the rests of the compared trees. Hom-disequalities, on the other hand, allow for comparisons that disregard subterms, depending on the constructors under which these subterms occur.

Future work and possible applications. Not surprisingly, there is still much to do. E.g., for the algorithms proposed in our work it would be interesting

to determine precise complexity bounds. So far we have developed prototype implementations for most of the described transformation and decision procedures. It remains to evaluate how they behave on practical examples and in how far $\mathcal{H}_1$-clauses with disequalities can be applied to enhance analyses of term-manipulating programs. We also wish to see our methods applied in the modelling of protocols. A direct treatment of disequality conditions is now conveniently possible in Horn clause representations.

Let us amplify here one concrete example application: cryptographic protocols. Such protocols play a key role in mechanisms for protecting confidential data from unauthorized access. They are widely used with good success, e.g., for authentication purposes [BM03]. The insight that it is often insufficient to only "manually" prove that a protocol is secure has grown at the end of last century. One witness exemplifying the need of automatic verification in the area of cryptographic protocols is the famous Needham-Schroeder protocol [NS78] for public key encryption whose insecureness remained undetected for more than one and a half decades [Low95].

The modelling of cryptographic protocols for analysis and verification purposes normally abstracts from cryptography by *Dolev-Yao models* [DY83], replacing the underlying cryptographic operations by term algebras. Typically, the models of cryptographic protocol analyses contain (at least) one *attacker*, and the goal is to prove that it is impossible for the attacker to deduce certain knowledge, e.g., a *secret* that must be protected from unauthorized access. Protocol steps as well as actions of potential attackers who try to exploit weaknesses of the protocol often can be modelled by Horn clauses in a natural way. E.g., [Bla01] provides protocol analyses based on Prolog descriptions, and [NNS02] analyzes, by means of $\mathcal{H}_3$-clauses (a subclass of $\mathcal{H}_1$), the Spi calculus, an extension of the Pi calculus with primitives for encryption and decryption, intended to support cryptographic protocol modelling and analysis.

The result of such an analysis then is an overapproximation of the attacker's knowledge and – hopefully – the proof that it does not contain anything that the protocol is supposed to protect from unauthorized access. Technically, this means an empty intersection of the sets of terms that are accepted by the corresponding predicates for the attacker and the secret, respectively.

Since Horn clauses do not allow for a direct treatment of disequality, it is common practice in the modelling of protocols that negative information is not directly modelled at all – even if the protocol explicitly specifies disequality conditions (see, e.g., [Aba03, AF04]). The knowledge lost in this way may make a subsequent analysis doubtful or even unsound as it ignores the fact that an attacker could be able to deduce useful knowledge not only from successful but also from unsuccessful attempts at protocol steps – such as, e.g., message decryption.

Example 29 Let $know$ be a predicate collecting the knowledge of an attacker. Assume that keys are modelled as terms $b_1(b_2(\ldots(b_l(\bot))\ldots))$ where $b_i \in \{a, b\}$ are bits, and l denotes the key length. A *partially decrypted* encrypted message is given by a term

$$enc(secret, b_1(\ldots(b_i(\bot))\ldots), s^i(0)) \qquad 0 \leq i \leq l, \text{ all } b_j \in \{a, b\}$$

where still $i \geq 0$ bits have to be decrypted *step-wise*. While the number of remaining bits shall be unknown, a fully decrypted secret can be obtained by the attacker:

$$know(sec(S)) \Leftarrow know(enc(S, \bot, 0))$$

The ability of the attacker to step-wise decrypt messages is modelled by the clauses:

$$\begin{array}{lcl} know(enc(S, K, N)) & \Leftarrow & know(key(d(X), s(N))), know(enc(S, d(K), s(N))) \\ know(key(X, N)) & \Leftarrow & know(key(d(X), s(N))), know(enc(S, d(K), s(N))) \end{array}$$

for $d \in \{a, b\}$. Then an attacker who knows the key $k = b_1(\ldots(b_l(\bot))\ldots)$ can obtain the secret, i.e., from the facts

$$know(enc(secret, k, s^l(0))) \quad \text{and} \quad know(key(k, s^l(0)))$$

the fact $know(sec(secret))$ can be deduced[1]. If, on the other hand, the attacker's key does not match the encryption key, the decryption "silently" fails so far. Thus, the attacker is only successful if he or she knows the encryption key.

Assume now that the attacker knows an *arbitrary* key, e.g., the key that consists of a's only:

$$know(key(a(\ldots(a(\bot))\ldots), s^l(0))) \Leftarrow$$

Then the attacker could step-wise try that key, and if there is feedback in case that the attempt was not successful, this information might be used by the attacker in order to adjust the key. This step may conveniently be modelled with the help of disequality constraints. E.g., the $\mathcal{H}_1$-clauses, for $d \in \{a, b\}$,

$$fail(key(X, N)) \quad \Leftarrow \quad know(key(X, N)), \quad know(enc(S, d(K), N)), \quad X \neq d(Y) \tag{4.1}$$

store the information that the attacker's approach of "decrypting" the bit at position N has failed. The attacker can now use this knowledge in order to adjust the bit at position N:

$$know(key(d'(X), N)) \quad \Leftarrow \quad fail(key(d(X), N))$$

[1] The analysis problem would ask for satisfiability of a fresh predicate p when the clause $p(X) \Leftarrow know(sec(secret))$ is added.

for $d, d' \in \{a, b\}$, $d \neq d'$. In order to achieve the $\mathcal{H}_1$ properties (in particular, a flat head), an overapproximation could be applied here. Alternatively, the clauses:

$$know(key(X, N)) \quad \Leftarrow \quad fail(key(d(X'), N)),\ X \neq d(Y)$$

for $d \in \{a, b\}$ express that the attacker can construct an arbitrary key with the corrected bit at position N. With a modelling of these steps, the protocol can now be shown to be potentially insecure. □

Including failure information hence can reveal a particularly simple attack which might have been "overlooked" by a straight-forward modelling neglecting negative information. Note that a simple pattern matching approach for clause (4.1) of Example 29 would disturb the $\mathcal{H}_1$ property (the head $fail(key(a(X'), N))$ is not flat, containing the two constructors key and a), which might lead to undecidability of the analysis problem.

The important thing here is that the attacker's abilities have to be modelled *according to* the protocol steps; guessing a whole key, e.g., is not permitted in the Dolev-Yao abstraction from "real actions". Since the attacker can non-deterministically choose actions to take, these actions, as a general rule, correspond to protocol steps and their *implications*. Therefore, it is important to model these single steps as adequately as possible.

Other potential applications consist in the analysis of functional programming languages like Haskell [Hei94] or logic programs such as Prolog [FSVY91]. For functional programs, the goal is to determine for all occurring sub-expressions e safe supersets of the sets of terms to which e may evaluate. Jones describes these sets by means of *regular tree grammars* (in [AH87], see, e.g., [JA07]). In [HJ90], *set constraints* are used to describe such sets. Horn clauses with disequality constraints would constitute a decent alternative to both. Consider, for instance, a functional program that computes with lists of integers. Typically, functional programs provide a function hd that returns the first element of a given list. A call to this function with an empty list (denoted $[\,]$) will cause some kind of exception, and might halt the whole program. The expression

$$\textbf{if } X \neq [\,] \textbf{ then } \mathrm{hd}\, X \textbf{ else null}$$

is safe, but the analyzing tool needs to "know" that X is a non-empty list when the expression $\mathrm{hd}\, X$ is reached. A constant propagation analysis could find out that in the *else*-part, $X = [\,]$ holds. This, however, does not help in this example, since we are interested in *negative knowledge* for the *then*-branch.

For the analysis of logic programs, [HJ90] uses *set constraints* in order to approximate the least model of a program, while [FSVY91] uses *uniform Horn clauses* for that purpose. Using $\mathcal{H}_1$ with disequalities instead, one could perhaps better account for negative conditions.

Acknowledgements

In this section, we feel that we should be thankful.

Proposition 25 *This thesis would not have been possible without the valuable support of great people, including,* but not limited to

> *Helmut Seidl, Javier Esparza,*
> *Prof. Dr. rer. nat./Harvard Univ. Erhard Plödereder,*
> *Florent Jacquemard, Hubert Comon-Lundh, Andreas Gaiser*

Proof sketch. I immensely profited from their help and collaboration, including illuminating discussions, encouragement, and proofreading. Thank you all! □

Funding Acknowledgement

I was supported by the DFG-Graduiertenkolleg 1480 (PUMA).

Errata

Errare humanum est. Wie jedes Werk enthält auch dieses Fehler. Evtl. lohnt sich daher ein Blick ins Zwischennetz, z.B. dorthin:

`http://www2.in.tum.de/`

Bibliography

[Aba03] Martín Abadi. Private authentication. In *2. PET 2002*, volume 2482 of *LNCS*, pages 27–40. Springer, 2003.

[AF04] Martín Abadi and Cédric Fournet. Private authentication. *Theoretical Computer Science*, 322(3):427–476, 2004.

[AH87] Samson Abramsky and Chris Hankin. *Abstract Interpretation of Declarative Languages*. Ellis Horwood, 1987. ISBN 0-7458-0109-9.

[BCG+10] Luis Barguño, Carles Creus, Guillem Godoy, Florent Jacquemard, and Camille Vacher. The emptiness problem for tree automata with global constraints. In *LICS*, pages 263–272. IEEE Computer Society Press, 2010. ISBN 978-0-7695-4114-3.

[Bla01] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *CSFW*, pages 82–96. IEEE Computer Society, 2001. ISBN 0-7695-1146-5.

[BM03] Colin Boyd and Anish Mathuria. *Protocols for Authentication and Key Establishment*. Springer, 2003. ISBN 3-540-43107-1.

[BR05] Michele Bugliesi and Sabina Rossi. Non-interference proof techniques for the analysis of cryptographic protocols. *Journal of Computer Security*, 13(1):87–113, 2005.

[BRS07] A. Baskar, R. Ramanujam, and S. P. Suresh. Knowledge-based modelling of voting protocols. In *TARK*, pages 62–71. Presses universitaires de Louvain, 2007. ISBN 978-2-87463-077-4.

[BT92] Bruno Bogaert and Sophie Tison. Equality and disequality constraints on direct subterms in tree automata. In *STACS*, volume 577 of *LNCS*, pages 161–171. Springer, 1992.

[CCC+94] Anne-Cécile Caron, Hubert Comon, Jean-Luc Coquidé, Max Dauchet, and Florent Jacquemard. Pumping, cleaning and symbolic constraints solving. In *ICALP*, volume 820 of *LNCS*, pages 436–449. Springer, 1994.

[CCD93] Anne-Cécile Caron, Jean-Luc Coquidé, and Max Dauchet. Encompassment properties and automata with constraints. In *RTA*, volume 690 of *LNCS*, pages 328–342. Springer, 1993.

[CDG+07] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

[Cha07] Konstantinos Chatzikokolakis. *Probabilistic and Information-Theoretic Approaches to Anonymity*. PhD thesis, École polytechnique, 2007.

[CJ94] Hubert Comon and Florent Jacquemard. Ground reducibility and automata with disequality constraints. In *STACS*, volume 775 of *LNCS*, pages 151–162. Springer, 1994.

[CJ97] Hubert Comon and Florent Jacquemard. Ground reducibility is exptime-complete. In *LICS*, pages 26–34. IEEE Computer Society, 1997. ISBN 0-8186-7925-5.

[DY83] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[FOO93] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *AUSCRYPT*, volume 718 of *LNCS*, pages 244–251. Springer, 1993.

[FSVY91] Thom W. Frühwirth, Ehud Y. Shapiro, Moshe Y. Vardi, and Eyal Yardeni. Logic programs as types for logic programs. In *LICS*, pages 314–328. IEEE Computer Society, 1991. ISBN 0-8186-2230-X.

[FTT08] Emmanuel Filiot, Jean-Marc Talbot, and Sophie Tison. Tree automata with global constraints. In *DLT*, volume 5257 of *LNCS*, pages 314–326. Springer, 2008.

[GGRÀ10] Guillem Godoy, Omer Giménez, Lander Ramos, and Carme Àlvarez. The hom problem is decidable. In *STOC*, pages 485–494. ACM, 2010. ISBN 978-1-4503-0050-6.

[GL05] Jean Goubault-Larrecq. Deciding H1 by resolution. *Information Processing Letters*, 95(3):401–408, 2005.

[GLP05] Jean Goubault-Larrecq and Fabrice Parrennes. Cryptographic protocol analysis on real C code. In *VMCAI*, volume 3385 of *LNCS*, pages 363–379. Springer, 2005.

[GM82] Joseph A. Goguen and José Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society, 1982.

[Hei94] Nevin Heintze. Set-based analysis of ML programs. In *LFP*, pages 306–317. ACM, 1994. ISBN 0-89791-643-3, 336.

[HJ90] Nevin Heintze and Joxan Jaffar. A decision procedure for a class of set constraints (extended abstract). In *LICS*, pages 42–51. IEEE Computer Society, 1990. ISBN 0-8186-2073-0.

[JA07] Neil D. Jones and Nils Andersen. Flow analysis of lazy higher-order functional programs. *Theoretical Computer Science*, 375(1-3):120–136, 2007.

[Low95] Gavin Lowe. An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1995.

[LW09] Christof Löding and Karianto Wong. On nondeterministic unranked tree automata with sibling constraints. In *FSTTCS*, volume 4 of *LIPIcs*, pages 311–322. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2009.

[Mon81] J. Mongy. Transformation de noyaux reconnaissables d'arbres, 1981. PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, Université des Sciences et Technologies de Lille, Villeneuve d'Ascq, France.

[MORS05] Markus Müller-Olm, Oliver Rüthing, and Helmut Seidl. Checking Herbrand equalities and beyond. In *VMCAI*, volume 3385 of *LNCS*, pages 79–96. Springer, 2005.

[NNS02] Flemming Nielson, Hanne Riis Nielson, and Helmut Seidl. Normalizable Horn Clauses, Strongly Recognizable Relations, and Spi. In *SAS*, volume 2477 of *LNCS*, pages 20–35. Springer, 2002.

[NS78] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[RS10] Andreas Reuß and Helmut Seidl. Bottom-up tree automata with term constraints. In *LPAR-17*, volume 6397 of *LNCS*, pages 581–593. Springer, 2010.

[RS12] Andreas Reuß and Helmut Seidl. Crossing the syntactic barrier: Hom-disequalities for H1-clauses. In *CIAA*, volume 7381 of *LNCS*, pages 301–312. Springer, 2012.

[RV01] John Alan Robinson and Andrei Voronkov. *Handbook of Automated Reasoning*. Elsevier and MIT Press, 2001. ISBN 0-444-50813-9, 0-262-18223-8.

[SN99] Helmut Seidl and Andreas Neumann. On guarding nested fixpoints. In *CSL*, volume 1683 of *LNCS*, pages 484–498. Springer, 1999.

[SR11] Helmut Seidl and Andreas Reuß. Extending H1-clauses with disequalities. *Information Processing Letters*, 111(20):1007–1013, 2011.

[SR12] Helmut Seidl and Andreas Reuß. Extending H1-clauses with path disequalities. In *FoSSaCS*, volume 7213 of *LNCS*, pages 165–179. Springer, 2012.

[Tom92] M. Tommasi. Automates d'arbres avec tests d'égalité entre cousins germains, 1992. Mémoire de DEA, Univ. Lille I.

[TW68] James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[vEK76] Maarten H. van Emden and Robert A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.

[Wei99] Christoph Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In *CADE*, volume 1632 of *LNCS*, pages 314–328. Springer, 1999.

Glossary

- $[\![p]\!]_{\mathcal{A}}$ Set of ground terms recognized (or: accepted) by predicate p in the least model of the set $\mathcal{A}$ of clauses – page 17
- $[\![p]\!]^i_{\mathcal{A}}$ Set of ground terms t where the fact $p(t)$ can be deduced by clauses from $\mathcal{A}$ using at most i rounds of fixpoint iteration – page 17
- bv Variables occurring in the body but not in the head of a Horn clause – page 66
- dc Number of atomic disequality constraints within a single constrained Horn clause – page 24
- H denotes a tree homomorphism – page 15
- $\mathcal{H}_1$ subclass of definite first-order Horn clauses; unary predicates, the head has at most one constructor and no variable twice – page 17
- HDA Generalized automaton (complex heads, auxiliary variables, term disequalities) – page 66
- hv Variables occurring in the head of a Horn clause – page 66
- $\mathcal{M}$ Least model of the indexed set of Horn clauses – page 16
- $\top$ Predicate defined such that it holds for any ground term over the given ranked alphabet Σ – page 18
- $\mathcal{T}_\Sigma$ Set of all ground (i.e., variable-free) terms, or trees; $\mathcal{T}_\Sigma(\mathbf{V})$ denotes terms containing variables from the set $\mathbf{V}$ – page 14
- TCA (Tree) automaton with term equalities and term disequalities – page 26
- $\text{TCA}_{\neq}$ Generalized automaton with term disequalities – page 24