UNIVERSITÉ DE GRENOBLE

Ecole Doctorale Mathématiques, Sciences et
Technologies de l'Information, Informatique

Habilitation à Diriger des Recherches

Automata and Logics for Program Verification

Radu IOSIF

Acknowledgments

TODO

Contents

Chapter 1

Introduction

The verification of algorithms, programs and, in general, computer systems, is a major research area, stemming in the seminal works of Church and Turing, which set the basis of modern computing science. The early study of basic models of computation such as the λ-calculus [Chu32] and the Turing Machines [Tur37] revealed that most basic problems are unsolvable, such as the existence of (1) a normal form for a given λ-term, (2) an execution leading to a given control location (the *reachability* problem), or (3) an infinite execution (the *halting* problem). These undecidability results are direct consequences of the unsolvability of the *Entscheidungsproblem* [HA28], which asks, given a logical formula, for a mechanical way of proving its validity.

The story of program verification thus starts with a logical paradox and continues with a psychological one, which finally turned undecidability into rather efficient practical algorithms. One possible explanation is that modern society is increasingly dependent on software, whose failure can result in dramatic human and economical loss. Consequently, the process of software development demands for rigorous verification methods that guarantee the absence of design and implementation errors. Moreover, many practical instances of verification problems are feasible, despite the barrier imposed by theoretical undecidability, or prohibitive complexity lower bounds.

During the last decades, the pioneering works of Floyd [Flo67] and Hoare [Hoa69] have enabled major steps in this direction, leading to various verification techniques such as: abstract interpretation [CC79], predicate abstraction [GH96], assisted proof techniques [FM07, Abr96], regular model checking [BJNT00] and shape analysis [SRW02], among others. The great majority of these methods have integrated industrial-scale platforms for software analysis and development, such as the Spec# and CodeContracts

projects at Microsoft [BFL+11], the SPARK framework for Ada [Bar03], the FRAMA-C [CKK+12], POLYSPACE and ASTRÉE [BCC+03, CCF+07] static analyzers for C, and the INFER shape analysis tool [CD11] for C, Java and JavaScript, at Facebook.

Generally speaking, one can distinguish between (i) *certification* methods, that provide proofs of correctness (absence of errors) with respect to a certain semantic model of the program, and (ii) *bug-finding* methods, that explore systematically the state space of the program, in search for errors. First, certification is used at early stages during program development, and is expected to perform in a reasonably small amount of time, like a compiler. To this end, a certification approach either (a) gives up on precision and reports false alarms (as in *abstract interpretation* [CC79]), or (b) requires the user to provide annotations, such as pre-, post-conditions, invariants and ranking functions to help the verifier (as in Floyd-Hoare *deductive verification* [Flo67, Hoa69]). Depending on the type of property, a *certificate* is either an inductive invariant, defining an over-approximation of the sets of reachable configurations (in case of reachability properties), or a ranking function, that witnesses termination (for more general temporal properties that involve reasoning about infinite computations).

Second, bug-finding is used at later stages of development and is granted more time, in exchange of precision. Here no false alarms are reported, but, on the other hand, the verification procedure does not terminate in all cases[1]. As examples of bug-finding methods, we mention *predicate abstraction model checking* [GH96] with counter-example guided abstraction refinement (CEGAR) [CGJ+00] and *acceleration* [Boi99, FL02a, BIK10].

Program verification has an intrinsic connection with logic and automated reasoning. First of all, logic is used to specify the expected behavior of a program, in the form of pre- and post-conditions (contracts). Second, logic is used internally as a symbolic representation of the potentially infinite sets of configurations that must be considered by the verifier. The verification problem is usually decomposed into a large set of logical conditions that are checked for validity. As a result, the decidability status and complexity of certain program verification problems can usually be found by reduction to the validity problem for a suitable logical fragment.

The connection between models of computation and logic can be traced back to the encoding of λ-calculus and Turing Machines in first-order integer arithmetic [Chu32, Tur37]. More recently, this relation has provided several deep decidability results in logic, via a connection with the theory of formal

[1] If the procedure terminates, a correctness certificate can also be provided.

languages and automata. The seminal works of Büchi [B̈62] and Rabin [Rab68] show that the languages recognized by finite state (tree) automata are exactly the sets of models of Monadic Second Order logic (MSO) formulae interpreted over possibly infinite words (trees). The reduction builds, (1) for each MSO formula ϕ an automaton A_ϕ, such that a word (tree) is a model of ϕ if and only if it is recognized by A_ϕ, and (2) for each automaton A, an MSO formula Φ_A, such that a word (tree) is recognized by A if and only if it is a model of Φ_A. Since language emptiness is decidable for finite-state automata over finite alphabets, this relation entails the decidability of the satisfiability, and thus, of the validity[2] problem for MSO.

The logic-automata connection extends also to fragments of first-order arithmetic, such as the theory of natural numbers $\langle \mathbb{N}, +, V_p \rangle$, with the classical interpretation of addition, where $V_p(x)$ is the largest power of p that divides x, for a prime number p [BHMV94]. The set of p-adic expansions of the models of a formula $\phi(x_1, \ldots, x_n)$ in this theory is the language of a finite automaton A_ϕ over the alphabet $\{0, \ldots, p-1\}^n$. Dually, for any automaton A recognizing words over this alphabet, there exists a formula $\Phi_A(x_1, \ldots, x_n)$ whose set of models corresponds to the language of the automaton. This argument establishes the decidability of the $\langle \mathbb{N}, +, V_p \rangle$ theory.

The relation with automata theory establishes a sharp boundary between decidability and undecidability in many logical theories. To this date, only few decidable logics do not have known automata-based proofs of decidability[3]. It is tempting to ask whether every decidable logic has an automata-based decidability proof, but this remains a conjecture, so far.

Reducing a program verification task to several verification conditions expressed in a decidable logic is the challenge faced by every verification technique. Such reductions are usually difficult because the program configurations handled by the verifier are way more complex than the structures (words,trees) recognized by automata. Indeed, one can distinguish several degrees of complexity.

First, automata always work with finite alphabets, which is not the case of programs. For instance, the set of configurations of a program handling an integer array is a set of words over the infinite alphabet of integer array values, and can hardly be represented using a finite alphabet. This restriction motivates the study of automata and logics using very large or infinite alphabets [BDM+11, DA14].

[2] A formula ϕ is valid if and only if $\neg\phi$ is not satisfiable.

[3] One such example is the theory $\langle \mathbb{N}, +, p^x \rangle$ with addition and the powering function $x \mapsto p^x$, proved to be decidable by Semenov [Sem79], using quantifier elimination.

Second, classical Rabin-Scott automata model iterative intra-procedural program executions. In order to capture inter-procedural executions, with unbounded stack usage, one must consider pushdown automata instead. However, even with a stack, the pushdown automata model is not expressive enough, for the following reason: during a procedure call, values from very large or infinite domains (such as machine or mathematical integers) are pushed onto the stack. The finite stack alphabet of a pushdown automaton is thus not enough to capture the set of stack frames of a recursive program.

Finally, the structures recognized by finite automata are typically limited to trees, whereas programs can build more complex graph structures using pointers and dynamic memory allocation. An interesting observation is that the heap structures created by programs working on recursively linked data types (e.g. singly- and doubly-linked lists, trees with parent pointers and linked leaves, etc.) can, in general, be viewed as families of graphs with bounded treewidth. Recent work of Courcelle [Cou90] shows that satisfiability (and validity) of MSO is decidable on classes of graphs with bounded treewidth, by reduction to the satisfiability of MSO on trees [Rab68], thus motivating attempts of using MSO on graphs for the verification of programs with dynamic memory.

In this thesis, we present several theoretical and practical results on program verification, the main purpose being that of providing cost-efficient solutions to problems that almost always belong to undecidable classes. We appeal to logic and automata theory as they provide essentially the mechanisms to problem solving that are needed for program verification. In this respect, we investigate:

- logics for reasoning about infinite sets of program configurations, involving infinite data domains (array logics) and complex recursive data structures (separation logic), and
- automata extended with integer variables (counter machines), possibly with unbounded stacks (recursive integer programs).

We devoted special attention to the connection between logic and automata theory, by using counter machines and tree automata as effective decision procedures for array logics and separation logic, respectively. In this respect, we identified new decidable logical fragments and classes of automata and studied the complexity of their decision problems, such as e.g. validity, reachability and termination, respectively. Most of these theoretical results have been implemented within prototype tools used to carry out experimental evaluations of their practical efficiency.

Chapter 2

Summary of the Results

This chapter gives a summary of the main results, which mirrors the structure of this document. Most of the material in this thesis relies on journal publications, that are extended versions of previous conference papers. Thorough journal reviewing led to the discovery and correction of several errors that have, unfortunately, slipped through the more lightweight conference reviewing process. Each chapter also presents a number of open problems related to the corresponding developments.

Section 2.1 presents *flat counter machines*, a class of automata extended with integer variables, whose reachability and termination problems are shown to be decidable in nondeterministic polynomial time. These results are the core of a tool for the verification of integer programs, based on *acceleration* (FLATA [KIB09]). Chapter 4 covers this topic in detail. A related result on the solvability of a class of non-linear Diophantine systems is given in Chapter 3.

Section 2.2 discusses *recursive counter machines*, an extension of the model of counter machines, with unbounded local variables, parameters and return values. We establish complexity bounds for the reachability problems of recursive counter machines, in some restricted cases. These ideas inspired an implementation (within the FLATA tool [KIB09]). Chapter 5 covers these topics in detail.

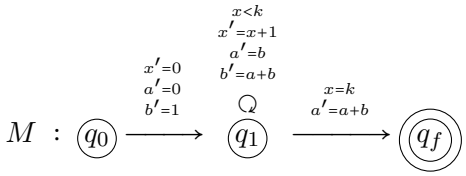
Section 2.3 describes several applications of counter machines to the verification of programs with pointer structures and integer arrays. We define decidable classes of programs with *dynamically allocated recursive pointer structures* and decidable *logics for reasoning about arrays of integers*, by reduction to the reachability and termination problems for flat counter machines. These techniques were implemented by a verifier for programs

with lists (L2CA [BIP]) and a solver for a logic of integer arrays (integrated within the FLATA tool [KIB09]). This topic is covered in detail by Chapter 6. A related result on the decidability of a fragment of the first-order integer arithmetic with addition and divisibility is given in Chapter 3.

Section 2.4 presents decidability and complexity results on *Separation Logic* (SL) [IO01, Rey02], an expressive logical framework for reasoning about programs with mutable heaps and recursive linked data structures. First, we show that SL with *inductive definitions* becomes decidable, under several natural restrictions on the syntax of the inductive rules. This is the most general decidability result on SL with inductive definitions, interpreted over unrestricted heaps, known so far. Second, we identified a fragment of this logic, for which the validity of entailments can be reduced to language inclusion between two tree automata, in polynomial time. This method was implemented by a solver for SL with user-defined inductive definitions (SLIDE [IRV]). These results are described in Chapter 7.

2.1 Flat Counter Machines

A *counter machine* (CM) is a finite state automaton, whose input alphabet consists usually of one letter[1], equipped with a finite set of variables (counters) that take integer values. For example, let us consider the following machine M computing the Fibonacci sequence F_n up to a given index $k > 0$:



For each transition rule, a', b' and x' denote the updated (next) values of a, b and x, respectively. A configuration of M is a pair (q, ν), where $q \in \{q_0, q_1, q_f\}$ is a *control location* and $\nu : \{a, b, x\} \to \mathbb{Z}$ is an integer valuation of the counters. The sequence of configurations below is an execution of M:

$$\begin{matrix} a \\ b \\ x \end{matrix} \begin{pmatrix} & ? \\ q_0, & ? \\ & ? \end{pmatrix} \Longrightarrow \begin{pmatrix} & 0 \\ q_1, & 1 \\ & 0 \end{pmatrix} \Longrightarrow \begin{pmatrix} & 1 \\ q_1, & 1 \\ & 1 \end{pmatrix} \Longrightarrow \begin{pmatrix} & 1 \\ q_1, & 2 \\ & 2 \end{pmatrix} \cdots \Longrightarrow \begin{pmatrix} & F_k \\ q_f, & ? \\ & k \end{pmatrix}$$

The values of a, b and x are not specified initially and can be randomly chosen. Whenever the control is at location q_1 and $x = n$, we have that

[1] This restriction can be lifted when considering several different input events.

$a = F_n$ and $b = F_{n+1}$. The machine iterates the loop $\widehat{q_1}$ as long as $x < k$ and exits towards q_f with $x = k$. It is easy to see that M has one execution of length $k+1$, and in the final configuration (q_f, ν), we have that $\nu(a) = F_{\nu(x)}$.

Many verification conditions for programs reduce to one of the following decision problems:

REACHABILITY:	given a counter machine, does it have an execution ending in a certain (set of) configuration(s) ?
TERMINATION:	given a counter machine, are all its executions finite ?

Despite their apparent simplicity, counter machines have the same power as Turing machines, even when restricted to have two counters, which can only be incremented, decremented and tested for equality with zero [Min67]. Due to this strong result of Minsky, the above problems are undecidable, in general. Finding classes of counter machines for which these problems are decidable is important, in order to identify verification sub-tasks that can be solved automatically, by push-button methods. Below we give several examples of decidable classes of CM.

Vector addition systems with states (VASS) [May81, Kos82, Ler12] are counter machines in which the only updates are of the form $\mathbf{x}' = \mathbf{x} + \mathbf{v}_i$, where $\mathbf{x} = \langle x_1, \ldots, x_d \rangle$ is the tuple of variables, ranging over *positive integers* $\mathbb{N}^d$, and $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \mathbb{Z}^d$ is a finite set of integer vectors. The reachability problem for VASS has been shown to be EXPSPACE-hard by Lipton [Lip76b] and currently no matching upper bound has been found. On the other hand, the problems of coverability and boundedness for VASS are shown to be EXPSPACE-complete [Rac78]. Because the transition relation of VASS can be defined as a finite disjunction of difference bounds constraints, these counter machines are, in principle, not flat. However, when restricting the number of counters to two, Hopcroft and Pansiot [HP79] have shown that the set of reachable configurations of a VASS is semilinear, thus definable in Presburger arithmetic. Along this line, Leroux and Sutre [LS04] showed that it is possible to build a flat counter machine, with the same transitions as the original 2-counter VASS and same reachable set of configurations. A close analysis of their construction revealed that reachability of 2-counter VASS (mostly known as 2-dimensional VASS) is textscPspace-complete [BFG+14].

Reversal-bounded counter machines [Iba78] are those CM in which, during each execution, a counter may switch between non-increasing and non-decreasing modes for a number of times bounded by a fixed constant. The

reachability problem for reversal-bounded CM is decidable in nondeterministic polynomial time [GI81]. On the other hand, it is undecidable whether a CM is reversal-bounded or not [FS08], which limits the applicability of this model for practical program verification purposes.

Flat counter machines [Boi99] are restricted to control structures in which each location belongs to at most one elementary cycle. In other words, no execution can indefinitely interleave two different cycles as in $\lambda\mu^2\lambda^3\mu^4\lambda^5\ldots$. Consequently, an algorithm for reachability and/or termination must only compute the transitive closures of the relations on the elementary cycles of the CM. This method is known as *acceleration* [Boi99, FL02b].

However, flatness alone does not ensure decidability of the reachability or termination problems[2]. To achieve decidability, one must further constrain the class of arithmetic relations that can label the transition rules within cycles. A first restriction was to consider cycles with guards defined by the additive theory of integers $\langle \mathbb{Z}, +, 0, \leq \rangle$, also known as Presburger arithmetic [Pre29], and deterministic updates $\mathbf{x}' = A\mathbf{x} + \mathbf{b}$, where the set of matrix powers $A, A^2, \ldots$ is finite. This *finite monoid condition*, ensures decidability of the reachability problem, by reduction to the satisfiability of a Presburger formula [Boi99, FL02b]. Although the reachability and termination problems are decidable for this class, the exact complexity bounds are still unknown for these problems.

2.1.0.1 Reachability Problems

In this joint work with Marius Bozga (VERIMAG) and Filip Konecny (PhD student at VERIMAG and Brno University of Technology, Czech Republic) we consider flat counter machines with cycles labeled by *octagonal constraints*, which are finite conjunctions of atomic constraints the form $\pm x \pm y \leq c$, where x and y are (possibly primed) variables and c is an integer constant. This class is incomparable with several previously studied CM models, because (i) it is not reversal-bounded, for instance a cycle with the relation $-1 \leq x - x' \leq 1$ allows arbitrarily long executions $x = 0, x = 1, x = 0, \ldots$ in which the counter switches between increasing and decreasing modes, and (ii) it allows nondeterminism, thus it cannot be represented by a single affine

[2] For instance, a famous open problem due to Skolem (see [OW12] for a description) can be expressed as the termination of a single cycle program with an affine update.

update. Our first main result for this class is stated below:

Theorem ([BIK14b, BIK13]). *The class of reachability problems for flat counter machines with cycles labeled by octagonal constraints is* NP*-complete.*

This result was achieved in several steps. First, we considered the simpler class of *difference bounds constraints*, that are finite conjunctions of atomic constraints $x - y \leq c$, where x and y are possibly primed variables and c is an integer constant. Comon and Jurski [CJ98] showed that the transitive closure $R^+ = \bigcup_{n=1}^{\infty} R^n$ of a relation[3] definable by a difference bounds constraint can be expressed by a formula in Presburger arithmetic. This reduces the reachability problem for flat CM with difference bounds constraints on cycles to the satisfiability of a formula in Presburger arithmetic [Pre29].

We first gave a simplified proof of their result [BIL09], based on the following observation. A relation R, defined by a difference bounds constraint, is represented by a weighted constraint graph $\mathcal{G}_R$, whose nodes are variables $\mathbf{x} \cup \mathbf{x}'$ and the weighted edges $x \xrightarrow{c} y$ correspond to atomic constraints $x - y \leq c$. Then each logical consequence $x - y' \leq c$ of R^n corresponds to a path of weight c, between extremal nodes $x^{(0)}$ and $y^{(n)}$, in the graph obtained by n adjacent copies of $\mathcal{G}_R$, with the nodes $\mathbf{x}$ and $\mathbf{x}'$ renamed to $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(i+1)}$, respectively, for $i \in [0, n-1]$. The set of such paths is regular, and one can effectively build a weighted automaton[4] $\mathcal{A}_R$ that recognizes all such paths. The problem of defining the transitive closure R^+ reduces to finding a Presburger formula $\phi(n, w)$ that defines the minimal weight w, among all paths of length n, between two designated states of the weighted automaton $\mathcal{A}_R$. This formula corresponds to the Parikh image of the weighted automaton, and can be computed in linear time, in the size of $\mathcal{A}_R$.

The initial acceleration algorithm, based on the construction of a weighted automaton, is, however, quite inefficient, because the size of this automaton is exponential in the number of variables, in the worst case. We bypassed this problem, and developed a very efficient acceleration algorithm, by noticing that the integer matrices which define the powers $R, R^2, \ldots$ of a difference bounds relation[5] form a *periodic sequence* $M_1, M_2, \ldots \in \mathbb{Z}^{2d \times 2d}$, where d is the number of variables (counters) that occur in the relation R. Intuitively, a sequence is periodic if all matrices situated at equal distance (period) in

[3] Let $R^1 = R$ and $R^{n+1} = R^n \circ R$, for all $n \geq 1$, where $\circ$ denotes composition of relations.

[4] A finite automaton with integer weights associated to transitions.

[5] The incidence matrices of the constraint graphs $\mathcal{G}_R, \mathcal{G}_{R^2}, \ldots$ for the relations $R, R^2, \ldots$

the sequence, beyond a certain threshold (prefix), differ by the same quantity (rate). Formally, there exist integers $b \geq 0$ and $c > 0$ and matrices $\Lambda_i, \ldots, \Lambda_{c-1}$ such that:

$$M_{b+(k+1)c+i} = M_{b+kc+i} + \Lambda_i, \forall k \geq 0 \ \ \forall i \in [0, c-1] \ . \tag{2.1}$$

It is then possible to characterize an infinite subsequence of powers $\left\{R^{b+kc}\right\}_{k\geq 0}$ simply by guessing the prefix b, the period c and computing the rate $\Lambda_0 = M_{b+c} - M_b$. From this, one computes the entire sequence $\{R^n\}_{n\geq b}$ by filling in the missing gaps $R^{b+kc+i} = R^{b+kc} \circ R^i$, $i \in [0, c-1]$. Also, the initial prefix $\{R^n\}_{n=1}^{b-1}$ is finite and can be computed using exponentiation by squaring.

In practice, the prefix and period of a difference bounds relation turn out to be rather small, resulting in a very fast acceleration algorithm [BIK10], that surpasses the weighted automata-based algorithm [BIL06, BIL09] by many orders of magnitude. For instance, this new algorithm can accelerate relations with several hundreds of variables in less than 10 seconds, while the same relations require several days with the old algorithm!

Moreover, the idea of periodic sequences can be generalized to octagonal relations, with relatively little effort. The main point is that octagonal relations can be represented by difference bounds matrices of size $4d \times 4d$, by encoding each variable x as the pair x_+, x_-. Intuitively, x_+ tracks the value of x, while x_- tracks the value $-x$. Then any octagonal constraint $x + y \leq c$ can be equivalently written as $x_+ - y_- \leq c$ or $y_+ - x_- \leq c$, with the implicit constraint $x_+ + x_- = 0$. In order to account for the latter condition, octagonal matrices must be *tightened* with respect to the domain of integers, by normalizing the constraints such as $2x \leq 1$ to the canonical form $2x \leq 0$. Also, tightening preserves periodicity, thus octagonal relations can be accelerated in the same way as difference bounds relations [BIK10].

Finally, we prove that the reachability problem for flat counter machines with octagonal cycles is in NP, by showing that both the prefix b and the period c of such sequences are of the order of $2^{\mathcal{O}(|R|)}$, where $|R|$ is the size of the binary encoding of R. Then the matrices M_b, M_{b+c} and the first rate Λ_0 of the sequence can be computed in polynomial time, by squaring, and the validity check (2.1) reduces to the satisfiability of quantifier-free Presburger formula, which is a known NP-complete problem [VSS05]. This provides a nondeterministic polynomial-time algorithm for the reachability problem [BIK14b, BIK13].

2.1.0.2 Termination Problems

The effective computation of transitive closures for octagonal relations is also key to establishing the decidability of the termination problem for flat counter machines with octagonal cycles. Let M be counter machine with variables $\mathbf{x}$, and $R \subseteq \mathbb{Z}^{\mathbf{x}} \times \mathbb{Z}^{\mathbf{x}}$ be the transition relation that labels a given cycle in M, where $\mathbb{Z}^{\mathbf{x}}$ is the set of all integer valuations of the variables in $\mathbf{x}$. The largest set of configurations from which there exists an infinite iteration of the cycle in M is called the *weakest non-terminating set*, and denoted as $wnt(R)$. This set is in fact the greatest fixpoint of the pre-image function $pre_R(X) = \{\mathbf{u} \in \mathbb{Z}^{\mathbf{x}} \mid \exists \mathbf{v} \in \mathbb{Z}^{\mathbf{x}} \,.\, (\mathbf{u}, \mathbf{v}) \in R\}$. Provided that the sequence $pre_{R^n}(\mathbb{Z}^{\mathbf{x}})$ stabilizes in a finite number of steps, we have:

$$wnt(R) = \bigcap_{n \geq 1} pre_{R^n}(\mathbb{Z}^{\mathbf{x}}) \ . \tag{2.2}$$

Since the sequence of powers $\{R^n\}_{n \geq 1}$ of an octagonal relation R can be characterized, using acceleration, by a formula $\widehat{R}(n, \mathbf{x}, \mathbf{x}')$ in Presburger arithmetic, the weakest non-terminating set is also definable in the same theory, by the formula:

$$wnt(R) \equiv \forall n \geq 1 \ \exists \mathbf{x}' \,.\, \widehat{R}(n, \mathbf{x}, \mathbf{x}') \ .$$

Then the termination problem for a flat counter machine with one cycle reduces to the validity of a Presburger formula with quantifier prefix $\exists^* \forall \exists$, a sufficient argument for decidability. The following theorem states the result, providing moreover tight complexity bounds:

Theorem ([BIK12, BIK14a]). *The class of termination problems for flat counter machines with octagonal cycles is in* P.

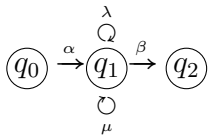
The polynomial upper bound is obtained by proving that, if the sequence $\{pre_{R^n}\}_{n \geq 1}$ stabilizes, then it stabilizes in at most $2^{k \cdot |R|}$ steps, for a computable constant $k > 0$. The stabilization condition can thus be decided in polynomial time, by computing the powers $\{R^n\}_{n \leq k \cdot |R|}$ by squaring.

2.1.0.3 Verification by Acceleration

The acceleration of octagonal constraints is at the core of a tool (FLATA [KIB09]) for the verification of (not necessarily flat) counter machines. The semi-algorithm implemented in FLATA attempts to compute a summary

(input/output) relation of a counter machine by eliminating all but the initial and the final control location[6]. Computing summaries is, in fact, essential for a *compositional* verification method. Each component of a counter machine needs, in fact, to be analyzed only once, its computed summary being used subsequently to represent the effect of any pair of transitions leading to and from that particular component.

The summary computation performs the following transformation, until a fixpoint is reached and no more states can be eliminated. Each subgraph of a counter machine of the form:



is replaced by an edge $q_0 \xrightarrow{\alpha \circ [\lambda^\dagger \circ \mu^\dagger \circ \cdots \circ \lambda^\dagger \circ \mu^\dagger] \circ \beta} q_2$, where $\lambda^\dagger \circ \mu^\dagger \circ \cdots \circ \lambda^\dagger \circ \mu^\dagger$ is a finite unfolding of the nested cycle on q_1, $R^\dagger = R^*$ is the reflexive and transitive closure, if R is octagonal, and $R^\dagger = R$, otherwise. Intuitively, we attempt to accelerate a disjunctive relation $\lambda \cup \mu$ by first trying to accelerate λ and μ separately as $\lambda^\dagger$ and $\mu^\dagger$, respectively, and then composing increasingly larger interleavings of $\lambda^\dagger$ and $\mu^\dagger$, thus obtaining increasingly better under-approximations of the reflexive and transitive closure of the cycle $(\lambda \cup \mu)^*$.

In many practical cases, this sequence of under-approximations converges and the acceleration yields a precise answer. Otherwise, the tool gives up after a given threshold, yielding an under-approximation (a stronger summary relation) that can still be useful to providing a positive answer to the reachability question (yet, we might fail to report a negative answer).

The FLATA tool [KIB09] uses a compositional semi-algorithm to check reachability and termination properties of unrestricted counter machines.

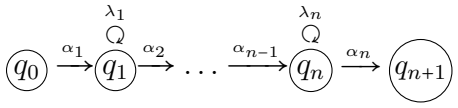
2.1.0.4 Predicate Abstraction with Interpolants

Another application of acceleration for octagonal constraints is predicate abstraction model checking with *interpolant-based abstraction refinement* [McM06]. Essentially, a predicate abstraction model checker associates a

[6]This procedure is similar to the state elimination algorithm that builds a regular expression defining the language of a finite-state automaton.

set of predicates (quantifier-free arithmetic formulae describing relations between variables) with each control location of a CM and builds an abstract model by exploring the boolean assignments of the predicates that are sound over-approximations of the set of reachable configurations. When a path $q_0 \xrightarrow{\alpha_1} q_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} q_{err}$ from an initial to a designated error location is discovered in the abstract model, it is checked whether this corresponds to a real counterexample, by automatically deciding the satisfiability of the path constraint $\alpha_1(\mathbf{x}^0, \mathbf{x}^1) \wedge \alpha_2(\mathbf{x}^1, \mathbf{x}^2) \wedge \ldots \wedge \alpha_n(\mathbf{x}^{n-1}, \mathbf{x}^n)$. If the path constraint is unsatisfiable, the solver used to check the satisfiability returns an *interpolant*, which is a sequence of predicates $I_0(\mathbf{x}), \ldots, I_n(\mathbf{x})$ such that $I_0 = \top$, $I_n = \bot$ and $I_i(\mathbf{x}) \wedge \alpha_i(\mathbf{x}, \mathbf{x}') \rightarrow I_{i+1}(\mathbf{x}')$, for all $i \in [0, n-1]$. These predicates are added to the existing domain in order to exclude the particular spurious counterexample path from future searches.

A typical problem of predicate abstraction is divergence in the form of longer and longer spurious counterexamples that follow the same pattern:



In order to converge, the solver must be capable of finding interpolants that are also inductive with respect to the cycles $\lambda_1, \ldots, \lambda_n$, i.e. $I_i(\mathbf{x}) \wedge \lambda_i(\mathbf{x}, \mathbf{x}') \rightarrow I_i(\mathbf{x}')$, for all $i \in [1, n]$. But this is not easy for an interpolating solver whose knowledge is always limited to one particular counterexample path, with no insight from the control structure (cycles) of the CM.

This is where acceleration of octagonal relation comes to help. By computing the reflexive and transitive closures of the cycles $\lambda_1^*, \ldots, \lambda_n^*$ of a given path scheme, we obtain a meta-trace $\alpha_1 \circ \lambda_1^* \circ \ldots \circ \alpha_{n-1} \circ \lambda_n^* \circ \alpha_n$. If the path constraint for the meta-trace is satisfiable we have found a real counterexample. Otherwise, we get the interpolants $\top, I_1', I_1'', \ldots, I_n', I_n'', \bot$. It is not hard to show that $\top, post(I_1', \lambda_1^*), \ldots, post(I_n', \lambda_n^*), \bot$ is a sequence of inductive interpolants for the above path scheme.

This idea was implemented in the ELDARICA [HR] model checker, developed jointly at Ecole Polytechnique Fédérale de Lausanne (Switzerland) and Uppsala University (Sweden). Acceleration provides a clear improvement in the convergence rate of the tool, on many practical examples [HIK+12].

2.1.0.5 Comparing Different Verification Techniques

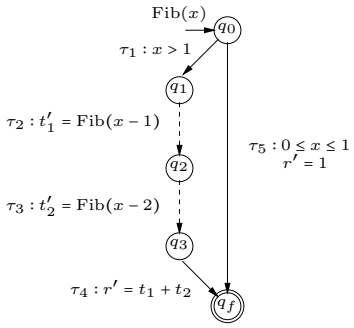
The comparison between acceleration (FAST [BFP], FLATA [KIB09]), predicate abstraction (ELDARICA [HR]) and abstract interpretation (ASPIC [Gon])

tools led to the idea of building a common benchmark library on which different approaches can be evaluated. The main effort here was the definition of a common language, called *Numerical Transition Systems* (NTS), that subsumes the input languages of existing tools, and allows the specification of both reachability and termination properties. We provide a clean definition of the language (abstract syntax and formal semantics), together with several open-source front- and back-ends [IKB].

Currently the NTS benchmark library [IKB] has over 200 benchmarks obtained by automatic translation from C programs (e.g. device drivers, network protocols), VHDL specifications of hardware, verification conditions for array logics (Chapter 6), etc. The results of a preliminary evaluation carried out on several tools (FAST [BFP], FLATA [KIB09], ASPIC [Gon] and ELDARICA [HR]) are reported in [HKG+12, HIK+12].

2.2 Recursive Counter Machines

Procedures and recursion are fundamental programming principles allowing to break a problem into smaller pieces, implemented by simple chunks of code. Consider, for instance, the following recursive counter machine that computes the Fibonacci sequence (a).



(a)

$$
\begin{array}{lcll}
\mathbf{p}_1 & : & Q_0 & \to \quad \tau_1 \cdot Q_1 \\
\mathbf{p}_2 & : & Q_0 & \to \quad \tau_5 \\
\mathbf{p}_3 & : & Q_1 & \to \quad \langle\!\langle \tau_2 \cdot Q_0 \cdot \tau_2 \rangle\!\rangle \cdot Q_2 \\
\mathbf{p}_4 & : & Q_2 & \to \quad \langle\!\langle \tau_3 \cdot Q_0 \cdot \tau_3 \rangle\!\rangle \cdot Q_3 \\
\mathbf{p}_5 & : & Q_3 & \to \quad \tau_4
\end{array}
$$

(b)

In this model, a program $\mathcal{P}$ is a set of counter machines $M_1, \ldots, M_n$, each with a list of input (parameters) and output (return) variables. In addition, we consider that a machine can have *call-return transitions* to other machines. A program is *recursive* if the call-graph[7] is cyclic.

[7] The graph whose nodes are $M_1, \ldots, M_n$ and there is an edge $M_i \to M_j$ iff M_i has a call-return transition to M_j.

The execution of a recursive program is a sequence of configurations consisting of a control location and a stack of integer valuations. For instance, the call-return transition $q_1 \xrightarrow{t_1'=\mathrm{Fib}(x-1)} q_2$ above is executed by saving the values of the local variables x, t_1, t_2 on the stack and transferring the control to q_0, with the updated value $x' = x - 1$. Upon return from the call, the values of x, t_1, t_2 are restored from the stack, and the value of the return variable r is copied into t_1.

The set of *interprocedurally valid execution paths* (IVP) of the program must meet the condition that the number of calls to a machine M equals the number of returns from M, for each call-return transition rule involving M, i.e. the execution starts and ends with an empty stack. If one does not consider the guards on the transitions, the IVPs of a recursive program form a context-free language. This observation inspired two orthogonal approaches for inter-procedural verification [SP81], which model the program either as a context-free grammar, or a pushdown automaton. The pushdown model faces the problem that the set of stack entries consists of unbounded integer valuations, which cannot be captured by a finite stack alphabet.

In turn, we model the program using context free grammars (Figure (b) above), where control locations are represented by nonterminals, and transitions by terminal symbols. In particular, each call-return transition τ is represented by a pair of terminals $\langle\!\langle\tau$ and $\tau\rangle\!\rangle$. The production rules of the grammar model the control flow of the program. For instance, the transition $\tau_2 : q_1 \xrightarrow{t_1'=\mathrm{Fib}(x-1)} q_2$ of the program (a) is modeled by the production $Q_1 \to \langle\!\langle\tau_2 \cdot Q_0 \cdot \tau_2\rangle\!\rangle \cdot Q_2$ of the grammar (b). We represent the guards and the updates of the integer variables of the program by mapping each terminal to an integer relation, as shown below.

$$\begin{array}{llll} \tau_1 \mapsto x > 1 & \tau_4 \mapsto r' = t_1 + t_2 & \tau_5 \mapsto 0 \le x \le 1 \wedge r' = 1 \\ \langle\!\langle\tau_2 \mapsto x' = x - 1 & \phi_{\tau_2} \equiv x' = x \wedge t_2' = t_2 & \tau_2\rangle\!\rangle \mapsto t_1' = r \\ \langle\!\langle\tau_3 \mapsto x' = x - 2 & \phi_{\tau_3} \equiv x' = x \wedge t_1' = t_1 & \tau_3\rangle\!\rangle \mapsto t_2' = r \end{array}$$

In addition, each call-return transition τ has a *local frame condition* ϕ_τ which copies the values of those local variables, not updated by the return, across the call.

The semantics of a program is given by the relation between the input (parameters) and output (return) variables, which is the union, over all interprocedurally valid program executions, of the relations corresponding to each execution. Formally, if $G_{\mathcal{P}}$ is the grammar corresponding to a program $\mathcal{P}$ and Q is the nonterminal corresponding to the initial location, we define the semantic relation $[\![\mathcal{P}]\!] = \bigcup_{w \in L_Q(G_{\mathcal{P}})} [\![w]\!]$, where $L_Q(G_{\mathcal{P}})$ is the language

of the grammar $G_{\mathcal{P}}$ with axiom Q, and the semantic relation of a word w is the composition of the relations labeling the terminal symbols on w, intersected with the frame conditions on the matching call/return positions. For instance the semantics of the word $w = \tau_1 \langle\!\langle \tau_2 \tau_5 \tau_2 \rangle\!\rangle \langle\!\langle \tau_3 \tau_5 \tau_3 \rangle\!\rangle \tau_4$ is:

$$[\![w]\!] = \rho_{\tau_1} \circ \left((\rho_{\langle\!\langle \tau_2} \circ \rho_{\tau_5} \circ \rho_{\tau_2 \rangle\!\rangle}) \cap \phi_{\tau_2} \right) \circ \left((\rho_{\langle\!\langle \tau_3} \circ \rho_{\tau_5} \circ \rho_{\tau_3 \rangle\!\rangle}) \cap \phi_{\tau_3} \right) \circ \rho_{\tau_4}$$

where ρ_τ denotes the relation associated with the symbol τ, as above.

Due to the undecidability of the reachability problems for non-recursive counter machines, the summary of a recursive program is not definable in a decidable theory, such as Presburger arithmetic. Moreover, the computation of summaries is challenging, in the presence of recursive procedures with integer parameters, return values, and local variables. While many analysis tools exist for non-recursive programs, only a few ones address the problem of recursion [RHS95, LAJ]. These tools use abstract interpretation [CC79] to give over-approximations of summaries. In this joint work with Pierre Ganty (IMDEA, Madrid), we propose a novel technique to generate arbitrarily precise underapproximations of summary relations.

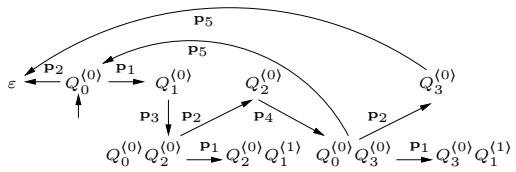
2.2.0.6 Index-bounded Underapproximation of Summaries

Our technique is based on the following idea. The control flow of procedural programs is captured precisely by the language of a context-free grammar. A k-index underapproximation of this language (where $k \geq 1$) is obtained by filtering out those derivations of the grammar that exceed a budget, called *index*, on the number (at most k) of occurrences of nonterminals occurring at each derivation step. As expected, the higher the index, the more complete the coverage of the underapproximation. From there we define the k-index summary relations of a program by considering the k-index underapproximation of its control flow. Our method then reduces the computation of k-index summary relations for a recursive program to the computation of summary relations for a non-recursive program, which is, in general, easier to compute because of the absence of recursion.

Given a (possibly recursive) program $\mathcal{P}$ and an index k, we define a source-to-source program transformation, based on the idea that the semantics of $\mathcal{P}$ can be also defined on the set of depth-first derivations[8] of $G_{\mathcal{P}}$. Then the set of depth-first derivations of index at most k can be represented by a finite automaton $A_{\mathcal{P}}^{\mathbf{df}(k)}$, whose states are ordered sequences of

[8] A derivation is said to be depth-first if it corresponds to a depth-first traversal of the associated parse tree.

nonterminals annotated with increasing priorities, of length at most k, and whose transitions are labeled with production rules. For example, the finite automaton $A_{\mathcal{P}}^{\mathbf{df}(2)}$ for the above Fibonacci program, is shown below.



This automaton recognizes infinitely many derivations, that are iterations of the sequence $Q_0 \xrightarrow{\mathbf{p}_1\mathbf{p}_3\mathbf{p}_2\mathbf{p}_4\mathbf{p}_5} \tau_1\langle\!\langle\tau_2\tau_5\tau_2\rangle\!\rangle\langle\!\langle\tau_3 Q_0\tau_3\rangle\!\rangle\tau_4$, corresponding to a set of program executions, with unbounded stack depth. By annotating the transition rules of the automaton $A_{\mathcal{P}}^{\mathbf{df}(k)}$ with suitable arithmetic constraints, that capture the semantics of the program, we obtain an under-approximation $[\![\mathcal{P}]\!]^{(k)} \subseteq [\![\mathcal{P}]\!]$ that captures all execution paths generated by derivations of index at most k. If the sequence of under-approximations $[\![\mathcal{P}]\!]^{(1)} \subseteq [\![\mathcal{P}]\!]^{(2)} \subseteq \ldots$ converges in finitely many steps, we obtain the precise summary of the program.

We have implemented, in the FLATA [KIB09] tool, a program transformation that returns, given $\mathcal{P}$ and $k > 0$, a non-recursive program with summary $[\![\mathcal{P}]\!]^{(k)}$, in a cost-effective way, by reusing, at each step k, the previous iterate of the sequence $[\![\mathcal{P}]\!]^{(k-1)}$. Several experiments show that this approach can compute precisely the summary of challenging recursive programs, such as Knuth's generalization of *McCarthy's 91 function* [Cow00].

The FLATA tool [KIB09] uses k-index under-approximation to compute precise summaries of recursive integer programs.

2.2.0.7 Acceleration of Bounded Execution Paths

Having defined the k-index under-approximation sequence of the summary semantics of a recursive program, a natural question to ask is *when does this sequence converge* ? We answered this question by considering *bounded context-free languages* [GS64]. These are languages generated by context-free grammars, which are, moreover, included in a regular pattern of the form $w_1^* \ldots w_n^*$, for some non-empty words $w_1, \ldots, w_n$. A typical example is the context-free language $\{\alpha^n\beta^n \mid n \geq 0\}$, which is included in $\alpha^*\beta^*$.

In some sense, bounded context-free languages extend the model of flat counter machines from the intra- to the inter-procedural setting. Given a recursive program $\mathcal{P}$ and a bounded pattern $\mathbf{b} = w_1^* \dots w_n^*$ over its transition rules, one can always compute a program $\mathcal{P}_\mathbf{b}$ whose executions are those executions of $\mathcal{P}$ of the form $\mathbf{b}$. This is useful for bug-finding, as increasingly large patterns cover more and more executions of a given program.

As Luker shows [Luk78], bounded context-free languages can be generated considering only derivations of index at most linear in the number of nonterminals[9]. Thus $[\![\mathcal{P}_\mathbf{b}]\!] = [\![\mathcal{P}_\mathbf{b}]\!]^{(k)}$ for an index $k = \mathcal{O}(|\mathcal{P}|)$, where $|\mathcal{P}|$ denotes the number of control locations (i.e. size of) $\mathcal{P}$.

The remaining question is related to the computability of the summary $[\![\mathcal{P}_\mathbf{b}]\!]^{(k)}$, for a given $k > 0$. Based on the previous result on the acceleration of octagonal relations (Section 2.1), we show that $[\![\mathcal{P}_\mathbf{b}]\!]^{(k)}$ is always computable, using acceleration, provided that all statements of $\mathcal{P}$ are defined by octagonal constraints. Moreover, we provide an upper bound on the complexity of the reachability problem for bounded recursive programs with octagonal guards and updates.

Theorem ([GI15a])**.** *The class of reachability problems for recursive programs with octagonal constraints and bounded set of interprocedurally valid execution paths is in* NEXPTIME*, with an* NP*-hard lower bound. Moreover, this class becomes* NP*-complete if the maximal derivation index is a constant, not part of the input.*

The main idea of the proof is to compute a bounded expression $\Gamma_\mathbf{b}$ over the transition rules of the finite automaton $A_\mathcal{P}^{\mathbf{df}(k)}$ and reduce the interprocedural reachability problem $[\![\mathcal{P}_\mathbf{b}]\!]^{(k)} = \varnothing$ to the intra-procedural reachability problem, for a bounded subset of the executions of the automaton $A_\mathcal{P}^{\mathbf{df}(k)}$, labeled with arithmetic constraints that capture the summary relation $[\![\mathcal{P}_\mathbf{b}]\!]^{(k)}$. Moreover, $\Gamma_\mathbf{b}$ is computable in time $|\mathcal{P}|^{\mathcal{O}(k)}$. Since $k = \mathcal{O}(|\mathcal{P}|)$ [Luk78], we obtain the NEXPTIME upper bound. Despite our best efforts, we did not close the complexity gap yet. However, if the index is fixed, the computation of $\Gamma_\mathbf{b}$ runs in polynomial time, which establishes the NP-completeness of the reachability problem, in this particular case.

[9] Luker's proof can be adapted to show that it is sufficient to consider $k = 2n$, where n is the number of nonterminals.

2.3 Program Verification with Counter Machines

The deployment of modern programming languages, such as C,C++ or Java, in industrial-scale software development is hindered by an important obstacle: guaranteeing the reliability of high-level programs requires for more powerful computer supported analysis and verification techniques than those currently available. Indeed, most programming languages used nowadays in industry exhibit a wealth of features which greatly complicate the task of checking correctness, for instance:

- *dynamic memory structure*: in order to achieve efficient memory management, real-life imperative programming languages use low-level mechanisms such as dynamic allocation, in-place pointer updates, or garbage collection. Moreover, the size of the memory can become arbitrarily large and its topology can vary during the execution of the program.
- *infinite data domains*: memory cells may contain data ranging over very large or infinite domains, such as integers, arrays of integers, etc.

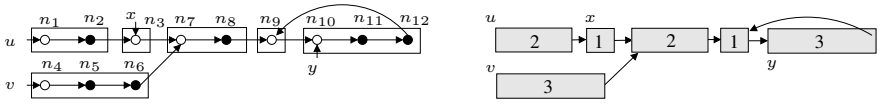
We applied existing methods for solving decision problems on counter machines to verification problems of programs with (i) arrays of integers, and (ii) dynamically linked recursive data structures, such as lists and trees. The basic idea is to reduce checking a safety/termination property of a program with pointers or arrays to a reachability/termination problem for a counter machine, by associating quantitative information with parts of the program's heap, such as the length of a list segment, or the size of a subtree. Sometimes, such as in the case of programs with singly-linked lists, the reduction is without loss of information, thus a yes/no answer obtained on the counter machine automatically holds on the original program. In other cases, such as programs with tree-shaped data structures, the reduction yields a counter machine that is a safe over-approximation of the program, which is useful, in principle, for purposes of certification (proof of correctness).

Based on the type (number of selector fields) of the memory cells allocated by the program, we distinguish between programs with (i) *lists* (one selector, possibly with sharing and circularities), and (ii) *trees* (two or more selectors, but no sharing or cycles). This difference has a crucial impact on the symbolic abstract domain chosen to encode possibly infinite sets of program configurations, and thus, on the verification method we propose.

2.3.0.8 Programs with Lists

If each cell in the heap may have at most one selector pointing to a different cell, the heap of a program can be viewed as a set of reversed trees, whose

edges are directed towards the root, possibly with a simple cycle at the root. Although the number of heap graphs is infinite, the number of *cut points*, i.e. nodes directly pointed to by a program variable, or by two other nodes, is bounded by the number of pointer variables of the program. A finite-range abstraction consists in mapping each list segment, between two cut points, into an abstract node, as follows:

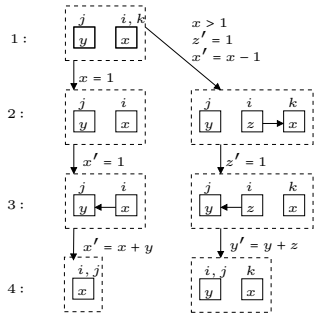


Here u, v, x and y are program variables and the cut points are n_1, n_3, n_4, n_7, n_9 and n_{10}. The corresponding *shape graph*, where each list segment between two cut points is represented by an abstract node, is given in Figure (b). It is not hard to prove that the number of shape graphs is bounded by n^n, where n is the number of program variables.

This abstraction is obviously not precise. In order to define a precise abstraction, we need to reason about the size of each list segment. This leads to the idea of using a counter machine, whose control states are shape graphs, and whose variables track the sizes of the list segments represented by the abstract heap nodes.

The translation of programs with singly-linked lists into counter machines is best explained by means of an example. Consider the sequence of pointer manipulations from Figure (a) below. In Figure (b) we show the sequence of transition rules of the corresponding counter machine:

1: k ← i.next
2: i.next ← j
3: j ← i
4: i ← k



(a) (b)

The input configuration consists of two disjoint list segments of length x and y, pointed to by the program variables i,k and j, respectively. The

first update k ← i.next may have two outcomes, depending on the length of the list segment pointed to by i. If this length is one ($x = 1$), the next field of the cell is null, thus k becomes a null pointer (line 2, left). In the other case ($x > 1$) the abstract node pointed to by i is split into a node of size 1, pointed to by i, and a node of size $x - 1$, pointed to by k (line 2, right).

If the program does not test the data values within the heap cells, i.e. it is *data-independent*, the counter machine abstraction is without loss of precision. Formally, we prove that the semantics of the counter machine obtained from a program with lists is a *bisimulation* of the semantics of the program [BBH+06, BBH+11]. Since bisimulation preserves safety, termination, and, more generally, temporal logic properties, any property proved or disproved on the counter machine carries on to the original program. This translation was implemented in the L2Ca tool, developed by Swann Perarnau (VERIMAG).

The L2Ca tool [BIP] produces counter machines that precisely simulate the behavior of programs with singly-linked lists.

The tight relation between programs with lists and counter machines can be exploited further, to reveal several (un-)decidability results concerning the reachability and termination problems for the class of data-independent programs with lists. Since, in general, a program with lists can simulate a 2-counter machine [Min67], the reachability and termination problems are undecidable, unless several restrictions are applied.

The first restriction is, naturally, flatness of the control structure of the program, i.e. absence of nested cycles, or conditional statements inside a cycle. However, the above translation scheme does not guarantee that a CM obtained from a flat program is flat. The problem is with the translation of statements such as k ← i.next, which test whether the length of the list segment pointed to by i is greater or equal to one. If such statements are used inside a program cycle, the resulting CM might not be flat.

We tackled this problem by restricting further the class of programs we consider, and excluding the selector update statements such as i.next ← j. That is, a program can only traverse the input heap, but not change its structure. Surprisingly, even with this restriction, the reachability and termination problems for flat programs with lists are undecidable. The source of undecidability lies, this time, in the complexity of the input data structure. We noticed that the *least common multiple* relation $x = [y, z]$ can be encoded by programs running on input structures with at least two (sepa-

rate) cycles. As a consequence, one can encode *Hilbert's Tenth Problem*[10] (HTP) [Mat70], as a safety property of a flat program with lists, without selector updates. The same method can be also used to reduce HTP to the termination problem for flat programs with lists.

We draw a sharp decidability boundary, by showing that, if the input heap is restricted to having at most one cycle, the reachability and termination become decidable [BI07]. To this end, we defined a different encoding than the one described in [BBH+06, BBH+11], and deal with the dereferencing statements k ← i.next in a way which preserves the flatness of the resulting counter machine.

Theorem ([BI07]). *The safety and termination problems are undecidable for flat programs without selector updates, running on heaps consisting of singly-linked lists with two or more cycles, and become decidable if we restrict the input heap to having at most one cycle.*

The latter decidability result is obtained by reduction to a class of counter machines with guards that are conjunctions of linear and *divisibility* constraints, of the form $f(\mathbf{x}) \mid g(\mathbf{x},\mathbf{y})$, and updates of the form $x' = y + c$, where f, g are linear terms, c is an integer constant, and, moreover, at most one linear term may occur on the left-hand side of a divisibility constraint $f(\mathbf{x}) \mid g(\mathbf{x},\mathbf{y})$. The transitive closures of the relations labeling the cycles of the flat CM are definable in a decidable fragment of integer arithmetic with addition and divisibility, which we defined a in previous work [BI05]. For self-containment reasons, this result is given in Chapter 3.

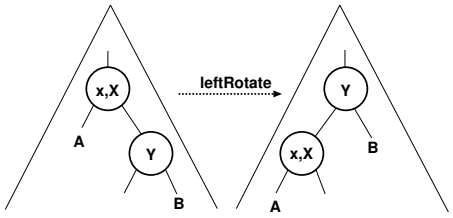
2.3.0.9 Programs with Trees

If the data types of several heap cells allocated by a program have two or more outgoing pointer fields, collapsing each list segment into an abstract node does not yield a finite-range abstraction. However, if each heap configuration consists of a set of (disjoint) trees, *tree automata* [CDG+05] are a natural finite representation of a potentially infinite set of heaps. For instance, tree automata are used as internal representation within program analysis tools such as ARTMC [RV] and FORESTER [HHR+] (developed at Brno University of Technology, Czech Republic). In general, the use of tree automata in program verification incurs two major problems:

[10] An algorithm that determines the existence of solutions for Diophantine systems.

- imperative programs perform destructive updates of selector fields, changing a tree-shaped data structure by temporarily introducing sharing of branches and/or cycles. This is the case of tree rotations, which are implemented as a finite sequence of selector updates introducing a cycle in the tree, and subsequently re-establishing the tree shape.
- tree automata represent regular sets of trees, which is not sufficient when one needs to reason in terms of balanced trees as in the case of AVL and RED-BLACK tree algorithms.

In order to overcome the first problem, we observe that most algorithms working on balanced trees use the following operations: (i) pointer assignments $\mathrm{x} \leftarrow \mathrm{null}$, $\mathrm{x} \leftarrow \mathrm{y}$, $\mathrm{x} \leftarrow \mathrm{y}.\{\mathrm{left}|\mathrm{right}|\mathrm{up}\}$, $\mathrm{x.data} \leftarrow d$, (ii) conditional statements $\mathrm{x} = \mathrm{null}, \mathrm{x} = \mathrm{y}, \mathrm{x.data} = d$, (iii) leaf insertion and subtree removal, and (iv) tree rotations (see above figure).



The second inconvenience is dealt with by introducing a novel class of tree automata, called *Tree Automata with Size Constraints* (TASC) [HIV06, HIV10]. TASC are tree automata whose actions are triggered by difference bounds constraints involving the sizes of the subtrees at the current node. The size of a tree is a numerical function defined inductively on the tree structure such as, for instance, the height, the maximum number of black nodes on all paths, etc. TASC recognize non-regular sets of tree languages, such as the AVL trees, the red-black trees, and, in general, sets of trees involving arithmetic reasoning about the lengths (depths) of various (possibly all) paths in the trees.

For instance, let us consider the ranked alphabet $\Sigma = \{red, black, null\}$, with arities $\#(red) = \#(black) = 2$ and $\#(null) = 0$, and size function $|red(t_1, t_2)| = \max(|t_1|, |t_2|)$, $|black(t_1, t_2)| = \max(|t_1|, |t_2|) + 1$ and $|null| = 1$, which counts the maximal number of black nodes from the root to a leaf. The bottom-up TASC recognizing the set of balanced red-black trees has two states q_b and q_r, where q_b is final, and the transition rules are:

$$null \rightarrow q_b \qquad black(q_{r/b}, q_{r/b}) \xrightarrow{|1|=|2|} q_b \qquad red(q_b, q_b) \xrightarrow{|1|=|2|} q_r$$

Intuitively, q_b is reached upon reading a black (null) node, and q_r indicates that the current node is red and its successors are black. Moreover, the size constraint $|1| = |2|$ on the transition rules requires that the number of black

nodes is the same on all paths from the current node to a leaf.

Theorem ([HIV06, HIV10]). *The class of TASC is closed under the operations of union, intersection, and complement, and, moreover, the emptiness problem is decidable.*

We thus obtain a class of automata which are an interesting theoretical contribution by itself. Moreover, the semantics of the programs performing tree updates (node coloring, rotations, leaf nodes appending/removal) can be effectively represented as changes on the structure of the automata.

The verification approach for tree manipulating programs requires the user to provide pre-, post-conditions and loop invariants. The verification problem reduces to checking the validity of Hoare triples {P}C{Q}, where P and Q are sets of configurations represented by TASC, and C is a loop-free program fragment. This is equivalent to checking language inclusion between two TASC, which is decidable, in the light of the above theorem.

Going beyond verification of safety properties, we propose a method for checking universal termination of programs manipulating tree data structures [HIRV07]. Namely, we are interested in proving that such a program terminates for any input tree out of a given set described as an infinite regular tree language over a finite alphabet. We represent a given program as a control flow graph whose nodes are annotated with tree automata, that over-approximate the sets of reachable configurations, computed using ARTMC [RV]. From the annotated control flow graph, we build a counter machine that simulates the program. The variables of the CM keep track of different measures within the working tree: the distances from the root to nodes pointed to by certain variables, the sizes of the subtrees below such nodes, and the numbers of nodes with a certain data value (we consider a finite data domain). Termination of the CM is analyzed by existing tools [BMS05, CPR06].

If the CM is shown to terminate, termination of the program is proved too. Otherwise, the CM termination analyzer outputs a lasso-shaped counterexample. This counterexample is translated back into a sequence of program instructions and analyzed for spuriousness on the program. If the counterexample is found to be real, the procedure reports non-termination. Otherwise, the program control flow graph is refined by splitting some of its nodes (actually, the sets of program configurations associated with certain control locations), and the abstract-check-refine loop is reiterated.

In a further generalization of this method [IR09, IR13], we consider programs working on an infinite data domain $\mathcal{D}$ equipped with an arbitrary number of well-founded partial orders $\preceq_1, \ldots, \preceq_n$, such that, for any data transformation $\Rightarrow \subseteq \mathcal{D} \times \mathcal{D}$, induced by a program statement, the emptiness problem $\Rightarrow \cap \preceq_i = \emptyset$ is decidable, for all $i \in [1, n]$. For instance, if $\mathcal{D}$ is the set of terms (trees) over a finite ranked alphabet, then $\preceq_i$ is a classical well-founded ordering on terms, e.g. Recursive Path Ordering, Knuth-Bendix Ordering, etc. Given the data domain $\langle \mathcal{D}, \preceq_1, \ldots, \preceq_n \rangle$, the program is automatically abstracted into a Büchi automaton, recognizing infinite words over the alphabet of tuples $\{\preceq, =, \bowtie\}^n$. Intuitively, for a symbol, $\sigma \in \{\preceq, =, \bowtie\}^n$, if σ_i is $\preceq$ $(=)$ then $\preceq_i$ $(\preceq_i \cap \succeq_i)$ holds, whereas $\bowtie$ means "don't know".

A counterexample is an infinite path of the form $\sigma\lambda^\omega$ (called lasso) whose cycle λ cannot be proved to terminate, based on the abstract labeling with relations. Spuriousness of the lasso is checked by a domain-specific procedure. If the lasso is found to be spurious, we eliminate it by intersecting the Büchi automaton representing the program, with the *weak deterministic Büchi automaton*[11] representing the lasso, and check the new model for existence of infinite counterexamples.

We experimented this method by proving termination of non-trivial algorithms that manipulate tree-like, and even more complex data structures, such as the Deutsch-Schorr-Waite tree traversal, the Red-black rebalancing after insertion/deletion, or insertion/deletion in a tree with *linked leaves*. Most of these algorithms could not be automatically verified before, their correctness being the object of heavy textbook handwritten proofs.

2.3.0.10 Programs with Integer Arrays

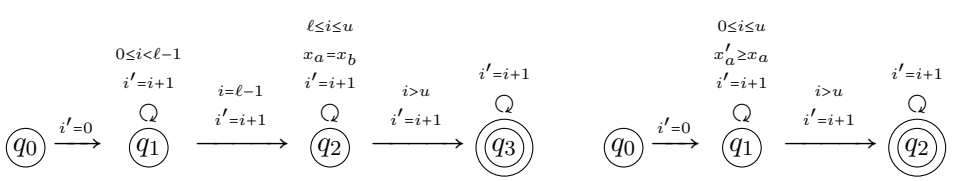
Arrays are fundamental data structures used by all modern imperative programming languages. The verification of programs that manipulate arrays requires expressive logics, able to capture properties such as bounded array equality, sortedness and, in general, arithmetic properties of array entries situated at arbitrarily large distances. Moreover, in order to be effective, one needs a decidable logic, that is supported by an automated decision procedure, running within reasonable complexity bounds.

Properties typically expressed about arrays in a program are existentially quantified boolean combinations of formulae $\forall i_1 \ldots \forall i_k \ . \ g(i_1, \ldots, i_k) \rightarrow v(i_1, \ldots, i_k, a_1, \ldots, a_n)$, where $i_1, \ldots, i_k$ are index and $a_1, \ldots, a_n$ are array variables, occurring in array read terms of the form $a[i]$. For instance

[11] Each SCC of a weak Büchi automaton contains either final or non-final states, but not both. A deterministic weak Büchi automaton can be complemented in linear time.

bounded array equality can be written as $\forall i \,.\, \ell \le i \le u \to a[i] = b[i]$, and sortedness as $\forall i \forall j \,.\, i < j \to a[i] \le b[j]$, or, equivalently, as $\forall i \,.\, a[i] \le a[i+1]$.

We start from the basic observation that the models of such universally quantified array properties are second-order valuations associating each array symbol a a finite word from $\mathcal{V}_a^*$, where $\mathcal{V}_a$ is the value sort of a. For integer arrays ($\mathcal{V}_a \equiv \mathbb{Z}$), these words, over the infinite alphabet of integers, can be encoded by the executions of counter machines that have a counter x_a for each array symbol a:



We defined two logics, called *Singly Indexed Logic* (SIL) [HIV08a] and *Logic of Integer Arrays* (LIA) [HIV08b], for reasoning about integer arrays. The decidability of these logics is established by proving a connection between the logic and flat counter machines with octagonal constraints: for each array logic formula φ, there exists a flat counter machine M_φ whose runs are in one-to-one correspondence with the set of models (array valuations) of φ. Thus, φ is satisfiable if and only if M_φ is not empty, and the latter problem is decidable, as previously discussed (see Chapter 4).

The first logic, SIL [HIV08a], is the fragment of existentially quantified boolean combinations of Presburger constraints on bound variables and *array properties* $\forall i \,.\, g(i, \mathbf{l}) \to v(i, \mathbf{a}, \mathbf{l})$, where $\mathbf{a}$ is a set of *array variables*, $\mathbf{l}$ is a set of existentially quantified *bound variables*, and i is an *index variables*, the only variable occurring under the scope of a universal quantifier, and:

- the guard g is a boolean combination of bound ($\pm i \pm \ell \sim n$) and modulo ($i \equiv_m p$) constraints on index variables, and
- the value constraint v is an octagonal constraint consisting of a finite conjunction of terms of the form $a[i(+1)] \sim \ell$, $\pm a[i] \pm i \sim n$ and $\pm a[i] \pm b[i(+1)] \sim n$,

where $\sim \in \{\le, \ge\}$, a and b are array variables, ℓ is a bound variable, a length term $|a|$, or an integer constant, $n, m, p \in \mathbb{N}$ are positive integer constants, with $m > 0$. Each array property can be converted into a *deterministic*[12] flat counter machine, that encodes its set of models. Since deterministic flat CM can be complemented, by reversing final and non-final states, it turns

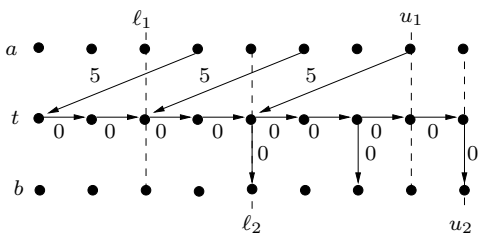
[12] Determinism is ensured by the handling of the index counter within the guards.

out that the construction of a counter machine for an entire SIL formula can be done compositionally, by induction of the structure of this formula.

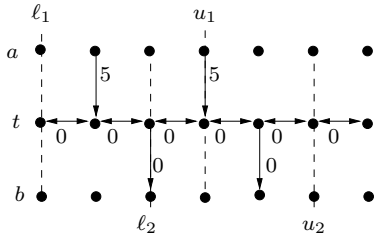
Theorem ([HIV08a]). *The satisfiability problem is decidable for the logic* SIL.

Second, we work around the single index restriction from the definition of SIL, and consider more general array properties of the form $\forall \mathbf{i} \,.\, g(\mathbf{i}, \mathbf{l}) \rightarrow v(\mathbf{i}, \mathbf{a}, \mathbf{l})$, where $\mathbf{i}$ is a set of universally quantified index variables that, moreover, can occur within difference bounds constraints of the form $i - j \leq c$ and conjunctions thereof, for some $i, j \in \mathbf{i}$ and $c \in \mathbb{Z}$. We also generalize the syntax of value constraints to include comparisons $\pm a[i+n] \pm b[j+m] \sim p$, of array values within a constant window. The result is the logic LIA [HIV08b].

The decidability of LIA goes by reduction to flat counter machines. The use of more than two universally quantified variables induces non-local constraints, involving array positions at arbitrarily large distance. The idea of the reduction is that these non-local constraints are obtained as transitive closures of a larger set of local constraints, that occur within a constant size window. In fact, we eliminate the array properties involving more than one index variable, by introducing additional array variables, and using the transitivity of difference constraints, as shown in the examples below:



(a) $\forall i \forall j \,.\, \ell_1 \leq i \leq u_1 \wedge \ell_2 \leq j \leq u_2 \wedge i - j \leq 3 \wedge i \equiv_2 0 \wedge j \equiv_2 1 \rightarrow a[i] - b[j] \leq 5$



(b) $\forall i \forall j \,.\, \ell_1 \leq i \leq u_1 \wedge \ell_2 \leq j \leq u_2 \wedge i \equiv_2 0 \wedge j \equiv_2 1 \rightarrow a[i] - b[j] \leq 5$

For instance, the constraint $a[\ell_1 + 1] - b[\ell_2] \leq 5$, implied by the array property (a), is induced via the additional array variable t and the constraints $a[\ell_1 + 1] - t[\ell_1 - 2] \leq 5$, $t[\ell_2] - b[\ell_2] \leq 0$ and $\forall i \,.\, t[i] \leq t[i + 1]$. In a similar way, the constraint $a[\ell_1 + 1] - b[\ell_2 + 2] \leq 5$, implied by the array property (b), is captured via the additional array t, with the constraints $a[\ell_1 + 1] - t[\ell_1 + 1] \leq 5$, $t[\ell_2 + 2] - b[\ell_2 + 2] \leq 0$ and $\forall i \,.\, t[i] = t[i + 1]$.

The reduction from LIA to counter machines requires also that the negated array properties, with more than one universally quantified index variable,

be eliminated beforehand. This observation leads to the following:

Theorem ([HIV08a]). *The satisfiability problem is decidable for the logic* LIA.

Based on this work, we implemented an entailment solver for SIL, on top of the FLATA [KIB09] tool for the verification of counter machines. This solver has been used to perform verification of programs manipulating integer arrays, and automatically validate several procedures such as *array sorting*, *insertion*, and *rotation* [BHI+09].

2.4 Separation Logic

Separation Logic (SL) [Rey02] is a logical framework for describing dynamically allocated mutable data structures generated by programs that use pointers and low-level memory allocation primitives. The logics in this framework are used by a number of academic (SPACE INVADER [BCC+07], SLEEK [NC08]), and industrial (INFER [CD11]) tools for program verification. The main reason for choosing to work within the SL framework is the ability to provide compositional proofs, based on the principle of *local reasoning*: analyzing different parts of the program (e.g. functions, threads), that work on *disjoint parts of the global heap*, and combining the analysis results a-posteriori.

The main ingredients of SL are (i) the *separating conjunction* $\varphi * \psi$, which asserts that φ and ψ hold for separate portions of the memory (heap), and (ii) the *frame rule*, that exploits separation to provide modular reasoning about programs. Consider, for instance the following memory configuration, in which two different cells are pointed to by the program variables x and y. The x cell has an outgoing selector field to the y cell, and viceversa:

$\mathsf{x} \mapsto \mathsf{y} * \mathsf{y} \mapsto \mathsf{x}$

The heap can be split into two disjoint parts, each containing exactly one cell, and described by an atomic proposition $\mathsf{x} \mapsto \mathsf{y}$ and $\mathsf{y} \mapsto \mathsf{x}$, respectively. Then the entire heap is described by the formula $\mathsf{x} \mapsto \mathsf{y} * \mathsf{y} \mapsto \mathsf{x}$, read x *points to* y *and separately* y *points to* x..

When reasoning about programs that manipulate data structures, it is crucial to have the ability of describing infinite sets of heaps, that are in-

stances of recursively defined data structures, such as singly- or doubly-linked lists, trees and such. This is achieved in SL by introducing *inductive definitions*. For instance, the inductive definition below defines a set of doubly-linked lists:

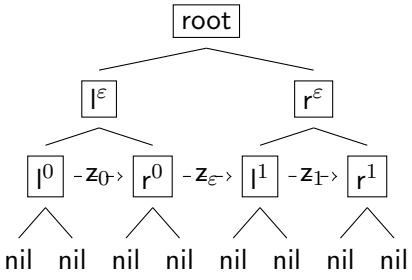
$$\begin{array}{lcll} \mathsf{DLL}(\mathsf{head},\mathsf{prev},\mathsf{tail}) & \equiv & \mathsf{tail} \mapsto (\mathsf{prev},\mathsf{null}) \wedge \mathsf{head} = \mathsf{tail} \vee & (\mathsf{r}_1) \\ & & \exists \mathsf{x} \;.\; \mathsf{head} \mapsto (\mathsf{prev},\mathsf{x}) * \mathsf{DLL}(\mathsf{x},\mathsf{head},\mathsf{tail}) & (\mathsf{r}_2) \end{array}$$

The first case (r_1) of the definition is the *base rule*, i.e. the heap consists of exactly one cell and head and tail are equal, while the second case (r_2) is the *inductive rule* which corresponds to the one-step unfolding of the definition. The separating conjunction here states that the cell pointed to by head is disjoint from the rest of the heap, which is, moreover, a $\mathsf{DLL}(\mathsf{x},\mathsf{head},\mathsf{tail})$. A complete n-step unfolding of the definition produces the heap structure:

$$\mathsf{prev} \leftarrow \underbrace{\boxed{\mathsf{head}}}_{\mathsf{r}_2} \leftrightarrows \underbrace{\boxed{\mathsf{x}^1}}_{\mathsf{r}_2} \leftrightarrows \underbrace{\boxed{\mathsf{x}^2}}_{\mathsf{r}_2} \leftrightarrows \cdots \leftrightarrows \underbrace{\boxed{\mathsf{x}^{n-2}}}_{\mathsf{r}_2} \leftrightarrows \underbrace{\boxed{\mathsf{tail}}}_{\mathsf{r}_1}$$

The following example shows the definition of a tree structure, whose leaves are linked in a list, in preorder.

$$\begin{array}{lcl} \mathsf{TLL}(\mathsf{root},\mathsf{ll},\mathsf{lr}) & \equiv & \mathsf{root} \mapsto (\mathsf{nil},\mathsf{nil},\mathsf{lr}) \wedge \mathsf{root} = \mathsf{ll} \vee \\ & & \exists \mathsf{l},\mathsf{r},\mathsf{z} \;.\; \mathsf{root} \mapsto (\mathsf{l},\mathsf{r},\mathsf{nil}) * \mathsf{TLL}(\mathsf{l},\mathsf{ll},\mathsf{z}) * \mathsf{TLL}(\mathsf{r},\mathsf{z},\mathsf{lr}) \end{array}$$



We show a possible unfolding of the structure described by the TLL definition. To understand how this structure is created, we annotate each existentially quantified variable with the position in the tree (ε for the root, 0 and 1 for the left and right children of the root) where this variable is introduced during the unfolding of the definition. The links between the leaves are introduced by the base rule, using the existentially quantified z variables introduced by the inductive rule. We annotate each edge between leaves with the variable responsible for its occurrence.

In general, one considers a system of inductive definitions of the form:

$$\left\{\mathsf{P}_\mathsf{i}(\mathsf{x}_{i,1},\ldots,\mathsf{x}_{i,n_i}) \equiv \bigvee_{j=1}^{m_i} \mathsf{r}_{ij}(\mathsf{x}_{i,1},\ldots,\mathsf{x}_{i,n_i})\right\}_{i=1}^{n}$$

where $\mathsf{P}_1, \ldots, \mathsf{P}_k$ are *predicates*, $\mathsf{x}_{i,1}, \ldots, \mathsf{x}_{i,n_i}$ are *parameters*, and the formulae r_{ij} are the rules of P_i. Concretely, a rule r_{ij} is of the form:

$$\exists \mathbf{z} \,.\, \Sigma * \mathsf{P}_{i_1}(\mathbf{y}_1) * \ldots * \mathsf{P}_{i_m}(\mathbf{y}_m) \;\wedge\; \Pi$$

where Σ is a finite set of points-to formulae of the form $\mathsf{x} \mapsto (\mathsf{y}_1, \ldots, \mathsf{y}_n)$, that describe single heap cells, joined by separating conjunctions, and $\mathsf{P}_{i_1}(\mathbf{y}_1), \ldots, \mathsf{P}_{i_m}(\mathbf{y}_m)$ is a (possibly empty) list of predicates, and Π is a finite conjunction of equalities and disequalities between variables.

For a predicate P, the set of models $[\![\mathsf{P}]\!]$ is the least set of heap structures that satisfies the inductive definition of P. Technically speaking, this is the least fixpoint of a monotonic and continuous function that mirrors the definitions in the system. Given two predicates P and Q, the *entailment problem* $\mathsf{P} \models_{\mathsf{SL}} \mathsf{Q}$ asks whether $[\![\mathsf{P}]\!] \subseteq [\![\mathsf{Q}]\!]$. Incidentally, we also consider the *satisfiability problem*: given P, is it the case that $[\![\mathsf{P}]\!] = \varnothing$?

The entailment problem for SL with inductive definitions can be showed undecidable, by reduction from the universality problem for context-free languages, a well-known undecidable problem. To bypass this problem, certain restrictions on the syntax of the SL system are required, namely:

- *Progress*: each rule allocates exactly one node, called the root of the rule. This condition makes our technical life easier, and can be lifted in many cases — rules with more than one allocation can be split by introducing new predicates.
- *Connectivity*: for each inductive rule of the form $\Sigma * \mathsf{P}_1(\mathbf{x}_1) * \ldots * \mathsf{P}_n(\mathbf{x}_n) \wedge \Pi$, there exists at least one edge between the root of the rule and the root of each rule of P_i, for all $i \in [1, n]$. This restriction prevents the encoding of context-free languages in SL, which requires disconnected rules.
- *Establishment*: all existentially quantified variables in a recursive rule are eventually allocated. This restriction is not required for the satisfiability problem, but it is essential for entailment, as explained next.

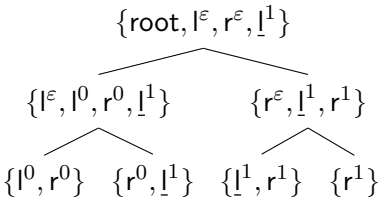
The fragment of SL obtained by applying the above, rather natural, restrictions, is denoted $\mathsf{SL}_{\mathrm{btw}}$ in the following. The proof of decidability for entailments in $\mathsf{SL}_{\mathrm{btw}}$ relies on three main ingredients:

1. for each predicate $[\![\mathsf{P}]\!]$ in the system, all heaps from the least solution $[\![\mathsf{P}]\!]$ are represented by graphs, whose *treewidth is bounded* by a linear function in the size of the system.
2. we define, for each predicate P, a formula Ψ_{P} in *monadic second-order logic* (MSO) of graphs whose models are exactly the graphs encoding the heaps from the least solution $[\![\mathsf{P}]\!]$.

3. the entailment $\mathsf{P} \models_{\mathsf{SL}} \mathsf{Q}$ is reduced to the satisfiability of an MSO formula $\Psi_{\mathsf{P}} \wedge \neg\Psi_{\mathsf{Q}}$. Since all models of P (thus of Ψ_{P}) have bounded treewidth, this problem is decidable, by Courcelle's Theorem [Cou90].

Intuitively, the *treewidth* of the graph measures how close the graph is of being a tree. Formally, the treewidth is the minimum width among all tree decompositions of the graph, where the width of the decomposition is the maximal size among its sets minus one. A *tree decomposition* is a coverage of the graph using a tree labeled with sets of vertices, such that the endpoints of each edge are present in the same set, and each vertex corresponds to a connected path in the tree.

For heaps that are models of predicates in a $\mathsf{SL}_{\mathrm{btw}}$ system of inductive definitions, a natural tree decomposition uses the structure of the corresponding unfolding trees. By the progress condition, each node in the unfolding tree contains a points-to proposition that allocates a heap cell. The selector edges between two cells allocated by adjacent nodes in the unfolding tree are said to be *local*. For instance the edges $\mathsf{root} \mapsto \mathsf{l}^{\varepsilon}$, $\mathsf{root} \mapsto \mathsf{r}^{\varepsilon}$, $\mathsf{l}^{\varepsilon} \mapsto \mathsf{ll}$ and $\mathsf{r}^{\varepsilon} \mapsto \mathsf{rr}$ are all local, for the model of the $\mathsf{TLL}(\mathsf{root}, \mathsf{ll}, \mathsf{lr})$ predicate depicted above. On the other hand, the edges $\mathsf{ll} \mapsto \mathsf{r}^0$, $\mathsf{r}^0 \mapsto \mathsf{l}^1$ and $\mathsf{l}^1 \mapsto \mathsf{lr}$ are non-local in this heap. Moreover, we place in each node of the tree the set of heap cells which are allocated by the formula labeling that node in the (isomorphic) unfolding tree and those which are the destination of a local edge. A cell which is the destination of a non-local edge is placed in the same node as the origin of that edge, and in all nodes on the path between the two endpoints of the edge. For instance, the tree decomposition of the previous model of $\mathsf{TLL}(\mathsf{root}, \mathsf{ll}, \mathsf{lr})$ is shown below:



Observe that the memory cell l^1 (underlined) is placed in all nodes on the path between the node who allocates and the one who references it. The width of this decomposition (and thus the tree width of the heap structure) is bounded by a linear function in the number of existentially quantified variables that create non-local edges in the heap, hence also in the size of the system. In our example, each node of the tree decomposition is traversed by at most one non-local edge, established by the existentially quantified variable z, in the inductive rule of the definition of TLL.

The translation of a predicate P into an MSO formula on graphs Ψ_{P} uses standard definitions of trees in MSO. The non-local edges are defined by first building *tree walking automata* (TWA) [Boj08] that track the destination of a non-local edge through the unfolding tree, using the propagation information provided by the parameters of each rule. Then a TWA is encoded in MSO using a standard automata-logic connection. Consequently,

we obtain the following result:

Theorem ([IRS13]). *The classes of entailment and satisfiability problems for the logic* $\mathsf{SL}_{\mathrm{btw}}$ *are in* ELEMENTARY.

The direct translation of $\mathsf{SL}_{\mathrm{btw}}$ into MSO is not practical, because of its high complexity, and the lack of efficient solvers for MSO on graphs. For these reasons, we defined a subset of $\mathsf{SL}_{\mathrm{btw}}$, called $\mathsf{SL}_{\mathrm{loc}}$ [IRV14b], in which only heap structures with local edges (w.r.t. the unfolding tree) can be defined. For a $\mathsf{SL}_{\mathrm{loc}}$ predicate P, we can build, in polynomial time, a tree automaton A_{P} that encodes all models of P. The entailment problem is then reduced to a language inclusion problem between tree automata.

The main difficulty incurred by the direct translation of $\mathsf{SL}_{\mathrm{loc}}$ into tree automata is the *polymorphic representation problem*: the same set of structures can be defined in several different ways, and tree automata simply mirroring the definition will not report the entailment. For example, doubly-linked lists defined by the predicate DLL, can alternatively be defined as:

$$\begin{array}{lcl} \mathsf{DLl}_{rev}(\mathsf{head},\mathsf{next},\mathsf{tail}) & \equiv & \mathsf{head} \mapsto (\mathsf{nil},\mathsf{next}) \wedge \mathsf{head} = \mathsf{tail} \vee \\ & & \exists x \ . \ \mathsf{tail} \mapsto (x,\mathsf{next}) * \mathsf{DLL}_{rev}(\mathsf{head},\mathsf{tail},x) \end{array}$$

Then the entailment $\mathsf{DLL}(\mathsf{head},\mathsf{nil},\mathsf{tail}) \models_{\mathsf{SL}} \mathsf{DLL}_{rev}(\mathsf{head},\mathsf{nil},\mathsf{tail})$ holds, but a naïve structural translation of $\mathsf{SL}_{\mathrm{loc}}$ to TA might not detect this fact. To bridge this gap, we define a closure operation on TA, called *canonical rotation*, which adds all possible representations of a given inductive definition, encoded as a tree automaton. As the canonical rotation takes polynomial time, and, moreover, language inclusion between nondeterministic bottom-up tree automata is EXPTIME-complete [CDG+05], we obtain the following:

Theorem ([IRV14b]). *The class of entailment problems for the logic* $\mathsf{SL}_{\mathrm{loc}}$ *is* EXPTIME*-complete.*

The practical outcome of this work is the SLIDE tool [IRV], developed by Adam Rogalewicz (Brno University of Technology, Czech Republic). We contributed with several benchmarks and won a silver medal in the SL-COMP'14 [SC14] competition of SL solvers, organized as part of the more traditional Satisfiability Modulo Theories Competition SMT-COMP'14.

Chapter 3

Integer Arithmetic

In this chapter we introduce the first-order theory of *integer arithmetic* $\langle \mathbb{Z}, +, \cdot \rangle$ and discuss several of its decidable fragments. In particular, we present new results concerning the decidability within the fragment of the theory with addition and divisibility predicate [BI05], and a decidable class of non-linear Diophantine systems [BIL06, BIL09]. Besides purely theoretical interest, these results are needed to establish decidability and complexity results concerning program verification problems (Chapters 4 and 6).

We denote by $\mathbb{Z}$ the set of integers, by $\mathbb{N}$ the set of positive integers, zero included, and by $\mathbb{N}_+$ the set $\mathbb{N} \setminus \{0\}$. Formally, $\langle \mathbb{Z}, +, \cdot \rangle$ denotes the set of valid first-order sentences, where the quantified variables range over integers, and $+, \cdot$ are interpreted as the standard addition and multiplication functions. We define 0 and 1 using the predicates $zero(x) \equiv \forall y \,.\, x \cdot y = x$ and $one(x) \equiv \forall y \,.\, x \cdot y = y$. The set of positive integers is defined using Lagrange's Four Square Theorem: $pos(x) \equiv \exists y, z, t, w \,.\, x = y \cdot y + z \cdot z + t \cdot t + w \cdot w$. This allows us to further define the order relation $x \leq y \equiv \exists z \,.\, pos(z) \wedge x + z = y$. The divisibility predicate is defined as $x \mid y \equiv \exists z \,.\, pos(z) \wedge x \cdot z = y$. The *greatest common divisor* (gcd) and *least common multiple* (lcm) are defined by the formulae below:

$$\gcd(x, y) = z \equiv \forall t \,.\, t \mid x \wedge t \mid y \leftrightarrow t \mid z \qquad \mathrm{lcm}(x, y) = z \equiv \forall t \,.\, x \mid t \wedge y \mid t \leftrightarrow z \mid t$$

The divisibility predicate allows to define primality as $prime(x) \equiv \forall y \,.\, y \mid x \rightarrow (y = 1 \vee y = x)$ and encode famous open problems, such as Goldbach's Conjecture: $\forall x \,.\, x \geq 1 \rightarrow \exists y, z \,.\, prime(y) \wedge prime(z) \wedge x + x = y + z$.

It comes with little surprise that the validity problem for integer arithmetic: *given a sentence φ, does it belong to the theory $\langle \mathbb{Z}, +, \cdot \rangle$?* is undecidable. Formally, this occurs as a consequence of Gödel's Incompleteness Theorem [Gö31], and was proved by Church [Chu36]. This result is strengthened

further by the undecidability of the theory $\langle \mathbb{N}, |, + \rangle$ of divisibility and addition [Rob49] and of the existential fragment of integer arithmetic $\langle \mathbb{N}, +, \cdot \rangle^{\exists}$. The latter is a consequence of the undecidability of *Hilbert's Tenth Problem*, asking for the existence of solutions for a given non-linear Diophantine system, proved undecidable by Matiyasevich [Mat70].

On the positive side, one can recover the decidability by ignoring one of the two operations of $\langle \mathbb{Z}, +, \cdot \rangle$. Without loss of generality, we restrict the interpretation domain to $\mathbb{N}$, in the rest of this chapter. The additive $\langle \mathbb{N}, + \rangle$ and multiplicative $\langle \mathbb{N}, \cdot \rangle$ fragments of integer arithmetic where shown to be decidable by Presburger [Pre29] and Mostowski [Mos52]. More expressive decidable theories are $\langle \mathbb{N}, +, V_p \rangle$ [BHMV94] and $\langle \mathbb{N}, +, x \mapsto p^x \rangle$ [Sem79], where $V_p(x)$ is the greatest power of p that divides x and $x \mapsto p^x$ is the exponentiation function, for a fixed prime number p. Also, by restricting the quantifier prefix to existentials only, Bel'tyukov [Bel76] and Lipshitz [Lip76a] proved (independently) that the fragment $\langle \mathbb{N}, |, + \rangle^{\exists}$ is decidable.

In this thesis we present two new decidable extensions of Presburger arithmetic, i.e. the additive theory of natural numbers $\langle \mathbb{N}, + \rangle$. The first result is the decidability (of the validity problem) for the class of formulae of the form $Q_0 z Q_1 x_1 \dots Q_n x_n \,.\, \varphi(\mathbf{x}, z)$, where $Q_0, Q_1, \dots, Q_n \in \{\exists, \forall\}$, φ is quantifier-free, and all divisibility propositions are of the form $f(z) \mid g(\mathbf{x}, z)$, with f and g linear (additive) terms. Observe that only one variable, namely z, is allowed on the left-hand side of the divisibility sign. This fragment is called $\mathsf{L}^1_{\mathrm{div}}$ in the following. We show that any formula in this fragment can be reduced to a Presburger formula, by successively eliminating the quantifiers $Q_n, \dots, Q_1$, thus proving decidability of the $\mathsf{L}^1_{\mathrm{div}}$ fragment.

We further generalize the $\mathsf{L}^1_{\mathrm{div}}$ fragment, by allowing several existentially quantified variables to occur to the left of the divisibility sign that is, formulae of the form $\exists z_1 \dots \exists z_n Q_1 x_1 \dots Q_m x_m \varphi(\mathbf{x}, \mathbf{z})$, where φ does not contain quantifiers and the only divisibility propositions are of the form $f(\mathbf{z}) \mid g(\mathbf{x}, \mathbf{z})$, for linear terms f and g. In this fragment, denoted $\exists \mathsf{L}^*_{\mathrm{div}}$, decidability is established only for the formulae in which each divisibility proposition occurs under an even number of negations. This is sharpened by showing that negated divisibility propositions lead to undecidability.

The second result is the decidability of the existence of solutions for a class of non-linear Diophantine systems. A *Diophantine equation* is an atomic proposition $P(\mathbf{x}) = 0$, where $P(\mathbf{x}) = \sum_{i=1}^{n} a_i \cdot M_i(\mathbf{x}) + a_0$ is a polynomial with integer coefficients $a_0, \dots, a_n$ and $M_1, \dots, M_n$ are monomials of the form $M_\ell(\mathbf{x}) = \Pi_{j=1}^{k_\ell} x_j^{i_{\ell,j}}$, with $i_{\ell,1}, \dots, i_{\ell,k_\ell} \in \mathbb{N}$. A Diophantine equation is said to be *linear with parameter* x_p if, for every monomial M_ℓ, we have

$\sum_{j \neq p} i_j^\ell \leq 1$. Note that any Diophantine linear equation with parameter z can be equivalently written as $\sum_{i=1}^n P_i(z) \cdot x_i + P_0(z) = 0$, where P_i are polynomials of arbitrary degree in z. In the following, we denote by D^1 the set of positive boolean combinations of linear Diophantine equations with a certain parameter z. The decidability of the existence of solutions of Diophantine linear systems with parameter z is the key to showing the decidability of the D^1 fragment of integer arithmetic.

3.1 Decidability of the $\mathsf{L}^1_{\mathrm{div}}$ Fragment of $\langle \mathbb{N}, +, \cdot \rangle$

To simplify our technical life, we will work first under the assumption that each divisibility atomic proposition is of the form $z \mid f(\mathbf{x}, z)$, where f is a linear term. The generalization to atomic propositions of the form $az + b \mid f(\mathbf{x}, z)$, with $a, b \in \mathbb{Z}$ is explained at the end of this section. The input formula, given in DNF, is of the form:

$$Q_0 z Q_1 x_1 \ldots Q_n x_n \bigvee_{i=1}^{N} \Big(\bigwedge_{j=1}^{M_i} z \mid f_{ij}(\mathbf{x}, z) \wedge \bigwedge_{j=1}^{P_i} \neg z \mid g_{ij}(\mathbf{x}, z) \wedge \varphi_i(\mathbf{x}, z) \Big) \quad (3.1)$$

where $Q_0, \ldots, Q_n \in \{\exists, \forall\}$, f_{ij} and g_{ij} are linear terms, and φ_i are Presburger formulae with free variables $\mathbf{x} = \{x_1, \ldots, x_n\}$ and z.

The first step is to eliminate all variables from $\mathbf{x}$ that occur within linear atomic propositions in some Presburger constraint φ_i. By possibly eliminating the quantifiers in φ_i, we assume w.l.o.g. that $\varphi_i(\mathbf{x}, z) \equiv \bigvee_k \bigwedge_\ell \exists t_{k\ell} \,.\, t_{k\ell} \geq 0 \;\wedge\; h_{k\ell}(\mathbf{x}, z) + t_{k\ell} = 0 \;\wedge\; \bigwedge_\ell c_{k\ell} \mid h'_{k\ell}(\mathbf{x}, z)$, with $h_{k\ell}, h'_{k\ell}$ linear terms, and $c_{k\ell} \in \mathbb{N}$. That is, we write each linear inequality $h_{k\ell}(\mathbf{x}, z) \leq 0$ as an equality that uses an existentially quantified slack variable $t_{k\ell}$. Suppose now that x_m, for some $m = 1, \ldots, n$, appears in a linear term $h_{k\ell}(\mathbf{x}) = a_{k\ell} x_m + b_{k\ell}(\mathbf{x}, z)$, with non-zero coefficient. We multiply through with $a_{k\ell}$ by replacing all formulas of the form $h(\mathbf{x}, z) + t = 0$ with $a_{k\ell} h(\mathbf{x}, z) + a_{k\ell} t = 0$, $c \mid h'(\mathbf{x}, z)$ with $a_{k\ell} c \mid a_{k\ell} h'(\mathbf{x}, z)$, and $z \mid f(\mathbf{x}, z)$ with $a_{k\ell} z \mid a_{k\ell} f(\mathbf{x}, z)$. Then we substitute $a_{k\ell} x_m$ with the term $-b_{k\ell}(\mathbf{x}, z) - t_{k\ell}$, which does not contain x_m. Finally, we eliminate the existentially quantified slack variables $t_{k\ell}$ by successive applications of the *Chinese Remainder Theorem* (CRT):

$$\exists x \,.\, \bigwedge_{i=1}^{K} m_i \mid (x - r_i) \Leftrightarrow \bigwedge_{1 \leq i, j \leq K} \gcd(m_i, m_j) \mid (r_i - r_j) \;\;.$$

The remaining formula is of the form below:

$$Q_0 z Q_1 x_1 \ldots Q_n x_n \bigvee_{i=1}^{N} \Big(\bigwedge_{j=1}^{M_i} z_{ij} \mid f_{ij}(\mathbf{x}, z) \wedge \bigwedge_{j=1}^{P_i} \neg z_{ij} \mid g_{ij}(\mathbf{x}, z) \wedge \psi_i(z) \Big) \quad (3.2)$$

where $z_{ij} \in \{a_{ij}z, c_{ij}\}$, $a_{ij} \in \mathbb{N}_+$, $c_{ij} \in \mathbb{N}$, and $\psi_i(z)$ are Presburger formulae, in which z occurs free.

The decision procedure eliminates the quantifiers $Q_n x_n, \ldots, Q_1 x_1$, in this order, and reduces the outcome of this transformation to an equivalent Presburger formula. For the quantifier elimination, we consider three cases, based on the type of the last quantifier $Q_n \in \{\exists, \forall\}$ and the sign (positive, negative) of the divisibility propositions.

The Existential Positive Case

If $Q_n \equiv \exists$ and there are no negated divisibility propositions, (3.2) becomes:

$$\bigvee_{i=1}^{N} \exists x_n \bigwedge_{j=1}^{M_i} z_{ij} \mid (a_{ij}x_n + g_{ij}(x_1, \ldots, x_{n-1}, z)) \wedge \psi_i(z) \tag{3.3}$$

where $a_{ij} \neq 0$ and g_{ij} are linear terms not involving x_n. We eliminate x_n from the above formula, using the following generalization of the CRT:

$$\exists x \bigwedge_{i=1}^{K} m_i \mid (a_i x - r_i) \Leftrightarrow \bigwedge_{1 \leq i,j \leq K} \gcd(a_i m_j, a_j m_i) \mid (a_i r_j - a_j r_i) \wedge \bigwedge_{i=1}^{K} \gcd(a_i, m_i) \mid r_i$$

Using the equivalence $\gcd(az, c) \mid f \Leftrightarrow \bigvee_{r=0}^{c-1} az \equiv r \mod c \wedge \gcd(r, c) \mid f$, we show that the resulting formula has two types of divisibility predicates:
- $az \mid t(x_1, \ldots, x_{n-1}, z)$, and
- $a \mid t(x_1, \ldots, x_{n-1}, z)$ (Presburger constraints),

where $a \in \mathbb{N}_+$ and t is a linear term not involving x_n. We have obtained a formula of the form (3.3) with $n-1$ variables occurring on the right-hand side of the divisibility sign.

The Universal Positive Case

If $Q_n \equiv \forall$ and there are no negated divisibility propositions, we write first the formula (3.2) in CNF:

$$\bigwedge_{i=1}^{P} \forall x_n \bigvee_{j=1}^{Q_i} z_{ij} \mid (a_{ij}x_n + g_{ij}(x_1, \ldots, x_{n-1}, z)) \vee \psi_i(z) \tag{3.4}$$

In each conjunct above, the union of Q_i arithmetic progressions $\{x : a_{ij}x \equiv -g_{ij} \mod z_{ij}\}_{j=1}^{Q_i}$ covers the entire set of natural numbers. The following theorem, stated as a conjecture by Erdös[1] and proved by Crittenden and

[1] Erdös also put a 25$ prize on it.

Vanden Eynden [CE69], gives the means to eliminate the universal quantifier in this case:

Theorem 1 ([CE69])**.** *Let* $a_1, \ldots, a_n \in \mathbb{Z}, b_1, \ldots, b_n \in \mathbb{N}_+$. *Suppose there exists an integer* x_0 *satisfying none of the congruences:* $\{x \equiv a_i \mod b_i\}_{i=1}^n$. *Then there is such an* x_0 *among* $1, \ldots, 2^n$.

We shall use this theorem in its positive form: n arithmetic progressions $\{a_i + b_i\mathbb{N}\}_{i=1}^n$ cover $\mathbb{N}$ if and only if they cover the set $1, \ldots, 2^n$. The result of the quantifier elimination in (3.4) is thus:

$$\bigwedge_{i=1}^{P} \bigwedge_{t=1}^{2^{Q_i}} \bigvee_{j=1}^{Q_i} z_{ij} \mid a_{ij}t + g_{ij} \vee \psi_i(z) \ .$$

The Universal Mixed Case

If $Q_n \equiv \forall$ and there are negated divisibility propositions, we consider the formula (3.2) in CNF, with the negated propositions occurring on the left-hand side of the implication below:

$$\bigwedge_{i=1}^{P} \forall x_n \left(\left(\bigwedge_{j=1}^{R_i} z_{ij} \mid g_{ij}(\mathbf{x}, z) \right) \rightarrow \bigvee_{j=1}^{Q_i} z_{ij} \mid f_{ij}(\mathbf{x}, z) \right) \vee \psi_i(z) \tag{3.5}$$

Each formula $\forall x_n \ . \ \left(\bigwedge_{j=1}^{R_i} z_{ij} \mid a_{ij}x_n + b_{ij}(x_1, \ldots x_{n-1}, z) \right) \rightarrow \bigvee_{j=1}^{Q_i} z_{ij} \mid c_{ij}x_n + d_{ij}(x_1, \ldots, x_{n-1}, z)$ states that the arithmetic progression $\{x : \bigwedge_{j=1}^{R_i} z_{ij} \mid a_{ij}x + b_{ij}(x_1, \ldots x_{n-1}, z)\}$ is covered by the union of the arithmetic progressions $\{x : z_{ij} \mid c_{ij}x + d_{ij}(x_1, \ldots, x_{n-1}, z)\}$, respectively. An application of Theorem 1 eliminates x_n and transforms each conjunct of the formula (3.5) into:

$$\neg\exists y \bigwedge_{j=1}^{R_i} z_{ij} \mid a_{ij}y + b_{ij} \vee \exists y \bigwedge_{j=1}^{R_i} z_{ij} \mid a_j y + b_{ij} \wedge \bigwedge_{t=1}^{2^{Q_i}} \bigvee_{j=1}^{Q_i} z_{ij} \mid c_{ij}\left(y + \frac{zk_{ij}t}{\gcd(z, \ell_{ij})}\right) + d_{ij} \ .$$

The first disjunct is for the trivial case, in which the set $\{x : \bigwedge_{j=1}^{R} z_{ij} \mid g_{ij}(\mathbf{x}, z)\}$ is empty, while the second disjunct assumes the existence of an element y of this set and encodes the condition of Theorem 1: the first 2^{Q_i} elements of this set, starting with y, must be covered by the union of Q_i progressions. The existentially quantified variables y can be eliminated from the above formula using the CRT, as in the previous existential positive case. Observe that we have introduced subterms of the form $\frac{z}{\gcd(z,k)}$, with constants $k \in \mathbb{N}_+$, within the linear terms f_{ij}. This is reflected in the definition of the solved form, in the next paragraph.

The Solved Form

For any formula of type (3.2), the result of eliminating the quantifiers $Q_n x_n \dots Q_1 x_1$ is a formula in the following *solved form*:

$$\bigvee_{i=1}^{N} \bigwedge_{j=1}^{M_i} a_{ij} z \mid f_{ij}(z) \wedge \bigwedge_{j=1}^{P_i} \neg b_{ij} z \mid g_{ij}(z) \wedge \psi_i(z) \tag{3.6}$$

where $a_{ij}, b_{ij} \in \mathbb{N}_+$ are constants, f_{ij}, g_{ij} are linear combinations of terms of the form $\frac{z}{\gcd(z,k)}$, with constants $k \in \mathbb{N}_+$, and ψ_i are Presburger formulae[2]. Let $az \mid \sum_{i=1}^{n} \frac{zc_i}{\gcd(z,k_i)} + c_0$ be an atomic proposition in (3.6). We replace this proposition the an equivalent formula:

$$\bigvee_{(d_1,\dots,d_n) \,\in\, \mathrm{div}(k_1)\times\dots\times\mathrm{div}(k_n)} \bigwedge_{i=1}^{n} \gcd(z,k_i) = d_i \wedge aDz \mid z\Sigma_{i=1}^{n} c_i D_i + c_0 D$$

where $D = \Pi_{i=1}^{n} d_i$, $D_i = \frac{D}{d_i}$ and $\mathrm{div}(k)$ denotes the set of divisors of the constant $k \in \mathbb{N}_+$. Moreover, $aDz \mid z\Sigma_{i=1}^{n} c_i D_i + c_0 D$ implies $z \mid c_0 D$, hence the above formula is equivalent to:

$$\bigvee_{(d,d_1,\dots,d_n) \,\in\, \mathrm{div}(c_0 D)\times\mathrm{div}(k_1)\times\dots\times\mathrm{div}(k_n)} \bigwedge_{i=1}^{n} (d,k_i) = d_i \wedge aDd \mid d\Sigma_{i=1}^{n} c_i D_i + c_0 D$$

which involves only constants, and can be evaluated to true of false.

General Divisibility Predicates

In the case where the divisibility propositions are of the form $f(z) \mid g(\mathbf{x}, z)$, with f and g linear terms, the quantifier elimination procedure may produce subterms such as $\gcd(f_i(z), f_j(z))$ and $\mathrm{lcm}\left(\left\{\frac{f_j(z)}{\gcd(f_j(z),k_j)}\right\}_{j=1}^{R}\right)$, for some linear terms f_i, f_j and constants $k_j \in \mathbb{N}_+$. Essentially, we eliminate all terms of the form $\gcd(f_i(z), f_j(z))$ in a divisibility predicate $\gcd(f_i, f_j) \mid h_{ij}$ using polynomial division in the ring of univariate polynomials with integer coefficients $\mathbb{Z}[z]$. Summing up, we obtain the following theorem:

Theorem 2 ([BI05])**.** *Validity is decidable for the fragment* $\mathsf{L}^1_{\mathrm{div}}$ *of integer arithmetic, consisting of formulae* $Q_0 z Q_1 x_1 \dots Q_n x_n \,.\, \varphi(\mathbf{x}, z)$*, where all divisibility propositions are of the form* $f(z) \mid g(\mathbf{x}, z)$*, with* f, g *linear terms.*

[2] We write z as $\frac{z}{\gcd(z,1)}$.

Example 1. *Consider the open formula* $\forall x \forall y \ . \ z \mid 12x + 4y \rightarrow z \mid 3x + 12y$. *To eliminate* y *we apply the universal mixed case and obtain:*

$$\forall x \left(\neg \exists y \ z \mid 12x + 4y \vee \exists y \ z \mid 12x + 4y \wedge z \mid 3x + 12y \wedge z \mid 3x + 12(y + \frac{z}{\gcd(z,4)}) \right)$$

By an application of the CRT, $\exists y \ . \ z \mid 12x + 4y$ *is equivalent to* $\gcd(z,4) \mid 12x$ *which is trivially true, since* $\gcd(z,4) \mid 4$ *and* $4 \mid 12x$. *Moreover, if* $z \mid 3x + 12y$, *then* $z \mid 3x + 12y + 12\frac{z}{\gcd(z,4)}$ *is equivalent to* $z \mid 12\frac{z}{\gcd(z,4)}$, *which is also trivially true. Hence, the formula can be simplified down to:* $\forall x \exists y \ . \ z \mid 12x + 4y \wedge z \mid 3x + 12y$. *By an application of the CRT we obtain:* $\forall x \ . \ z \mid 33x \wedge \gcd(z,4) \mid 12x \wedge \gcd(z,12) \mid 3x$ *which, after trivial simplifications, is equivalent to* $z \mid 33 \wedge \gcd(z,12) \mid 3$, *leading to* $z \in \{1, 3, 11, 33\}$. ■

The decidability of the $\mathsf{L}^1_{\mathrm{div}}$fragment of integer arithmetic has been used to establish a decidability result concerning the verification of safety properties on programs with singly-linked list data structures (Chapter 6).

3.2 Decidability within the $\exists \mathsf{L}^*_{\mathrm{div}}$ Fragment of $\langle \mathbb{N}, +, \cdot \rangle$

We generalize the $\mathsf{L}^1_{\mathrm{div}}$theory by considering divisibility propositions $f(\mathbf{z}) \mid g(\mathbf{x}, \mathbf{z})$, where $\mathbf{z}$ are existentially quantified, $\mathbf{x} \cap \mathbf{z} = \emptyset$ and f and g are linear terms. By first eliminating the variables that occur within linear constraints as in the previous, we obtain the following type of formulae:

$$\exists z_1 \ldots \exists z_n Q_1 x_1 \ldots Q_m x_m \bigvee_{i=1}^{N} \left(\bigwedge_{j=1}^{M_i} f_{ij}(\mathbf{z}) \mid g_{ij}(\mathbf{x}, \mathbf{z}) \wedge \bigwedge_{j=1}^{P_i} \neg f'_{ij}(\mathbf{z}) \mid g'_{ij}(\mathbf{x}, \mathbf{z}) \wedge \varphi_i(\mathbf{z}) \right)$$

We show the decidability of the *positive* fragment, where $P_i = $ for all $i = 1, \ldots, N$, and undecidability of the $\exists \mathsf{L}^*_{\mathrm{div}}$ fragment, in general.

By applying the quantifier elimination procedure used for $\mathsf{L}^1_{\mathrm{div}}$, in the existential positive case, we introduce divisibility propositions of the form $\gcd(f_1(\mathbf{z}), \ldots, f_k(\mathbf{z})) \mid h(\mathbf{z})$, which are equivalently written, using CRT, as:

$$\exists y_1 \ldots \exists y_{k-1} \ . \ f_1(\mathbf{z}) \mid y_1 - h(\mathbf{z}) \wedge \bigwedge_{i=2}^{k-1} f_i(\mathbf{z}) \mid y_i - y_{i-1} \wedge f_k(\mathbf{z}) \mid y_{k-1} \ .$$

Observe that the universal positive case only substitutes variables for constants and the universal mixed case does not apply, because we are in the positive fragment of $\exists \mathsf{L}^*_{\mathrm{div}}$. We have thus reduced the positive fragment of $\exists \mathsf{L}^*_{\mathrm{div}}$ to the existential fragment of the $\langle \mathbb{N}, +, | \rangle$ theory, which has been shown to be decidable [Bel76, Lip76a].

Theorem 3 ([BI05]). *Validity is decidable for the positive fragment* $\exists \mathsf{L}^*_{\text{div}}$ *of integer arithmetic, consisting of formulae* $\exists z_0 \ldots \exists z_m Q_1 x_1 \ldots Q_n x_n \,.\, \varphi(\mathbf{x}, \mathbf{z})$, *in which all divisibility propositions (i) are of the form* $f(\mathbf{z}) \mid g(\mathbf{x}, \mathbf{z})$, *with* f, g *linear terms, and (ii) they occur under even numbers of negations.*

This result is sharpened by observing that, allowing negated divisibility propositions in $\exists \mathsf{L}^*_{\text{div}}$ leads to undecidability, by the following reduction from Hilbert Tenth's Problem [Mat70]. First, we encode multiplication using addition and the squaring function as: $(x+y)^2 - (x-y)^2 = 4xy$. Then we define the squaring function using the least common multiple: $x^2 = y \Leftrightarrow x + y = \text{lcm}(x, x+1)$. Finally, we use the definition of the least common multiple [Rob49]: $\text{lcm}(x, y) = z \Leftrightarrow \forall t \,.\, x \mid t \wedge y \mid t \leftrightarrow z \mid t$. The latter definition belongs to the $\exists \mathsf{L}^*_{\text{div}}$ fragment that uses negated divisibility propositions.

3.3 Decidability of the D^1 Fragment of $\langle \mathbb{N}, +, \cdot \rangle$

Let us fix a linear Diophantine system with parameter z, of the form:

$$\Big\{ \sum_{j=1}^{n} P_{ij}(z) \cdot x_j + Q_i(z) = 0 \Big\}_{i=1}^{r} \tag{3.7}$$

where $P_{ij}, Q_i \in \mathbb{Z}[z]$ are univariate polynomials with integer coefficients and variable z, for all $i = 1, \ldots, r$. The problem we ask is the existence of a valuation $\nu : \{z, x_1, \ldots, x_n\} \to \mathbb{N}$ that satisfies all equations of the system.

Let us consider first the case when the system is homogeneous, i.e. $Q_i(z) = 0$, for all $i = 1, \ldots, r$. The general case is dealt with by adding a new variable x_{n+1}, replacing each occurrence of $Q_i(z)$ by $Q_i(z) \cdot x_{n+1}$, and looking only after solutions in which $x_{n+1} = 1$.

Let $P(z)$ be the polynomial greatest common divisor of all $P_{ij}(z)$, obtained by applying Euclid's algorithm in the polynomial ring $\mathbb{Z}[z]$. Since $P(z) = \sum_{i=0}^{k} p_i \cdot z^i$ is a univariate polynomial, its set of roots is finite and effectively computable. If $P(m_0) = 0$ for some $m_0 \in \mathbb{N}$, then $m_0, x_1, \ldots, x_n$ is a solution of the (homogeneous) system (3.7), for any choice of $x_1, \ldots, x_n \in \mathbb{N}$. Moreover, if there exists such m_0, it necessarily divides p_0, and, since $P(z)$ is a divisor of every $P_{ij}(z)$, then necessarily p_0 divides the constant term (monomial of zero degree) of each $P_{ij}(z)$. Hence the binary size of m_0 is polynomially bounded by the size of the description of the system (3.7), and this case can be dealt by a nondeterministic polynomial-time algorithm.

We assume in the following that $P(m) \neq 0$, for all $m \in \mathbb{N}$, in other words that, for no value of m, $P_{ij}(m)$ will all become zero at the same time. If

the system has a solution, then it also has a solution which is minimal with respect to the pointwise partial ordering on $\mathbb{N}^{n+1}$. Let $A = [P_{ij}(z)]$ be the $r \times n$ matrix of the system (3.7), and $\mathbf{x} = (x_1, \ldots, x_n)^\top$, in the following. For some $m \in \mathbb{N}$, we denote by $A(m)$ the matrix $[P_{ij}(m)] \in \mathbb{Z}^{r \times n}$. Let $C > 0$ and $K \geq 0$ be the maximal absolute value of all coefficients, and the maximum degree of the polynomials in A, respectively. The following is an immediate consequence of a theorem due to Pottier [Pot90]:

Theorem 4. *For a fixed $m_0 \geq \max(C, n, r)$, let $x_1, \ldots, x_n$ be any minimal solution of the homogeneous system $A(m_0)\mathbf{x} = \mathbf{0}$. Then, for all $1 \leq i \leq n$, we have $x_i \leq m_0^{(K+3)r+1}$.*

Let us first assume that there exists a constant $0 \leq m < \max(C, n, r)$ such that $A(m)\mathbf{x} = 0$ has a solution. In this case, there exists a nondeterministic algorithm that guesses m and solves the linear Diophantine system $A(m)\mathbf{x} = 0$. The algorithm moreover operates in time bounded by a polynomial in the size of he binary representation of the system (3.7), since (i) the value m is polynomially bounded, and (ii) solving a linear Diophantine system is possible in polynomial time, with a nondeterministic algorithm.

Otherwise, for any $m \geq \max(C, n, r)$ the solution $x_1, \ldots, x_n$ can be represented in base m using at most $M = (K+3)r + 1$ digits (Theorem 4). Let $(x_i)_m = \sum_{j=1}^{M} \chi_{ij} \cdot m^j$ be the polynomial representing x_i in base m, with the implicit constraint $0 \leq \chi_{ij} < m$, for all $i = 1, \ldots, n$. The homogeneous system $A(m)\mathbf{x} = \mathbf{0}$ is encoded in base m, as follows.

First, we write the system as a set of equations of the form $P(m, x_1, \ldots, x_n) = Q(m, x_1, \ldots, x_n)$, with all coefficients of P and Q being positive. Since m is assumed to be greater than C, the maximal value of all coefficients c of the system, we have $(c)_m = c$. The operations of addition, multiplication by a constant $0 < c < m$, and multiplication by m, respectively, can be defined using linear constraints only. Let $(d)_m = \sum_{i=0}^{M} \delta_i \cdot m^i$, $(e)_m = \sum_{i=0}^{M} \epsilon_i \cdot m^i$ and $(f)_m = \sum_{i=0}^{M} \phi_i \cdot m^i$, with $0 \leq \delta_i, \epsilon_i, \phi_i < m$. We have:

$$
\begin{array}{rcl}
(d)_m + (e)_m = (f)_m & \Leftrightarrow & \bigvee_{\mathbf{b} \in \{0\} \times \{0,1\}^{M-2} \times \{0\}} \bigwedge_{i=0}^{M-1} \delta_i + \epsilon_i + b_i = \phi_i + m \cdot b_{i+1} \\
c \cdot (d)_m = (e)_m & \Leftrightarrow & \bigvee_{\mathbf{r} \in \{0\} \times \{0,1\}^{M-2} \times \{0\}} \bigwedge_{i=0}^{M-1} c \cdot \delta_i + r_i = \epsilon_i + m \cdot b_{i+1} \\
m \cdot (d)_m = (e)_m & \Leftrightarrow & \delta_M = \epsilon_0 = 0 \wedge \bigwedge_{i=0}^{M-1} \delta_i = \epsilon_{i+1}
\end{array}
$$

Observe that each addition and multiplication by a constant introduce M arithmetic carry bits $\mathbf{b} = \langle b_0, \ldots, b_{M-1} \rangle$. The translation of the Diophantine system $A(m)\mathbf{x} = 0$ using the above equivalences introduces a number of bits that is bound by polynomial function in the size of the system. A nondeterministic algorithm can guess first a bitvector, of polynomial length, that determines the values of all these bits, and rewrite the system $A(m)\mathbf{x} =$

0 as a system of linear constraints, of size bounded by a polynomial in the size in the original system. Checking whether the linear system has a solution is possible in polynomial time, with a nondeterministic algorithm. Consequently, we obtain the following theorem:

Theorem 5 ([BIL06, BIL09])**.** *The class of validity problems for the fragment* D^1 *of existentially quantified boolean combinations of non-linear Diophantine equations with parameter* z *is* NP*-complete.*

The decidability of the D^1 fragment of integer arithmetic has been used to establish the decidability of the class of reachability problems concerning flat counter machines with cycles labeled by parametric difference bounds and octagonal constraints (Theorems 7 and 9 in Chapter 4).

3.4 Discussion and Open Problems

The set of open problems concerning the arithmetic of integers $\langle \mathbb{Z}, +, \cdot \rangle$ is rather large (see [Bóz02] for a good survey). For instance, the decidability status is unknown for the satisfiability problems for $\langle \mathbb{N}, +, P \rangle$ or its existential fragment, where $P(x)$ means that x is prime.

Besides decidability, one may ask for the computational complexity classes of the decidable fragments. For instance, for any $i > 0$, the Σ_{i+1}-fragment Presburger arithmetic with $i + 1$ quantifier alternations, beginning with an existential quantifier is complete for Σ_i^{EXP} [Haa14]. Regarding extensions of Presburger arithmetic, the existential fragment of $\langle \mathbb{N}, +, | \rangle$, proved to be decidable by Lipshitz [Lip76a] and Bel'tyukov [Bel76], has been shown to be in NEXPTIME [LOW15], with no matching lower bound (an obvious bound is the NP-hardness of the quantifier-free Presburger arithmetic).

We ask similar problems for the $\mathsf{L}^1_{\mathrm{div}}$ and $\exists\mathsf{L}^*_{\mathrm{div}}$ fragments of $\langle \mathbb{N}, +, | \rangle$, namely finding tight complexity bounds for these fragments. Currently, we can derive elementary upper bounds by reduction to Presburger arithmetic, in the case of $\mathsf{L}^1_{\mathrm{div}}$ and to $\langle \mathbb{N}, +, | \rangle^{\exists}$, in the case of $\exists\mathsf{L}^*_{\mathrm{div}}$, respectively.

Chapter 4

Flat Counter Machines

Counter machines (CM) are a generalization of classical Rabin-Scott finite nondeterministic automata, extended with a set of variables ranging over $\mathbb{Z}$, and transitions described by integer arithmetic formulae. Formally, a counter machine is a tuple $M = \langle \mathbf{x}, Q, \iota, F, \Delta \rangle$, where:

- $\mathbf{x} = \{x_1, \ldots, x_N\}$ is a non-empty set of integer variables (counters),
- Q is a set of control states,
- $\iota \in Q$ is an initial state and $F \subseteq Q$ is a set of final states,
- Δ is a set of rules of the form $q \xrightarrow{\varphi(\mathbf{x},\mathbf{x}')} q'$, where q and q' are the source and destination states, $\mathbf{x}' = \{x' \mid x \in \mathbf{x}\}$ denote the values taken by the variables as result of the transition, and φ is an integer arithmetic formula with free variables in the set $\mathbf{x} \cup \mathbf{x}'$.

A *configuration* of M is a pair (q, ν), where q is a control state and $\nu : \mathbf{x} \to \mathbb{Z}$ is a valuation of the counters. A *run* of M is a (possibly infinite) sequence $(q_0, \nu_0), (q_1, \nu_1), \ldots$ of configurations, such that $q_0 = \iota$ and for each $i \geq 0$ there exists a rule $q_i \xrightarrow{\varphi_i(\mathbf{x},\mathbf{x}')} q_{i+1} \in \Delta$ such that, replacing each $x \in \mathbf{x}$ with $\nu_i(x)$ and each $x' \in \mathbf{x}'$ with $\nu_{i+1}(x)$ yields a valid (true) formula.

The *reachability problem* asks, given a counter machine M, does it have a run leading to a final control state ? On the other hand, the *termination problem* asks if every execution of M is finite. Both problems are undecidable, even if M is restricted to two counters x_1 and x_2 with only increment $x_i' = x_i + 1$, decrement $x_i' = x_i - 1$ and zero test $x_i = 0$ operations [Min67]. In this chapter we define a class of counter machines, called *flat*, for which both the reachability and termination problems are decidable. Moreover, we give tight complexity bounds for these problems, for the considered classes of counter machines.

An *elementary cycle* is a path $q_1 \xrightarrow{\varphi_1} q_2 \xrightarrow{\varphi_2} \ldots \xrightarrow{\varphi_{n-1}} q_n$, where $q_1 = q_n$ and $q_1, \ldots, q_{n-1}$ are pairwise distinct control states. A counter machine is said to be *flat* if (i) every control state belongs to at most one elementary cycle, and (ii) every rule not in a cycle is labeled by a quantifier-free Presburger formula. Under the flatness assumption, it is sufficient to define the transitive closures of the arithmetic relations labeling the cycles of the counter machine, in a decidable fragment of integer arithmetic.

Example 2. *Consider the flat CM* $M = \langle \{i, j, b\}, \{\ell_0, \ell_1, \ell_2, \ell_3\}, \ell_0, \{\ell_3\}, \Delta \rangle$ *in Figure 4.1. The machine increments* i *and* j *by executing the self-cycle on state* ℓ_1 *a number of times equal to the value of* b*, that was guessed on the transition* $\ell_0 \to \ell_1$*, then it will move to* ℓ_2 *and will increment* i*, while decrementing* j*, until* $j = 0$*. Finally, it moves to its final state if* $i = 2b$*.* ■

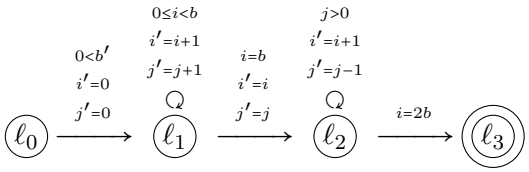


Figure 4.1: A flat counter machine

We will look at two classes of relations, defined by *difference bounds* [BIL06, BIL09] and *octagonal* [BGI09] constraints, whose transitive closures are Presburger-definable. Moreover, the reachability and termination problems for flat counter machines whose cycles are labeled with these kinds of relations are NP-complete (reachability) and in PTIME (termination).

First, let us coin several definitions. Let $\mathbb{Z}^{\mathbf{x}}$ be the set of valuations $\nu : \mathbf{x} \to \mathbb{Z}$. A relation $R \subseteq \mathbb{Z}^{\mathbf{x}} \times \mathbb{Z}^{\mathbf{x}}$ is *defined* by an integer arithmetic formula $\varphi(\mathbf{x}, \mathbf{x}')$ if $R = \{(\nu, \nu') \mid (\nu, \nu') \vDash \varphi\}$, where the forcing relation $(\nu, \nu') \vDash \varphi$ means that replacing each x by $\nu(x)$ and each x' by $\nu'(x)$ yields a valid formula. We denote by $\varnothing$ the empty (false) relation and by $\mathrm{I}_{\mathbf{x}}$ the identity relation defined by $\bigwedge_{x \in \mathbf{x}} x' = x$. The composition of two relations $R_1, R_2 \subseteq \mathbb{Z}^{\mathbf{x}} \times \mathbb{Z}^{\mathbf{x}}$ defined by $\varphi_1(\mathbf{x}, \mathbf{x}')$ and $\varphi_2(\mathbf{x}, \mathbf{x}')$, respectively, is the relation $R_1 \circ R_2 \subseteq \mathbb{Z}^{\mathbf{x}} \times \mathbb{Z}^{\mathbf{x}}$, defined by $\exists \mathbf{y} \,.\, \varphi_1(\mathbf{x}, \mathbf{y}) \wedge \varphi_2(\mathbf{y}, \mathbf{x}')$. Observe that $R \circ \varnothing = \varnothing \circ R = \varnothing$, for every relation R.

For any relation $R \subseteq \mathbb{Z}^{\mathbf{x}} \times \mathbb{Z}^{\mathbf{x}}$, we define $R^0 = \mathrm{I}_{\mathbf{x}}$ and $R^{n+1} = R^n \circ R = R \circ R^n$, for all $n \in \mathbb{N}$. R^n is called the n-th *power* of R in the sequel. The infinite sequence of relations $\{R^n\}_{n=0}^{\infty}$ is called the *power sequence* of R. With these notations, $R^+ = \bigcup_{n=1}^{\infty} R^n$ denotes the *transitive closure* of R, and $R^* = R^+ \cup \mathrm{I}_{\mathbf{x}}$

denotes the *reflexive and transitive closure* of R. A relation R is said to be $*$-*consistent* if and only if $R^n \neq \varnothing$, for all $n \in \mathbb{N}_+$. Observe that, if R is not $*$-consistent, there exists an integer $b > 0$ such that $R^n = \varnothing$, for all $n \geq b$.

Definition 1. *A* class *of relations is a monoid* $\langle \mathcal{R}(\mathbf{x}), \circ, \mathrm{I}_{\mathbf{x}} \rangle$*, where* $\mathcal{R}(\mathbf{x}) \subseteq 2^{\mathbb{Z}^{\mathbf{x}} \times \mathbb{Z}^{\mathbf{x}}}$ *is a set of relations with variables* $\mathbf{x}$*, that is closed under conjunction and composition and contains the relations* $\mathrm{I}_{\mathbf{x}}$ *and* $\varnothing$*. We write* $\mathcal{R}$ *for the union of all classes* $\mathcal{R}(\mathbf{x})$*.*

In this paper we will define classes by fragments of integer arithmetic. For a relation $R \in \mathcal{R}(\mathbf{x})$, we denote by $|R|$ the size of the canonical formula defining R, with coefficients represented in binary. As we show later on, all relations of interest in this chapter have canonical representations that can be computed in polynomial time, given any, not necessarily canonical, formula defining R.

4.1 Difference Bounds Relations

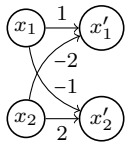
The core result concerning the definability of transitive closures in Presburger arithmetic is best understood by considering the class of *difference bounds* relations, defined by finite conjunctions of atomic propositions of the form $x - y \leq c$, where $c \in \mathbb{Z}$ is a constant. These constraints are also called *zones* in the literature on timed automata.

Definition 2. *A* difference bounds constraint $\phi(\mathbf{x})$ *is a finite conjunction of atomic propositions of the form* $x_i - x_j \leq \alpha_{ij}$*,* $1 \leq i, j \leq N$*, where* $\alpha_{ij} \in \mathbb{Z}$*. A relation* $R \subseteq \mathbb{Z}^{\mathbf{x}} \times \mathbb{Z}^{\mathbf{x}}$ *is a* difference bounds relation *if and only if it is defined by a difference bounds constraint* $\phi_R(\mathbf{x}, \mathbf{x}')$*. The class of difference bounds relations* $R \subseteq \mathbb{Z}^{\mathbf{x}} \times \mathbb{Z}^{\mathbf{x}}$ *is denoted* $\mathrm{DB}(\mathbf{x})$*.*

If $\phi(\mathbf{x})$ is a difference bounds constraint, a *difference bounds matrix* (DBM) representing ϕ is the matrix M_ϕ, where $(M_\phi)_{ij} = \alpha_{ij}$ if $x_i - x_j \leq \alpha_{ij}$ occurs in ϕ, and $(M_\phi)_{ij} = \infty$, otherwise. Dually, every DBM M corresponds to the difference bounds constraint $\Phi_M \equiv \bigwedge_{i,j=1}^{N} x_i - x_j \leq M_{ij}$. The *constraint graph* $\mathcal{G}_\phi$ of a difference bounds constraint ϕ is the weighted graph $\mathcal{G}_{M_\phi}$, whose incidence matrix is M_ϕ (Fig. 4.2 (a)).

A DBM M is said to be *consistent* if and only if the constraint Φ_M has a satisfying valuation. This is the case if and only if the constraint graph of M (Φ_M) contains no negative weight cycle. A consistent DBM M is said to be *closed* if $M_{ii} = 0$, for all $1 \leq i \leq N$, and all triangle inequalities $M_{ik} \leq M_{ij} + M_{jk}$ hold, for all $1 \leq i, j, k \leq N$. For a consistent DBM M, the unique closed DBM

logically equivalent to M is denoted $M^\star$ (Fig. 4.2 (b)). The closed DBM is a canonical (unique) representation of a difference bounds constraint. This canonical representation of a DBM can be computed in cubic time, using the classical Floyd-Warshall shortest path algorithm.



(a) $\mathcal{G}_R$

$$\begin{array}{c} \\ x_1 \\ x_2 \\ \\ x'_1 \\ x'_2 \end{array} \begin{array}{c} \begin{array}{cc} x_1 & x_2 \end{array} \\ \left(\begin{array}{cc|cc} 0 & \infty & 1 & -1 \\ \infty & 0 & -2 & 2 \\ \hline \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{array}\right) \end{array} \begin{array}{c} \\ \\ \\ \\ \begin{array}{cc} x'_1 & x'_2 \end{array} \end{array}$$

(b) M^*_R

Figure 4.2: Let $\phi(x_1, x_2, x'_1, x'_2) \equiv x_1 - x'_1 \leq 1 \wedge x_1 - x'_2 \leq -1 \wedge x_2 - x'_1 \leq -2 \wedge x_2 - x'_2 \leq 2$ be a constraint defining a DB relation. **(a)** shows the graph $\mathcal{G}_\phi$ and **(b)** the closed DBM representation of ϕ.

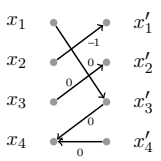
In the following, it is useful to define the constraint graph representing a composition of DB relations, in which the existentially quantified variables are kept explicitly. For a difference bounds relation R, defined by a constraint φ_R, we shall write $\mathcal{G}_R$ for the weighted graph $\mathcal{G}_{M^*_{\varphi_R}}$.

Definition 3. *Let $R \in \mathrm{DB}(\mathbf{x})$ be a relation and $n \in \mathbb{N}_+$ be an integer. The* unfolding graph *of R is $\mathcal{G}^n_R = \langle \bigcup_{k=0}^n \mathbf{x}^{(k)}, \rightarrow, w \rangle$, where $\mathbf{x}^{(k)} = \left\{ x_i^{(k)} \mid 1 \leq i \leq N \right\}$ and, for all $k = 0, \ldots, n$:*

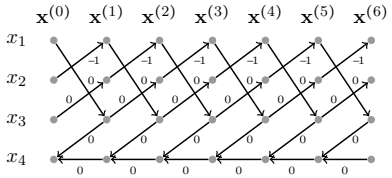
- $x_i^{(k)} \xrightarrow{c} x_j^{(k)}$ *if and only if* $x_i \xrightarrow{c} x_j$ *is an edge of* $\mathcal{G}_R$,
- $x_i^{(k)} \xrightarrow{c} x_j^{(k+1)}$ *if and only if* $x_i \xrightarrow{c} x'_j$ *is an edge of* $\mathcal{G}_R$,
- $x_i^{(k+1)} \xrightarrow{c} x_j^{(k)}$ *if and only if* $x'_i \xrightarrow{c} x_j$ *is an edge of* $\mathcal{G}_R$,
- $x_i^{(k+1)} \xrightarrow{c} x_j^{(k+1)}$ *if and only if* $x'_i \xrightarrow{c} x'_j$ *is an edge of* $\mathcal{G}_R$.

The following figure illustrates the definition of the constraint graph (a) and the unfolding graph (b) for the relation $R \equiv x_2 - x'_1 \leq -1 \wedge x_3 - x'_2 \leq 0 \wedge x_1 - x'_3 \leq 0 \wedge x'_4 - x_4 \leq 0 \wedge x'_3 - x_4 \leq 0$:

The key observation relating the power R^n of R and the unfolding graph $\mathcal{G}^n_R$ is the following: each difference constraint defining R^n is given by a minimal path between extremal vertices in $\mathcal{G}^n_R$. Formally, for all $i, j \in \{1, \ldots, N\}$,



(a) The constraint graph of R ($\mathcal{G}_R$)

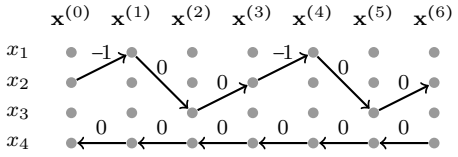


(b) An unfolding of $\mathcal{G}_R$ ($\mathcal{G}_R^6$)

the power R^n is defined by the conjunction of the following constraints:

$$\begin{array}{rcl} x_i - x_j & \leq & \mathrm{minw}^n(x_i^{(0)}, x_j^{(0)}) \\ x_i - x'_j & \leq & \mathrm{minw}^n(x_i^{(0)}, x_j^{(n)}) \\ x'_i - x_j & \leq & \mathrm{minw}^n(x_i^{(n)}, x_j^{(0)}) \\ x'_i - x'_j & \leq & \mathrm{minw}^n(x_i^{(n)}, x_j^{(n)}) \end{array} \tag{4.1}$$

where $\mathrm{minw}^n(x_i^{(p)}, x_j^{(q)}) = \min_{\ell \in \mathbb{N}} \left\{ \mathrm{minw}^n(x_i^{(p)}, x_j^{(q)}, \ell) \right\}$, and $\mathrm{minw}^n(x_i^{(p)}, x_j^{(q)}, \ell)$ is the weight of a minimal path of length ℓ between $x_i^{(p)}$ and $x_j^{(q)}$ in $\mathcal{G}_R^n$, for $p, q \in \{0, n\}$. As usual, we consider that $\min(\varnothing) = \infty$. When the length is not important, we denote a path between $x_i^{(p)}$ and $x_j^{(q)}$ as $x_i^{(p)} \xrightarrow{*} x_j^{(q)}$. The figure below (c) shows two such paths in an unfolding graph.



(c) Forward and backward paths in $\mathcal{G}_R^6$

We distinguish four types of paths in $\mathcal{G}_R^n$. A path $x_i^{(k)} \xrightarrow{*} x_j^{(\ell)}$ is said to be *odd forward* if $k = 0$ and $\ell = n$, *even forward* if $k = \ell = 0$, *odd backward* if $k = n$ and $\ell = 0$, and *even backward* if $k = \ell = n$. Observe that the symbols needed to represent an odd path have an odd number of edges, while the ones representing even paths have an even number of edges.

The first proof of Presburger-definability of transitive closures for difference bounds relations has been given by Comon and Jurski [CJ98], using an over-approximation, called *folded graph*, of the set of paths in the unfolding of a constraint graph. Their proof is based on the fact that only certain paths in this graph are relevant for the definition of the closed form, namely those paths that do not change direction while traversing vertices from the same SCC of the graph.

We give an alternative proof of this result [BIL09], based on the fact that these paths can be recognized by a finite *weighted automaton*, thus the weights of these paths can be captured by a quantifier-free Presburger formula that defines the Parikh image of this automaton[1]. Consequently, the weights of the minimal paths can be defined by a Presburger formula with one quantifier alternation.

4.1.1 Zigzag Automata

Consider an unfolding $\mathcal{G}_R^n$ of the constraint graph $\mathcal{G}_R$, for some $n > 0$. We recall the constraints (4.1) which define the closed form of R, using the minimal paths in $\mathcal{G}_R^n$ with endpoints in the set $\mathbf{x}^{(0)} \cup \mathbf{x}^{(n)}$. Each such path can be seen as a word over the finite alphabet of subgraphs of $\mathcal{G}_R$, and the set of paths between two distinguished vertices is the language of a finite (weighted) automaton, called *zigzag automaton* [BIL09]. Intuitively, a zigzag automaton reads, at step i in the computation, all edges between $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(i+1)}$ simultaneously. The weight of a transition fired by the zigzag automaton at step i is the sum of the weights of these edges. Each run of length n in a zigzag automaton recognizes a word consisting of a single path between two extremal vertices in $\mathcal{G}_R^n$, i.e. from the set $\mathbf{x}^{(0)} \cup \mathbf{x}^{(n)}$.

Formally, a *weighted automaton*[2] [Sch61] is a tuple $A = \langle \Sigma, \omega, Q, I, F, \Delta \rangle$, where Σ is a finite alphabet, $\omega : \Sigma \to \mathbb{Z}$ is a function associating integer weights to alphabet symbols, Q, I, F are the set of states, initial and final states, respectively, and $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation. The weight of a non-empty word $w = \sigma_1 \dots \sigma_n \in \Sigma^+$ is defined as $\omega(w) = \sum_{i=1}^{n} \omega(\sigma_i)$, and $\omega(\varepsilon) = 0$ is the weight of the empty word ε.

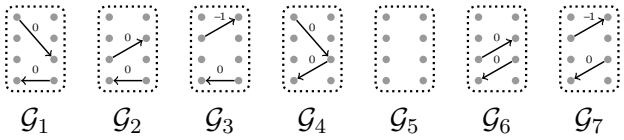
A *run* of A is a sequence $q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \dots \xrightarrow{\sigma_{n-1}} q_n$, denoted $q_0 \xrightarrow{\sigma_0 \cdots \sigma_{n-1}} q_n$. A word $w \in \Sigma^*$ is accepted by A if there exists a run $q_0 \xrightarrow{w} q_n$ such that $q_0 \in I$ and $q_n \in F$. We denote by $\mathcal{L}(A)$ the set of words accepted by A, i.e. the *language* of A. Moreover, we define the function $\mathrm{minw}_A : \mathbb{N} \to \mathbb{Z}$, where $\mathrm{minw}_A(n) = \min \{\omega(w) \mid w \in \mathcal{L}(A), |w| = n\}$.

The alphabet of the zigzag automaton is the set Σ_R of weighted graphs $\mathcal{G} = \langle \mathbf{x} \cup \mathbf{x}', \to, w \rangle$, where:

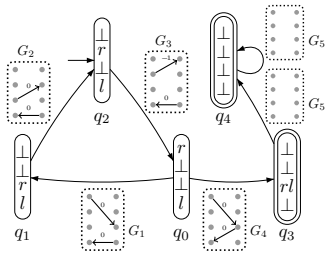
1. $x \xrightarrow{c} y$ if and only if $x - y \leq c$ occurs in $\Phi_{\sigma(R)}$, for all $x, y \in \mathbf{x} \cup \mathbf{x}'$,
2. the in-degree and out-degree of each node are at most 1, and
3. the difference between the number of edges from $\mathbf{x}$ to $\mathbf{x}'$ and the number of edges number of edges from $\mathbf{x}'$ to $\mathbf{x}$ is either -1, 0 or 1.

[1] See [VSS05] for a linear-time construction of this formula.

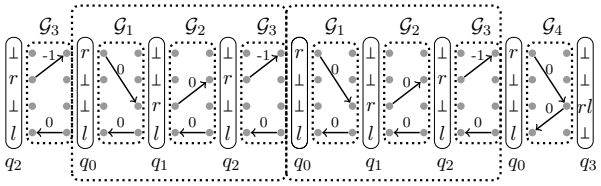
[2] We adopt a simplified version of the standard definition [Sch61], sufficient for our purposes.



(a) A subset of the zigzag alphabet Σ_R



(b) The zigzag automaton $\mathcal{A}_{24}^{ef}$



(c) A run of $\mathcal{A}_{24}^{ef}$ on the word $\mathcal{G}_3.(\mathcal{G}_1.\mathcal{G}_2.\mathcal{G}_3)^2.\mathcal{G}_4 \in \Sigma_R^+$.

Figure 4.3: A zigzag automaton for the relation $R \equiv x_2 - x_1' \leq -1 \wedge x_3 - x_2' \leq 0 \wedge x_1 - x_3' \leq 0 \wedge x_4' - x_4 \leq 0 \wedge x_3' - x_4 \leq 0$.

The weight of a graph symbol $\mathcal{G} \in \Sigma_R$ is the sum of the weights that occur on its edges, i.e. $\omega(\mathcal{G}) = \sum_{x \xrightarrow{c} y} c$. For an example, see Figure 4.3 (a).

The set of states of the zigzag automaton is $Q = \{\ell, r, \ell r, r\ell, \bot\}^N$, i.e. the set of N-tuples of symbols $\ell, r, \ell r, r\ell$ and $\bot$. Intuitively, these symbols capture the direction of the incoming and outgoing edges of the alphabet symbols: ℓ for a path traversing from right to left, r for a path traversing from left to right, ℓr for a right incoming and right outgoing path, $r\ell$ for a left incoming and left outgoing path, and $\bot$ when there are no incoming nor outgoing edges from that node. As a remark, the number of states of a zigzag automaton is bounded by 5^N. For example, Figure 4.3 (c) shows the use of states in a zigzag automaton.

The transition relation $\Delta \subseteq Q \times \Sigma_R \times Q$ is defined as follows. For all

$\mathbf{q}, \mathbf{q}' \in Q$ and $\mathcal{G} \in \Sigma_R$, we have $\mathbf{q} \xrightarrow{\mathcal{G}} \mathbf{q}'$, if and only if, for all $1 \leq i \leq N$:
- $\mathbf{q}_i = \ell$ iff $\mathcal{G}$ has one edge ending in x_i and no other edge involving x_i,
- $\mathbf{q}'_i = \ell$ iff $\mathcal{G}$ has one edge starting in x'_i and no other edge involving x'_i,
- $\mathbf{q}_i = r$ iff $\mathcal{G}$ has one edge starting in x_i and no other edge involving x_i,
- $\mathbf{q}'_i = r$ iff $\mathcal{G}$ has one edge ending in x'_i and no other edge involving x'_i,
- $\mathbf{q}_i = \ell r$ iff G has exactly two edges involving x_i, $x_j^{(\prime)} \to x_i \to x_k^{(\prime)}$,
- $\mathbf{q}'_i = r\ell$ iff G has exactly two edges involving x'_i, $x_j^{(\prime)} \to x'_i \to x_k^{(\prime)}$,
- $\mathbf{q}'_i \in \{\ell r, \bot\}$ iff G has no edge involving x'_i,
- $\mathbf{q}_i \in \{r\ell, \bot\}$ iff G has no edge involving x_i.

The zigzag automaton for R is a union of four types of automata. Formally, for each $i, j \in \{1, \ldots, N\}$ and $t \in \{of, ob, ef, eb\}$ (we use the abbreviations of=odd forward, ob=odd backward, ef=even forward and eb=even backward), the weighted automaton $\mathcal{A}^t_{ij} = \langle \Sigma_R, \omega, Q, I^t_{ij}, F^t_{ij}, \Delta \rangle$ recognizes those paths $x_i^{(p)} \xrightarrow{*} x_j^{(q)}$ of type t, where $p, q \in \{0, n\}$. The sets of initial and final states are defined according to the type of the automaton, as follows:

$$
\begin{array}{rcl}
I^{of}_{ij} & = & \{\mathbf{q} \mid \mathbf{q}_i = r \text{ and } \mathbf{q}_h \in \{\ell r, \bot\},\ \forall h \in \{1, \ldots, N\} \setminus \{i\}\} \\
F^{of}_{ij} & = & \{\mathbf{q} \mid \mathbf{q}_j = r \text{ and } \mathbf{q}_h \in \{r\ell, \bot\},\ \forall h \in \{1, \ldots, N\} \setminus \{j\}\} \\
I^{ob}_{ij} & = & \{\mathbf{q} \mid \mathbf{q}_i = \ell \text{ and } \mathbf{q}_h \in \{\ell r, \bot\},\ \forall h \in \{1, \ldots, N\} \setminus \{i\}\} \\
F^{ob}_{ij} & = & \{\mathbf{q} \mid \mathbf{q}_j = \ell \text{ and } \mathbf{q}_h \in \{r\ell, \bot\},\ \forall h \in \{1, \ldots, N\} \setminus \{j\}\} \\
I^{ef}_{ij} & = & \begin{cases} \{\mathbf{q} \mid \mathbf{q}_i = r,\ \mathbf{q}_j = \ell,\ \mathbf{q}_h \in \{\ell r, \bot\},\ \forall h \in \{1, \ldots, N\} \setminus \{i, j\}\} & \text{if } i \neq j \\ \{\mathbf{q} \mid \mathbf{q}_i = \ell r,\ \mathbf{q}_h \in \{\ell r, \bot\},\ \forall h \in \{1, \ldots, N\} \setminus \{i\}\} & \text{if } i = j \end{cases} \\
F^{ef}_{ij} & = & \{r\ell, \bot\}^N \\
I^{eb}_{ij} & = & \{\ell r, \bot\}^N \\
F^{eb}_{ij} & = & \begin{cases} \{\mathbf{q} \mid \mathbf{q}_i = \ell,\ \mathbf{q}_j = r,\ \mathbf{q}_h \in \{\ell r, \bot\},\ \forall h \in \{1, \ldots, N\} \setminus \{i, j\}\} & \text{if } i \neq j \\ \{\mathbf{q} \mid \mathbf{q}_i = r\ell,\ \mathbf{q}_h \in \{\ell r, \bot\},\ \forall h \in \{1, \ldots, N\} \setminus \{i\}\} & \text{if } i = j \end{cases}
\end{array}
$$

The following theorem [BIL09] relates the language of the zigzag automaton with the minimal paths in the unfolding graph of a relation $R \in \mathrm{DB}(\mathbf{x})$.

Theorem 6 ([BIL09])**.** *Let $R \in \mathrm{DB}(\mathbf{x})$ be a difference bounds relation. Then, for each $n \in \mathbb{N}_+$ such that $R^n \neq \varnothing$, and all $i, j \in \{1, \ldots, N\}$, the following hold:*

1. *each constraint graph $w \in \mathcal{L}(\mathcal{A}^{of}_{ij})$ consists of a path $x_i^{(0)} \xrightarrow{*} x_j^{(n)}$ and* $\mathrm{minw}^n(x_i^{(0)}, x_j^{(n)}) = \mathrm{minw}_{\mathcal{A}^{of}_{ij}}(n)$,
2. *each constraint graph $w \in \mathcal{L}(\mathcal{A}^{ob}_{ij})$ consists of a path $x_i^{(n)} \xrightarrow{*} x_j^{(0)}$ and* $\mathrm{minw}^n(x_i^{(n)}, x_j^{(0)}) = \mathrm{minw}_{\mathcal{A}^{ob}_{ij}}(n)$,
3. *each constraint graph $w \in \mathcal{L}(\mathcal{A}^{ef}_{ij})$ consists of a path $x_i^{(0)} \xrightarrow{*} x_j^{(0)}$ and* $\mathrm{minw}^n(x_i^{(0)}, x_j^{(0)}) = \mathrm{minw}_{\mathcal{A}^{ef}_{ij}}(n)$,

4. *each constraint graph* $w \in \mathcal{L}(\mathcal{A}^{eb}_{ij})$ *consists of a path* $x_i^{(n)} \xrightarrow{*} x_j^{(n)}$ *and* $\mathrm{minw}^n(x_i^{(n)}, x_j^{(n)}) = \mathrm{minw}_{\mathcal{A}^{eb}_{ij}}(n)$.

Proof. The points (1), (2), (3) and (4) are the Lemmas 4.6, 4.7, 4.3 and 4.4 from [BIL09], respectively. □

We are now ready to give the definition of the transitive closure R^+, of a difference bounds relation $R \in \mathrm{DB}(\mathbf{x})$, in Presburger arithmetic. Let $\mathcal{A}^t_{ij} = \langle \Sigma_R, \omega, Q, I^t_{ij}, F^t_{ij}, \Delta \rangle$ be a zigzag automaton for R, of one of the above types. We define first the set of weights $W_n(\mathcal{A}^t_{ij}) = \{\omega(w) \mid w \in \mathcal{L}(\mathcal{A}^t_{ij}) \text{ and } |w| = n\}$ by the formula below:

$$\Phi^t_{ij}(n, w) \equiv \exists \mathbf{y} \ . \bigvee_{q_f \in F^t_{ij}} \phi_{q_f}(\mathbf{y}) \wedge \sum_{p \xrightarrow{g} q \in \Delta} y_{pq} = n \wedge \sum_{p \xrightarrow{g} q \in \Delta} y_{pq} \cdot \omega(g) = w \quad (4.2)$$

where $\mathbf{y} = \left\{ y_{pq} \mid p \xrightarrow{g} q \in \Delta \right\}$ is the set of variables counting the number of times each transition of $\mathcal{A}^t_{ij}$ has been used in a run, and $\phi_q(\mathbf{y})$ defines the set of Parikh images of the runs[3], leading to a given state $q \in Q$. This is a quantifier-free formula of Presburger arithmetic that can be computed in time linear in the size of $\mathcal{A}^t_{ij}$, which is of the order of $2^{\mathcal{O}(N)}$ [VSS05].

The next step is to define minimal weight paths using Presburger formulae of the form $\exists^* \forall^*$: $\Theta^t_{ij}(n, w) \equiv \Phi^t_{ij}(n, w) \wedge \forall v < w \ . \ \neg \Phi^t_{ij}(n, v)$. Then the following formula defines the transitive closure of R:

$$TC_R(\mathbf{x}, \mathbf{x}') \equiv \exists n > 0 \ . \ \forall w \ \bigwedge_{i,j=1}^{N} \begin{pmatrix} \Phi^{of}_{ij}(n, w) \rightarrow x_i - y'_j \leq w \\ \Phi^{ob}_{ij}(n, w) \rightarrow x'_i - y_j \leq w \\ \Phi^{ef}_{ij}(n, w) \rightarrow x_i - y_j \leq w \\ \Phi^{eb}_{ij}(n, w) \rightarrow x'_i - y'_j \leq w \end{pmatrix} \quad (4.3)$$

Observe that the universally quantified variable w can be removed by replacing it with a term of the form $\sum_{p \xrightarrow{g} q \in \Delta} y_{pq} \cdot \omega(g)$ from (4.2). The resulting formula belongs thus to the $\exists^* \forall^*$ fragment of Presburger arithmetic. This argument is, by itself, sufficient to prove that the reachability problem for flat counter machines with cycles labeled with difference bounds relations is decidable. However, this proof of decidability does not establish the tight complexity bounds, which will be given next, in Section 4.4.

[3] The Parikh image of a run is the tuple giving the number of times each transition is fired along the run.

4.1.2 Parametric Difference Bounds Relations

We consider a generalization of the class of difference bounds relations, by allowing the bounds to be linear terms over a number of *parameter variables*. A parametric difference bounds constraint is defined as a finite conjunction of atomic propositions of the form $x - y \leq f(\mathbf{z})$, where $\mathbf{z}$ is a set of parameters, distinct from $\mathbf{x}$, and f is a linear term. Intuitively, the parameters $\mathbf{z}$ do not change during the execution of the counter machine. In this case, the transitive closure is not definable in Presburger arithmetic, as it was the case previously, but in the D^1 fragment of integer arithmetic (Section 3.3).

Let us consider the formulae Φ^t_{ij} (4.2) defining the weights of paths of length n in a zigzag automaton, whose weights are now described by linear terms over the parameters $\mathbf{z}$. Due to the occurrence of non-linear terms $y_{pq} \cdot \omega(g)$ in Φ^t_{ij}, in which the variables y_{pq} are multiplied by parameters $z \in \mathbf{z}$, the minimal weights cannot be directly defined by introducing a universal quantifier, as in the case of standard difference bounds relations. We address this problem by finding the elementary cycles of optimal ratio between their weight and length in the weighted automata $\mathcal{A}^t_{ij} = \langle \Sigma_R, \omega, Q, I^t_{ij}, F^t_{ij}, \Delta \rangle$, respectively. These cycles are usually called *critical cycles* in the literature.

We apply the following transformation to each disjunct from (4.2), i.e. for each $q_f \in F^t_{ij}$. Let $\mathbf{y} = \{y_1, \ldots, y_m\}$ be a renaming of the existentially quantified variables, and if y_i is the renaming of y_{pq}, then let ω_i denote the term $\omega(g)$, where $p \xrightarrow{g} q$ is the unique[4] transition between p and q in the zigzag automaton under consideration. Since $\phi_{q_f}(\mathbf{y}) \wedge \sum_{p \xrightarrow{g} q \in \Delta} y_{pq} = n$ is an open Presburger formula, it is either false or it defines a non-empty semilinear set [GS66], defined by a finite disjunction of formulae as:

$$\begin{pmatrix} y_1 \\ \vdots \\ y_m \\ n \end{pmatrix} = \begin{pmatrix} a_{01} \\ \vdots \\ a_{0m} \\ b_0 \end{pmatrix} + \begin{pmatrix} a_{11} \\ \vdots \\ a_{1m} \\ b_1 \end{pmatrix} \lambda_1 + \ldots + \begin{pmatrix} a_{k1} \\ \vdots \\ a_{km} \\ b_k \end{pmatrix} \lambda_k$$

for some new existentially quantified variables $\lambda_1, \ldots, \lambda_k$ ranging over $\mathbb{N}$, and constants $a_{ij}, b_i \in \mathbb{Z}$. Since $w = \sum_{p \xrightarrow{g} q \in \Delta} y_{qr} \cdot \omega(g)$, we obtain, for each disjunct of (4.2):

$$\begin{pmatrix} n \\ w \end{pmatrix} = \begin{pmatrix} b_0 \\ \sum_{j=1}^m a_{0j}\omega_j \end{pmatrix} + \begin{pmatrix} b_1 \\ \sum_{j=1}^m a_{1j}\omega_j \end{pmatrix} \lambda_1 + \ldots + \begin{pmatrix} b_k \\ \sum_{j=1}^m a_{kj}\omega_j \end{pmatrix} \lambda_k$$

[4] The uniqueness follows from the definition of the transition table for zigzag automata.

Since $n > 0$, it must be that $b_i > 0$, for all $1 \leq i \leq k$. Otherwise, if some $b_i < 0$ we can obtain a negative value for n by increasing λ_i sufficiently. On the other hand, if $b_i = 0$, for some $i = 1, \ldots, k$, there would be an infinite number of weights w corresponding to the same path of length w, resulting in a contradiction. Consider now the following formulae, for all $i = 1, \ldots, k$:

$$\kappa_i(\mathbf{z}) \equiv \bigwedge_{p=1}^{k} \frac{\sum_{j=1}^{m} a_{ij}\omega_j}{b_i} \leq \frac{\sum_{j=1}^{m} a_{pj}\omega_j}{b_p}$$

Intuitively, $\kappa_i(\mathbf{z})$ defines the set of valuations of the parameters $\mathbf{z}$ that make the i-th cycle critical in the weighted automaton $\mathcal{A}_{ij}^t$ under consideration. It is easy to see that $\bigvee_{i=1}^{k} \kappa_i$ covers the set $\mathbb{Z}^{\mathbf{z}}$ of parameter valuations. We perform a case split, in which the i-th case corresponds to a choice of parameter values that satisfy κ_i, in which case the minimal weight w for a given length n is defined by the following:

$$\begin{pmatrix} n \\ w \end{pmatrix} = \begin{pmatrix} b_0 \\ \sum_{j=1}^{m} a_{0j}\omega_j \end{pmatrix} + \sum_{1 \leq \ell \leq k}^{\ell \neq i} \begin{pmatrix} b_\ell \\ \sum_{j=1}^{m} a_{\ell j}\omega_j \end{pmatrix} r_\ell + \begin{pmatrix} b_i \\ \sum_{j=1}^{m} a_{ij}\omega_j \end{pmatrix} \Big(\lambda_i + \sum_{1 \leq \ell \leq k}^{\ell \neq i} q_\ell b_\ell\Big)$$

where q_ℓ and r_ℓ are the quotient and the remainder of λ_ℓ divided by b_i, for all $\ell \in \{1, \ldots, i-1, i+1, \ldots, k\}$. This is because b_i can be subtracted from any λ_ℓ, $\ell \neq i$, at most q_ℓ times, by adding at the same time $q_\ell b_\ell$ to λ_i, and without changing n. Observe that each r_ℓ can be replaced by constants in the range $0, \ldots b_i - 1$. Finally, substituting the term $\lambda_i + \sum_{1 \leq \ell \leq k}^{\ell \neq i} q_\ell b_\ell$ with a fresh variable m, we obtain a formula of D^1 with parameter m.

To sum up, the transitive closure of a parametric difference bounds relation can be defined by a (quantifier-free) formula of D^1 . Since D^1 is a decidable fragment of integer arithmetic (Section 3.3), we obtain the following theorem:

Theorem 7 ([BIL09])**.** *The reachability problem is decidable for flat counter machines with cycles labeled by parametric difference bounds relations.*

As in the non-parametric case, this proof of decidability does not provide nice complexity bounds for the reachability problem. For the time being, the complexity of the reachability problem for parametric difference bounds constraints has not been given the needed attention, thus we add it to the list of open problems for this chapter (Section 4.7).

4.2 Octagonal Relations

Octagonal constraints (also known as Unit Two Variables Per Inequality or UTVPI, for short) appear in the context of abstract interpretation where

they have been extensively studied as an abstract domain [Min06]. Since octagons are a generalization of difference bounds constraints, most results from this section extend the results from Section 4.1. In particular, we prove that the transitive closures of relations defined by octagonal constraints are Presburger-definable [BGI09], and obtain the decidability of the reachability problem for flat counter machines whose cycles are labeled by such relations.

Definition 4. *A formula $\phi(\mathbf{x})$ is an* octagonal constraint *if it is a finite conjunction of terms of the form $\pm x_i \pm x_j \leq c_{ij}$, where $c_{ij} \in \mathbb{Z}$, for all $1 \leq i, j \leq N$. A relation $R \subseteq \mathbb{Z}^{\mathbf{x}} \times \mathbb{Z}^{\mathbf{x}}$ over a set of variables $\mathbf{x}$ is an* octagonal relation *if it can be defined by an octagonal constraint $\phi(\mathbf{x}, \mathbf{x}')$. We denote by $\mathrm{OCT}(\mathbf{x})$ the class of octagonal relations over the variables $\mathbf{x}$.*

Given a set of variables $\mathbf{x} = \{x_1, \ldots, x_N\}$, an octagonal constraint $\phi(\mathbf{x})$ is usually represented by a difference bounds constraints $\overline{\phi}(\mathbf{y})$, where $\mathbf{y} = \{y_1, \ldots, y_{2N}\}$, y_{2i-1} stands for $+x_i$ and y_{2i} stands for $-x_i$, with the implicit requirement that $y_{2i-1} = -y_{2i}$, for each $1 \leq i \leq N$. Observe that this implicit condition cannot be directly represented as a difference bounds constraint. Formally, we have:

$$
\begin{array}{lcl}
x_i - x_j \leq c \text{ occurs in } \phi & \Leftrightarrow & y_{2i-1} - y_{2j-1} \leq c,\ y_{2j} - y_{2i} \leq c \text{ occur in } \overline{\phi} \\
-x_i + x_j \leq c \text{ occurs in } \phi & \Leftrightarrow & y_{2j-1} - y_{2i-1} \leq c,\ y_{2i} - y_{2j} \leq c \text{ occur in } \overline{\phi} \\
-x_i - x_j \leq c \text{ occurs in } \phi & \Leftrightarrow & y_{2i} - y_{2j-1} \leq c,\ y_{2j} - y_{2i-1} \leq c \text{ occur in } \overline{\phi} \\
x_i + x_j \leq c \text{ occurs in } \phi & \Leftrightarrow & y_{2i-1} - y_{2j} \leq c,\ y_{2j-1} - y_{2i} \leq c \text{ occur in } \overline{\phi}
\end{array}
$$

In order to handle the $\mathbf{y}$ variables in the following, we define $\bar{i} = i - 1$, if i is even, and $\bar{i} = i + 1$ if i is odd. Obviously, we have $\bar{\bar{i}} = i$, for all $i \in \mathbb{N}_+$. For example, the octagonal constraint $x_1 + x_2 = 3$ is represented as $y_1 - y_4 \leq 3 \wedge y_2 - y_3 \leq -3$, with the implicit constraints $y_1 + y_2 = y_3 + y_4 = 0$.

An octagonal constraint $\phi(\mathbf{x})$ is equivalently represented by the $2N \times 2N$ DBM $M_{\overline{\phi}}$, corresponding to $\overline{\phi}(\mathbf{y})$. A $2N \times 2N$ DBM M is *coherent* iff $M_{ij} = M_{\overline{j}\overline{i}}$ for all $1 \leq i, j \leq 2N$. This property is needed because an atomic proposition $x_i - x_j \leq c_{ij}$, $1 \leq i, j \leq N$, can be represented as both $y_{2i-1} - y_{2j-1} \leq c_{ij}$ and $y_{2j} - y_{2i} \leq c_{ij}$. Dually, a coherent $2N \times 2N$ DBM M corresponds to the following octagonal constraint:

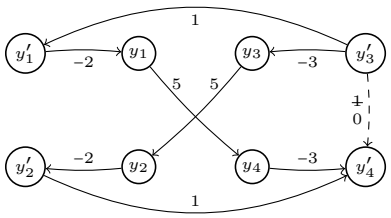
$$\Omega_M \equiv \bigwedge_{i,j=1}^{N} x_i - x_j \leq M_{2i-1,2j-1} \wedge \bigwedge_{i,j=1}^{N} x_i + x_j \leq M_{2i-1,2j} \wedge \bigwedge_{i,j=1}^{N} -x_i - x_j \leq M_{2i,2j-1}$$

A coherent DBM M is said to be *octagonal-consistent* if and only if Ω_M is consistent.

Definition 5. *An octagonal-consistent coherent $2N \times 2N$ DBM M is said to be* tightly closed *if and only if it is closed and $M_{ij} \leq \lfloor \frac{M_{i\bar{i}}}{2} \rfloor + \lfloor \frac{M_{\bar{j}j}}{2} \rfloor$, for all $1 \leq i, j \leq N$.*

The last condition from Definition 5 ensures that the knowledge induced by the implicit conditions $y_i + y_{\bar{i}} = 0$ has been propagated through the DBM. Since $2y_i = y_i - y_{\bar{i}} \leq M_{i\bar{i}}$ and $-2y_j = y_{\bar{j}} - y_j \leq M_{\bar{j}j}$, we have $y_i \leq \lfloor \frac{M_{i\bar{i}}}{2} \rfloor$ and $-y_j \leq \lfloor \frac{M_{\bar{j}j}}{2} \rfloor$, which implies $y_i - y_j \leq \lfloor \frac{M_{i\bar{i}}}{2} \rfloor + \lfloor \frac{M_{\bar{j}j}}{2} \rfloor$, thus $M_{ij} \leq \lfloor \frac{M_{i\bar{i}}}{2} \rfloor + \lfloor \frac{M_{\bar{j}j}}{2} \rfloor$ must hold, if M is supposed to be the most precise DBM representation of an octagonal constraint. If $j = \bar{i}$ in the previous, we obtain $M_{i\bar{i}} \leq 2\lfloor \frac{M_{i\bar{i}}}{2} \rfloor$, implying that $M_{i\bar{i}}$ is necessarily even, if M is tightly closed.

Example 3. *Consider the octagonal relation $R(x_1, x_2, x_1', x_2') \equiv x_1 + x_2 \leq 5 \wedge x_1' - x_1 \leq -2 \wedge x_2' - x_2 \leq -3 \wedge x_2' - x_1' \leq 1$. Its difference bounds representation is $\overline{R}(\mathbf{y}, \mathbf{y}') \Leftrightarrow y_1 - y_4 \leq 5 \wedge y_3 - y_2 \leq 5 \wedge y_1' - y_1 \leq -2 \wedge y_2 - y_2' \leq -2 \wedge y_3' - y_3 \leq -3 \wedge y_4 - y_4' \leq -3 \wedge y_3' - y_1' \leq 1 \wedge y_2' - y_4' \leq 1$, where $\mathbf{y} = \{y_1, \ldots, y_4\}$. Figure 4.4(a) shows the graph representation $\mathcal{G}_R$. Note that the implicit constraint $y_3' - y_4' \leq 1$, represented by a dashed edge in Figure 4.4(a), is not tight. The tightening step replaces the bound 1, crossed in Figure 4.4(a), with 0. Figure 4.4(b) shows the tightly closed DBM representation of R, denoted M_R^t.* ■



$\mathcal{G}_{\overline{R}}$

	y_1	y_2	y_3	y_4	y_1'	y_2'	y_3'	y_4'
y_1	0	∞	∞	5	∞	∞	∞	2
y_2	∞	0	∞	∞	∞	−2	∞	−1
y_3	∞	5	0	∞	∞	3	∞	4
y_4	∞	∞	∞	0	∞	∞	∞	−3
y_1'	−2	∞	∞	3	0	∞	∞	0
y_2'	∞	∞	∞	∞	∞	0	∞	1
y_3'	−1	2	−3	4	1	0	0	0
y_4'	∞	∞	∞	∞	∞	∞	∞	0

$M_{\overline{R}}^t$

Figure 4.4: Graph and matrix representation of the difference bounds representation $\overline{R}(\mathbf{y}, \mathbf{y}')$ of an octagonal relation $R(\mathbf{x}, \mathbf{x}') \equiv x_1 + x_2 \leq 5 \ \wedge \ x_1' - x_1 \leq -2 \ \wedge \ x_2' - x_2 \leq -3 \ \wedge \ x_2' - x_1' \leq 1$.

The following theorem [BHZ08] provides a cost-effective way of testing octagonal-consistency and computing the tight closure of a coherent DBM. Moreover, it shows that the tight closure of a given DBM is unique. Thus tightly closed DBMs are canonical representations for octagonal constraints, that can be computed in polynomial time from any non-canonical representation thereof.

Theorem 8 ([BHZ08]). *Let M be a coherent $2N \times 2N$ DBM. Then M is octagonal-consistent iff M is consistent and $\lfloor \frac{M^*_{i\bar{i}}}{2} \rfloor + \lfloor \frac{M^*_{\bar{i}i}}{2} \rfloor \geq 0$, for all $1 \leq i \leq 2N$. Moreover, if M is octagonal-consistent, the tight closure of M is the DBM M^t defined as:*

$$M^t_{ij} = \min \left\{ M^*_{ij}, \left\lfloor \frac{M^*_{i\bar{i}}}{2} \right\rfloor + \left\lfloor \frac{M^*_{\bar{j}j}}{2} \right\rfloor \right\}$$

for all $1 \leq i, j \leq 2N$ where M^* is the closure of M.

This theorem is the key for proving that the transitive closure of any octagonal relation is Presburger-definable. Consider a relation $R \in \mathrm{OCT}(\mathbf{x})$ and its corresponding difference bounds representation $\overline{R} \in \mathrm{DB}(\mathbf{y})$. For any $n > 0$, the unfolding graph $\mathcal{G}^n_{\overline{R}}$ defines the minimal weight paths that define the n-th power of $\overline{R}$. By applying the above theorem, we can compute the tightening of this graph, which defines the n-th power of R, for any $n > 0$. The most difficult part is the computation of the values $M^*_{i\bar{i}}$, for any $i = 1, \ldots, 2N$.

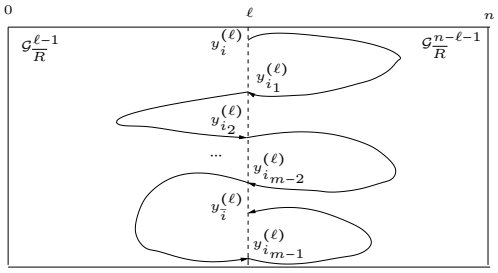


Figure 4.5: Minimal weight path $y_i^{(\ell)} \xrightarrow{*} y_{\bar{i}}^{(\ell)}$ in $\mathcal{G}^n_{\overline{R}}$.

To understand this point, consider Figure 4.5 below. Assume in the following that $\mathcal{G}^n_{\overline{R}}$ has no cycles of negative weight. The absence of negative cycles can be checked a-priori using Theorem 6 (Section 4.1). Since we are aiming at computing minimal weight paths, it is sufficient to consider acyclic paths only[5]. For a fixed $0 < \ell < n$, an acyclic path between the nodes $y_i^{(\ell)}$ and $y_{\bar{i}}^{(\ell)}$, for some $1 \leq i \leq 2N$, can be decomposed in at most $2N - 1$ segments $y_{i_0}^{(\ell)} \xrightarrow{*} y_{i_1}^{(\ell)}, y_{i_1}^{(\ell)} \xrightarrow{*} y_{i_2}^{(\ell)}, \ldots, y_{i_{m-1}}^{(\ell)} \xrightarrow{*} y_m^{(\ell)}$, where $i_0 = i$ and $i_m = \bar{i}$. These segments start and end in $\mathbf{y}^{(\ell)}$, but do not intersect with $\mathbf{y}^{(\ell)}$, other than in the beginning and in the end. Moreover, if the path considered is

[5] If a path has a cycle of a positive weight, it cannot be minimal.

of minimal weight, these segments are of minimal weight as well. Observe that, the paths involving nodes from $\mathbf{y}^{(\leq \ell)}$ connect the terminal nodes of $\mathcal{G}_{\overline{R}}^{\ell-1}$, whereas the paths involving nodes from $\mathbf{y}^{(\geq \ell)}$ connect the initial nodes of $\mathcal{G}_{\overline{R}}^{n-\ell-1}$. We have the following relations:

$$\begin{array}{lcl}
(M_n^*)_{i_j^{(\ell)}, i_{j+1}^{(\ell)}} & = & \min^{\ell-1}\left(y_{i_j}^{(\ell)}, y_{i_{j+1}}^{(\ell)}\right) \text{ if the path } y_{i_j}^{(\ell)} \xrightarrow{*} y_{i_{j+1}}^{(\ell)} \text{ involves only } \mathbf{y}^{(\leq \ell)} \\
(M_n^*)_{i_j^{(\ell)}, i_{j+1}^{(\ell)}} & = & \min^{n-\ell-1}\left(y_{i_j}^{(0)}, y_{i_{j+1}}^{(0)}\right) \text{ if the path } y_{i_j}^{(\ell)} \xrightarrow{*} y_{i_{j+1}}^{(\ell)} \text{ involves only } \mathbf{y}^{(\geq \ell)} \\
(M_n^*)_{i^{(\ell)}, \overline{i^{(\ell)}}} & = & \sum_{j=0}^{m-1} (M_n^*)_{i_j^{(\ell)}, i_{j+1}^{(\ell)}}
\end{array}$$

where M_n is the incidence matrix (DBM) of the weighted graph $\mathcal{G}_{\overline{R}}^n$. One of the results of Section 4.1 is that the minimal weight functions $\min^{\ell-1}(y_{i_j}^{(\ell)}, y_{i_{j+1}}^{(\ell)})$ and $\min^{n-\ell-1}(y_{i_j}^{(0)}, y_{i_{j+1}}^{(0)})$ are Presburger-definable, respectively. Thus, the matrix coefficients $(M_n^*)_{i^{(\ell)}, \overline{i^{(\ell)}}}$ are also Presburger-definable. Since the integer half functions can be defined as $\left\lfloor \frac{u}{2} \right\rfloor = v \Leftrightarrow 2v \leq u \leq 2v+1$, one can effectively use Theorem 8 to define the transitive closure of an octagonal relation in Presburger arithmetic.

As a by-product, we can consider a generalization of octagonal relations, in which the bounds are linear terms over a set of parameters, i.e. relations defined as finite conjunctions of atomic propositions of the form $\pm x \pm y \leq f(\mathbf{z})$, where $x, y \in \mathbf{x} \cup \mathbf{x}'$, $\mathbf{z}$ is disjoint from $\mathbf{x}$, and f is a linear term. In this case, the minimal weight functions are definable in the D^1 fragment of integer arithmetic, thus the transitive closure of a parametric octagonal relation is also definable in D^1 (Section 3.3). The following theorem summarizes the result of this section:

Theorem 9 ([BGI09])**.** *The reachability problem is decidable for flat counter machines with cycles labeled by parametric octagonal relations.*

Again, this theorem does not provide nice complexity bounds for the reachability and termination problems for flat counter machines. These points require further developments, and are treated in detail in Sections 4.4 and 4.6, respectively. The precise complexities of these problems for parametric octagonal relations are listed as open problems in Section 4.7.

4.3 Periodic Relations

The previous sections (4.1 and 4.2) established decidability results for the reachability problem for classes of flat counter machines with cycles labeled

by (parametric) difference bounds (Theorem 7) and octagonal (Theorem 9) relations. These results are obtained by a definition of the transitive closures of these relations into decidable fragments of integer arithmetic, such as Presburger arithmetic, or D^1 (Chapter 3, Section 3.3). In order to derive tight complexity bounds for this problem, we noticed that the behavior of a power sequence $\{R^n\}_{n=0}^{\infty}$, when $R \in \mathrm{OCT}(\mathbf{x})$, is *periodic*, when looking at the constants that occur in the definition of the successive powers $R^0, R^1, R^2, \ldots$ etc.

In this section, we give a formal proof of periodicity, based on a classical result from the literature on tropical (max-plus) algebra. We develop this result further into an estimation of the prefix and period of these power sequences, that allows to nail the reachability problem to NP-complete (Section 4.4) and the termination problem to PTIME (Section 4.6), for the case of octagonal relations, without parameters.

We consider infinite sequences $\{s_k\}_{k=0}^{\infty}$ in $\mathbb{Z}_{\pm\infty}$, with the following extension of addition: (i) for all $x \in \mathbb{Z}_{\infty}$, $x+\infty = \infty+x = \infty$, and (ii) for all $x \in \mathbb{Z}_{\pm\infty}$, $x + (-\infty) = -\infty + x = -\infty$. A sequence $\{s_k\}_{k=0}^{\infty}$ is an *arithmetic progression* if there exists a constant $\lambda \in \mathbb{Z}_{\pm\infty}$, called *rate*, such that $s_{k+1} = s_k + \lambda$, for all $k \geq 0$. A generalization of this notion are *periodic* sequences, defined below.

Definition 6. *An infinite sequence $\{s_k\}_{k=0}^{\infty}$, where $s_k \in \mathbb{Z}_{\pm\infty}$, for all $k \geq 0$, is said to be* periodic *if and only if there exist integer constants $b \geq 0$, $c > 0$ and $\lambda_0, \ldots, \lambda_{c-1} \in \mathbb{Z}_{\pm\infty}$ such that $s_{b+(k+1)c+i} = s_{b+kc+i} + \lambda_i$, for all $k \geq 0$ and all $i \in [c]$. The smallest b, c and λ_i are called the* prefix, period *and* rates *of the sequence.*

Note that an arithmetic progression is a periodic sequence with prefix 0 and period 1. In the following, we consider sequences of square matrices and say that an infinite sequence $\{M_k\}_{k=0}^{\infty}$ of matrices $M_k \in \mathbb{Z}_{\pm\infty}^{n\times n}$ is *periodic* if every sequence $\{(M_k)_{ij}\}_{k=0}^{\infty}$ is periodic, for all $i, j \in [n]$. The next lemma provides a characterization of periodicity for a sequence of matrices, with an estimation of its prefix and period.

Lemma 1. *A a sequence of $\mathbb{Z}_{\pm\infty}^{n\times n}$ matrices $\{M_k\}_{k=0}^{\infty}$ is periodic iff there exist integers $b \geq 0$, $c > 0$ and matrices $\Lambda_0, \ldots, \Lambda_{c-1} \in \mathbb{Z}_{\pm\infty}^{n\times n}$ such that:*

$$\forall k \geq 0 \forall i \in [c] \ . \ M_{b+(k+1)c+i} = M_{b+kc+i} + \Lambda_i \ .$$

If, moreover, b_{ij} and c_{ij} are the prefix and period of the sequence $\{(M_k)_{ij}\}_{k=0}^{\infty}$, then $b = \max_{1\leq i,j\leq n}(b_{ij})$, $c = \mathrm{lcm}_{1\leq i,j\leq n}(c_{ij})$ are the smallest such integers.

Proof. See [BIK13, Lemma 1]. □

Let us focus now on sequences of matrices that represent the power sequences $\{R^k\}_{k=0}^{\infty}$, where $R \in \mathrm{OCT}$. Formally, let $\sigma : \mathrm{OCT}_{\mathbf{x}} \to \mathbb{Z}_{\infty}^{4N\times 4N} \cup \{[-\infty]_{4N}\}$ be the bijection that maps each consistent relation R into its canonical DBM $\sigma(R) \in \mathbb{Z}_{\infty}^{4N\times 4N}$ and the inconsistent relation into the matrix $[-\infty]_{4N}$, which has $-\infty$ everywhere. Then R is said to be *periodic* if the matrix sequence $\{\sigma(R^k)\}_{k=0}^{\infty}$ is periodic. If every relation in a certain class is periodic, we call that class periodic as well.

Example 4. *Consider the octagonal relation $R \subseteq \mathbb{Z}^{\{x,y\}} \times \mathbb{Z}^{\{x,y\}}$ defined by the formula $x' = y + 1 \wedge y' = x$, where for all $\ell \in \mathbb{N}$:*

$\sigma(R^{2\ell+1}) =$

	x	y	x'	y'
x	0	∞	∞	ℓ
y	∞	0	$-\ell-1$	∞
x'	∞	$\ell+1$	0	∞
y'	$-\ell$	∞	∞	0

$\sigma(R^{2\ell+2}) =$

	x	y	x'	y'
x	0	∞	$-\ell-1$	∞
y	∞	0	∞	$-\ell-1$
x'	$\ell+1$	∞	0	∞
y'	∞	$\ell+1$	∞	0

Then $\{\sigma(R^k)\}_{k=0}^{\infty}$ is periodic with prefix $b = 1$ and period $c = 2$, where:

$$\Lambda_0 = \begin{vmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{vmatrix} \qquad \Lambda_1 = \begin{vmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{vmatrix}$$

One of the results is that the class OCT is periodic. The proof of this fact relies essentially on the fact that the class DB is periodic, and uses (a variant of) Theorem 8 to generalize this result from difference bounds to octagonal relations. In the next section we give a generic nondeterministic decision procedure for the reachability problem of flat counter machines with cycles labeled by relations from a periodic class $\mathcal{R}$. Moreover, we identify certain conditions under which each branch of the procedure terminates in polynomial time, providing an NP upper bound for the reachability problem.

4.4 An Algorithm for the Reachability Problem

In general, the decision procedures for the reachability problem for flat counter machines rely on *acceleration* [Boi99, FL02b], which is defining the transitive closure of the relations that occur on the cycles of these machines by formulae from the quantifier-free fragment of Presburger arithmetic. To show that these reachability problems belong to the class NP, it is essential to build these QFPA formulae in polynomial time.

For the sake of simplicity, we explain the idea of a nondeterministic algorithm (Algorithm 1) for the reachability problem on the flat counter machine below:

$$q_{\text{init}} \xrightarrow{I(\mathbf{x})} \overset{\phi(\mathbf{x},\mathbf{x}')}{\underset{q}{\circlearrowright}} \xrightarrow{F(\mathbf{x})} q_{\text{fin}} \tag{4.4}$$

where $I(\mathbf{x})$ and $F(\mathbf{x})$ are QFPA formulae and $\phi(\mathbf{x},\mathbf{x}')$ is an octagonal constraint defining a relation $R \in \mathrm{OCT}_{\mathbf{x}}$, where $\mathbf{x} = \{x_1, \ldots, x_N\}$ are variables.

Let us assume for now that this relation is periodic. The algorithm guesses candidate values for the prefix $b \geq 0$ and period $c > 0$ of R (line 2), computes a candidate rate Λ (line 3), and checks if b, c and Λ satisfy the following condition (line 4):

$$\textsc{Ind}(B,C,\Lambda): \quad \forall n \geq 0 \;.\; \sigma\left(\sigma^{-1}(B + n\cdot\Lambda) \circ \sigma^{-1}(C)\right) = B + (n+1)\cdot\Lambda \tag{4.5}$$

where B, C and Λ are square matrices of equal dimension, in our case $B = \sigma(R^b)$, $C = \sigma(R^c)$ and Λ is such that $\sigma(R^b) + \Lambda = \sigma(R^{b+c})$. Intuitively, this means that b, c and Λ are valid choices for the prefix, period and rate of the sequence of matrices $\left\{\sigma(R^k)\right\}_{k=0}^{\infty}$, in the sense of Lemma 1.

In case the reachability problem for M has a positive answer, i.e. there exists a run from q_{init} to q_{fin}, two cases are possible. Either the number of iterations of the cycle is (i) strictly smaller than b, or (ii) between $b + nc$ and $b + (n+1)c$, for some $n \geq 0$. The first case is captured by the QFPA formula $\phi^{<b}$ (line 6), where $\Omega(\sigma(R^i))$ is the canonical octagonal constraint representing the relation R^i.

The second case is encoded by the QFPA formula $\phi^{\geq b}$ (line 8). Here $k \notin \mathbf{x}$ is a parameter variable and by $\mathbb{Z}[k]_\infty$ we denote the set of univariate linear terms of the form $a \cdot k + b$, with $a, b \in \mathbb{Z}_\infty$. Also $\mathbb{Z}[k]_\infty^{m\times m}$ denotes the set of $m \times m$ square matrices of such terms. With these notations, ς is a mapping of matrices $M[k] \in \mathbb{Z}[k]_\infty^{4N\times 4N}$ into parametric octagonal constraints consisting of atomic propositions of the form $\pm x \pm y \leq a \cdot k + b$, defined in the same way as the octagonal constraint $\Omega(M)$ is defined for a matrix $M \in \mathbb{Z}_\infty^{4N\times 4N}$. Moreover, ς satisfies the following condition:

$$\forall M \in \mathbb{Z}[k]_\infty^{4N\times 4N} \forall n \in \mathbb{N} \;.\; \varsigma(M)(n) = \sigma^{-1}(M[n/k]) \tag{4.6}$$

The final step is checking the satisfiability of the disjunction $\phi^{<b} \vee \phi^{\geq b}$ (line 9). If the formula produced by a nondeterministic branch of the algorithm is satisfiable, the reachability question has a positive answer. Otherwise, if no branch produces a satisfiable formula, the reachability question has a negative answer.

Algorithm 1 algorithm for the reachability problem (4.4)

input: $M = \langle \mathbf{x}, \{q_{\text{init}}, q, q_{\text{fin}}\}, q_{\text{init}}, q_{\text{fin}}, \Delta \rangle$ of the form (4.4), where $\mathbf{x} = \{x_1, \ldots, x_N\}$

output: YES if and only if M has a run from q_{init} to q_{fin}

let R be the relation defined by $\phi(\mathbf{x}, \mathbf{x}')$
chose $b \geq 0$ and $c > 0$
let $\Lambda \in \mathbb{Z}_\infty^{4N \times 4N}$ be a matrix such that $\sigma(R^b) + \Lambda = \sigma(R^{b+c})$
if $\text{IND}(\sigma(R^b), \sigma(R^c), \Lambda)$ **then**
    chose $i \in [b]$
    $\phi^{<b} \leftarrow I(\mathbf{x}) \wedge \Omega(\sigma(R^i)) \wedge F(\mathbf{x}')$
    chose $j \in [c]$
    $\phi^{\geq b} \leftarrow k \geq 0 \wedge I(\mathbf{x}) \wedge \varsigma(\sigma(R^{b+j}) + k \cdot \Lambda) \wedge F(\mathbf{x}')$
    if $\phi^{<b} \vee \phi^{\geq b}$ is satisfiable **then**
        return YES
fail

To prove that the class of reachability problems for these counter machines is in NP, it is enough to show that, for any machine M of the form (4.4), each branch of Algorithm 1 terminates in $\text{PTIME}(|M|)$. For this, the matrices $\sigma(R^c)$, $\sigma(R^i)$ and $\sigma(R^{b+j})$ must be computable in $\text{PTIME}(|R|)$, for all $i = 0, \ldots, b$ and $j = 0, \ldots, c$ and the condition $\text{IND}(\sigma(R^b), \sigma(R^c), \Lambda)$ (4.5) must be decidable in $\text{NPTIME}(|R|)$. Under these conditions, the QFPA formulae $\phi^{<b}$ and $\phi^{\geq b}$ are of polynomial size in $|I|+|R|+|F|$, and the satisfiability of their disjunction is decidable in $\text{NPTIME}(|I| + |R| + |F|)$.

The following theorem generalizes this argument to arbitrary flat counter machines by giving sufficient conditions under which the class of reachability problems for flat counter machines with cycles labeled by octagonal relations is NP-complete. In the following, we denote this class of problems by REACHFLAT(OCT).

Theorem 10. REACHFLAT(OCT) *is* NP*-complete if there exists a constant* d*, such that the following hold, for each relation* $R \in \text{OCT}$*:*

1. $|R^n| = \mathcal{O}((|R| \cdot \log n)^d)$*, for all* $n > 0$*,*
2. R *is periodic with prefix and period of the order of* $2^{\mathcal{O}(|R|^d)}$*.*

Proof. First, we show that the condition $\text{IND}(B, C, \Lambda)$ is decidable in nondeterministic polynomial time, by reduction to the satisfiability of a QFPA formula. The proof relies on a symbolic tight closure algorithm, which builds such a formula using a cubic number of steps [BIK13, Lemma 2]. For the rest of the proof, see [BIK13, Theorem 2]. □

4.5 The Exponential Periodicity of Octagons

In order to apply Theorem 10 we start by proving that the first assumption from its statement holds for each octagonal relation $R \in \mathrm{OCT}$, namely that the size of the binary representation of the n-th power R^n is bounded by a polynomial function with arguments $|R|$ and $\log n$. In this case, the binary size of an exponentially large power R^n, where $n = 2^{\mathcal{O}(|R|^d)}$ and d is a constant, is bounded by a polynomial in $|R|$. Moreover, such powers can be computed in a polynomial number of steps, using exponentiation by squaring. This is essential in proving that each branch of the nondeterministic Algorithm 1 terminates in polynomial time, and also in the generalization of this reasoning to arbitrary flat CM with cycles labeled by octagonal constraints (Theorem 10).

The core of the proof is showing periodicity of difference bounds relations and establishing the upper bounds for the prefix and period of sequence $\{\sigma(R^k)\}_{k=0}^{\infty}$, where $R \in \mathrm{DB}$. The main idea is that the coefficients of any matrix $\sigma(R^k)$ can be derived from the k-th power of a larger matrix $\mathcal{M}_R$, where the matrix product is defined using min as addition and + as multiplication. More precisely, $\mathcal{M}_R$ is the incidence matrix of the common transition table of the weighted (zigzag) automata $\mathcal{A}_{ij}^t$ for R. The sequence $\{\mathcal{M}_R^k\}_{k=0}^{\infty}$ gives the minimal weights of the paths of length $k = 0, 1, \ldots$ in $\mathcal{A}_{ij}^t$ (Theorem 6). To obtain the simply exponential bounds on the period and prefix of a sequence $\{\sigma(R^k)\}_{k=0}^{\infty}$, we develop this periodicity result further, by studying the structure of the strongly connected components of zigzag automata.

Let $\mathcal{G} = \langle V, E, w\rangle$ be a weighted graph, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, and $w : E \to \mathbb{Z}$ is a weight function. We denote by $\mu(G) = \max(\{\mathrm{abs}(\alpha) \mid u \xrightarrow{\alpha} v\} \cup \{1\})$ the maximum between the absolute values of the weights of G and 1. The *average weight* of a path π is $\overline{w}(\pi) = \frac{w(\pi)}{|\pi|}$. A cycle is said to be *critical* if it has minimal average weight among all cycles of $\mathcal{G}$. The *critical graph* $\mathcal{G}^c$ consists of those vertices and edges of $\mathcal{G}$ that belong to a critical cycle. If C is a strongly connected component (SCC) of $\mathcal{G}^c$, we define its *cyclicity* as the greatest common divisor of the lengths of all its elementary cycles. The cyclicity of $\mathcal{G}^c$ is the least common multiple of the cyclicities of its SCCs, and the cyclicity of $\mathcal{G}$, denoted $c(\mathcal{G})$, is the cyclicity of $\mathcal{G}^c$.

Weighted graphs are intimately related with the powers of their incidence matrices, defined as follows. For two matrices $A, B \in \mathbb{Z}_{\infty}^{n\times n}$, let $(A \boxtimes B)_{ij} = \min_{1\le k\le n}(A_{ik} + B_{kj})$ and $\mathbf{1}_n$ be the matrix $(\mathbf{1}_n)_{ii} = 0$, for all $1 \le i \le n$ and $(\mathbf{1}_n)_{ij} = \infty$, for all $1 \le i, j \le n$, where $i \ne j$. The powers of a matrix M

are defined as $M^0 = \mathbf{1}_n$ and $M^{k+1} = M \boxtimes M^k$, for all $k \geq 0$. If M is the incidence matrix of a weighted graph $\mathcal{G}$, the coefficient $(M^k)_{ij}$ is the weight of a minimal path of length k between the vertices i and j in $\mathcal{G}$. In this case, we write $\mu(M)$ and $c(M)$ for $\mu(\mathcal{G})$ and $c(\mathcal{G})$, respectively. The following theorem provides the tool for proving periodicity of a sequence of matrices:

Theorem 11. *For a matrix $M \in \mathbb{Z}_\infty^{n\times n}$, the sequence $\{M^k\}_{k=0}^\infty$ is periodic and its period divides $c(M)$.*

Proof. See [Sch00, Theorem 3.3]. □

Despite our best efforts, no estimation of the prefix of a power sequence of a matrix could be found in the literature. This gap is filled next:

Theorem 12. *Given a matrix $M \in \mathbb{Z}_\infty^{n\times n}$, the prefix of the periodic sequence $\{M^k\}_{k=0}^\infty$ is at most $4\mu(M)\cdot n^6$.*

We are now ready to prove that the sequence of matrices $\{\sigma(R^n)\}_{n=0}^\infty$ is periodic, where $R \in \mathrm{DB}(\mathbf{x})$ is any difference bounds relation and $\mathbf{x} = \{x_1, \ldots, x_N\}$ is a set of variables. The coefficients of $\sigma(R^n)$ are the weights of the minimal paths between extremal vertices from the set $\mathbf{x}^{(0)} \cup \mathbf{x}^{(n)}$ in the unfolding $\mathcal{G}_R^n$ of the constraint graph $\mathcal{G}_R$ — see the constraints (4.1). By Theorem 6, these weights are given by the functions $\mathrm{minw}_{\mathcal{A}_{ij}^t}(n)$, where $\mathcal{A}_{ij}^t = \langle Q, \omega, I_{ij}^t, F_{ij}^t, \Delta\rangle$ are the zigzag automata for the relation R. Since these functions are periodic, it follows that the sequence $\{\sigma(R^n)\}_{n=0}^\infty$ is periodic. Moreover, the prefix of this sequence is polynomially bounded by $\|Q\|$ and its period divides this cyclicity. Since $\|Q\| = 2^{\mathcal{O}(N)}$ by the construction of zigzag automata, we are left with bounding the cyclicity of zigzag automata.

A path $x_k^{(p)} \xrightarrow{*} x_k^{(q)}$ in the unfolding $\mathcal{G}_R^n$ of the constraint graph $\mathcal{G}_R$ is *essential* if all variables occurring on it are pairwise distinct, except for the labels of its source and destination vertices. Clearly, the length of an essential path is bounded by the number N of variables occurring on this path. An *essential power* is a path ξ^n obtained from the concatenation of an essential path ξ with itself $n > 0$ times. The main idea of the proof is that each critical cycle $\mathbf{q} \xrightarrow{\gamma} \mathbf{q}$ in a zigzag automaton $\mathcal{A}$ is necessarily connected to a critical cycle $\mathbf{q} \xrightarrow{\lambda} \mathbf{q}$, where λ consists of a finite set of essential powers. This allows us to bound the length of λ by a simply exponential value, which divides $\mathrm{lcm}(1, \ldots, N)$. It follows that the common cyclicity of these zigzag automata is a divisor of $\mathrm{lcm}(1, \ldots, N)$. We use the fact that $\mathrm{lcm}(1, \ldots, N) = 2^{\mathcal{O}(N)}$ [Nai82], which occurs as a consequence of the Prime Number Theorem, and bound the cyclicity of zigzag automata by $2^{\mathcal{O}(N)}$.

We have gathered all the elements necessary to prove the second point of Theorem 10, namely that the octagonal relations are periodic, with simply

exponential prefixes and periods. As a consequence, the class of problems ReachFlat(OCT) is Np-complete. The theorem below is a consequence of the relation between the powers of octagonal relations and their encodings using difference bounds constraints (Theorem 8). We infer that the class OCT is periodic, because for two periodic sequences $\{s_k\}_{k=0}^{\infty}$ and $\{t_k\}_{k=0}^{\infty}$, with prefixes $b_s, b_t \geq 0$ and periods $c_s, c_t > 0$, respectively, the following sequences are also periodic:

1. $\{s_k + t_k\}_{k=0}^{\infty}$ with prefix at most $\max(b_s, b_t)$ and period which divides $\mathrm{lcm}(c_s, c_t)$,
2. $\left\{\lfloor \frac{s_k}{2} \rfloor\right\}_{k=0}^{\infty}$ with prefix b_s and period $2c_s$,
3. $\{\min(s_k, t_k)\}_{k=0}^{\infty}$ with prefix at most $\max(b_s, b_t) + \sum_{i=0}^{\mathrm{lcm}(c_s,c_t)}(\mathrm{abs}(s_i) + \mathrm{abs}(t_i))$ and period which divides $\mathrm{lcm}(c_s, c_t)$.

Theorem 13. *There exists a constant $d > 0$ such that, for every relation $R \in$ OCT, the sequence $\{\sigma(R^k)\}_{k=0}^{\infty}$ is periodic, with prefix $b = 2^{\mathcal{O}(|R|^d)}$ and period $c = 2^{\mathcal{O}(|R|^d)}$.*

Proof. See [BIK13, Theorem 8]. □

4.6 The Termination Problem

In this section we address the termination problem for flat counter machines whose cycles are labeled with octagonal relations. More precisely, for a given such cycle, we determine the set of configurations from which the cycle can be iterated ad infinitum. A first observation is that the set of configurations from which an infinite iteration is possible is the greatest fixpoint of the pre-image pre_R of the transition relation[6] R. This set, called the *weakest recurrent set*, and denoted $\mathrm{wrs}(R)$, is the limit of the descending sequence $\mathrm{pre}_R^0(\top), \mathrm{pre}_R^1(\top), \mathrm{pre}_R^2(\top), \ldots$, i.e. $\mathrm{wrs}(R) = \bigcap_{i=1}^{\infty} pre_R^n(\top)$, if either (i) the descending Kleene sequence that over-approximates the greatest fixpoint eventually stabilizes, or (ii) the relation is well founded, i.e. $\mathrm{wrs}(R) = \varnothing$.

If, moreover, the closed form defining the infinite sequence of precondition sets $\{\mathrm{pre}_R^n(\top)\}_{n\geq 1}$ can be defined using a decidable fragment of arithmetic, we obtain decidability proofs for the termination problem. Besides showing decidability, we also establish a polynomial-time upper bound for the case of (non-parametric) octagonal relations $R \in \mathrm{OCT}(\mathbf{x})$.

Let $\mathbf{x} = \{x_1, \ldots, x_N\}$ be a set of variables in the rest of this section. For a relation $R \subseteq \mathbb{Z}^{\mathbf{x}} \times \mathbb{Z}^{\mathbf{x}}$, let $\mathrm{pre}_R : 2^{\mathbb{Z}^{\mathbf{x}}} \to 2^{\mathbb{Z}^{\mathbf{x}}}$ be the *pre-image* function

[6] This definition is the dual of the *reachability set*, needed for checking safety properties: the reachability set is the least fixpoint of the post-image of the transition relation.

defined as $\mathrm{pre}_R(S) = \{\nu \mid \exists \nu' \in S \ . \ (\nu, \nu') \in R\}$, for any set $S \subseteq \mathbb{Z}^{\mathbf{x}}$ of configurations. It is not difficult to prove that pre_R is a monotonic function, thus the sequence $\{\mathrm{pre}_R^n(\mathbb{Z}^{\mathbf{x}})\}_{n=0}^{\infty}$ is descending.

A relation $R \subseteq \mathbb{Z}^{\mathbf{x}} \times \mathbb{Z}^{\mathbf{x}}$ is said to be *well-founded* if and only if there is no infinite sequence of valuations $\{\nu_i\}_{i=0}^{\infty}$ such that $(\nu_i, \nu_{i+1}) \in R$, for all $i \geq 0$. We recall that a relation R is said to be $*$*-consistent* if and only if $R^i \neq \varnothing$, for all $i \geq 0$. Observer that if a relation is not $*$-consistent, then it is also well-founded. However the dual is not true. For instance, the relation $R = \{(n, n-1) \mid n \in \mathbb{N}_+\}$ is both $*$-consistent and well-founded.

Definition 7. *A set $S \subseteq \mathbb{Z}^{\mathbf{x}}$ is said to be a* non-termination precondition *for a relation $R \subseteq \mathbb{Z}^{\mathbf{x}} \times \mathbb{Z}^{\mathbf{x}}$ if and only if for each $\nu \in S$ there exists an infinite sequence of valuations $\{\nu_i\}_{i=0}^{\infty}$ such that $\nu = \nu_0$ and $\nu_i \in \mathbb{Z}^{\mathbf{x}}$, $(\nu_i, \nu_{i+1}) \in R$, for all $i \geq 0$.*

If $S_0, S_1, \ldots$ are all non-termination preconditions for R, then the (possibly infinite) union $\bigcup_{i=0,1,\ldots} S_i$ is a non-termination precondition for R as well. The set $\mathrm{wnt}(R) = \bigcup\{S \in \mathbb{Z}^{\mathbf{x}} \mid S \text{ is a non-termination precondition for } R\}$ is called the *weakest non-termination precondition* for R. A relation R is well founded if and only if $\mathrm{wnt}(R) = \varnothing$. A set S such that $S \cap \mathrm{wnt}(R) = \varnothing$ is called a *termination precondition*.

Definition 8. *A set $S \subseteq \mathbb{Z}^{\mathbf{x}}$ is said to be* recurrent *for a relation $R \subseteq \mathbb{Z}^{\mathbf{x}} \times \mathbb{Z}^{\mathbf{x}}$ if and only if $S \subseteq \mathrm{pre}_R(S)$.*

Notice that if S is a recurrent set for a relation R, then for each $\nu \in S$ there exists $\nu' \in S$ such that $(\nu, \nu') \in R$, thus one can build an infinite sequence of configurations, corresponding to an infinite iteration of R. It is easy to check that the union of several (possibly infinitely many) recurrent sets is also a recurrent set.

The set $\mathrm{wrs}(R) = \bigcup\{S \in \mathbb{Z}^{\mathbf{x}} \mid S \text{ is a recurrent set for } R\}$ is called the *weakest recurrent set* for R. Thus $\mathrm{wrs}(R)$ is recurrent for R. The following lemma shows that in fact, $\mathrm{wrs}(R)$ is exactly the set of valuations from which an infinite iteration of R is possible and, equivalently, the greatest fixpoint of the transition relation's pre-image.

Lemma 2. *For every relation $R \subseteq \mathbb{Z}^{\mathbf{x}} \times \mathbb{Z}^{\mathbf{x}}$, $\mathrm{wrs}(R) = \mathrm{wnt}(R) = \mathrm{gfp}(\mathrm{pre}_R)$.*

Proof. See [BIK14a, Lemma 3.6]. □

The following lemma gives sufficient conditions under which $\mathrm{wrs}(R)$ can be computed as the limit $\bigcap_{n \geq 0} \mathrm{pre}_R^n(\mathbb{Z}^{\mathbf{x}})$ of the infinite descending Kleene sequence $\mathbb{Z}^{\mathbf{x}} \supseteq \mathrm{pre}_R(\mathbb{Z}^{\mathbf{x}}) \supseteq \mathrm{pre}_R^2(\mathbb{Z}^{\mathbf{x}}) \supseteq \mathrm{pre}_R^3(\mathbb{Z}^{\mathbf{x}}) \supseteq \ \ldots$

Lemma 3. *Let $R \subseteq \mathbb{Z}^{\mathbf{x}} \times \mathbb{Z}^{\mathbf{x}}$ be a relation such that at least one of the following holds:*

1. $\bigcap_{n\geq 1} \mathrm{pre}_R^n(\mathbb{Z}^{\mathbf{x}}) = \emptyset$, *or*
2. $\mathrm{pre}_R^{n_2}(\mathbb{Z}^{\mathbf{x}}) = \mathrm{pre}_R^{n_1}(\mathbb{Z}^{\mathbf{x}})$ *for some* $n_2 > n_1 \geq 1$.

Then, we have $\mathrm{wrs}(R) = \bigcap_{n\geq 1} \mathrm{pre}_R^n(\mathbb{Z}^{\mathbf{x}})$. *Moreover,* $\mathrm{wrs}(R) = \emptyset$ *if (1) holds and* $\mathrm{wrs}(R) = \mathrm{pre}_R^{n_1}(\mathbb{Z}^{\mathbf{x}})$ *if (2) holds.*

In the rest of this section we show that octagonal relations satisfy at least one of the two conditions above. Let us denote by $\widehat{\mathrm{pre}_R}(k, \mathbf{x})$ the *closed form* of the sequence $\{\mathrm{pre}_R^n(\mathbb{Z}^{\mathbf{x}})\}_{n=0}^{\infty}$:

$$\forall \nu \in \mathbb{Z}^{\mathbf{x}} \ \forall n \in \mathbb{N} : \ \nu \in \mathrm{pre}_R^n(\mathbb{Z}^{\mathbf{x}}) \Leftrightarrow \nu \vDash \widehat{\mathrm{pre}_R}[n/k] \ .$$

As shown in Section 4.2, the closed form of the power sequence $\{R^n\}_{n\geq 0}$ is definable by a Presburger formula $\widehat{R}(k, \mathbf{x}, \mathbf{x}')$. Since, moreover, $\mathrm{pre}_R^n = \mathrm{pre}_{R^n}$, for all $n \in \mathbb{N}$, we have $\widehat{\mathrm{pre}_R}(k, \mathbf{x}) \equiv \exists \mathbf{x}' \ . \ \widehat{R}(k, \mathbf{x}, \mathbf{x}')$. The above lemma shows that $\mathrm{wnt}(R)$ is defined by the Presburger formula:

$$\forall k \geq 0 \ . \ \widehat{\mathrm{pre}_R}(k, \mathbf{x}) \Leftrightarrow \forall k \geq 0 \exists \mathbf{x}' \ . \ \widehat{R}(k, \mathbf{x}, \mathbf{x}') \ .$$

Since satisfiability is decidable for Presburger arithmetic [Pre29], the (universal) termination problem for octagonal relations is decidable as well.

Example 5. *Consider the relation* $R(x, x') \Leftrightarrow x \geq 0 \wedge x' = x - 1$. *The closed form of the sequence* $\{\mathrm{pre}_R^n(\mathbb{Z}^{\mathbf{x}})\}_{n\geq 1}$ *is* $\widehat{\mathrm{pre}_R}(k, x) \Leftrightarrow k \geq 1 \wedge x \geq k - 1$. *Then we have:*

$$(\mathrm{wnt}(R))(\mathbf{x}) \Leftrightarrow \forall k \geq 1 \ . \ \widehat{\mathrm{pre}_R}(k, x) \Leftrightarrow \forall k \geq 1 \ . \ k \geq 1 \wedge x \geq k - 1 \Leftrightarrow \bot$$

Hence the relation R is well founded. ∎

4.6.1 Computing Weakest Recurrent Sets in Polynomial Time

If $R \in \mathrm{OCT}(\mathbf{x})$ is an octagonal relation, let $\overline{R} \in \mathrm{DB}(\mathbf{y})$, for $\mathbf{y} = \{y_1, \ldots, y_{2N}\}$, be its difference bounds representation. The main result of this section is a (deterministic) algorithm (Algorithm 2) that computes the weakest recurrent set of an octagonal relation R in $\textsc{Ptime}(|R|)$. The main insight of the algorithm is that the Kleene sequence $\{\mathrm{pre}_R^n(\mathbb{Z}^{\mathbf{x}})\}_{n=0}^{\infty}$ either (1) never stabilizes, in which case

$$\mathrm{pre}_R^1(\mathbb{Z}^{\mathbf{x}}) \supsetneq \mathrm{pre}_R^2(\mathbb{Z}^{\mathbf{x}}) \supsetneq \mathrm{pre}_R^3(\mathbb{Z}^{\mathbf{x}}) \supsetneq \ \ldots$$

and $\mathrm{wrs}(R) = \varnothing$, or (2) stabilizes after at most 5^{2N} steps, in which case

$$\mathrm{wrs}(R) = \mathrm{pre}_R^{5^{2N}}(\mathbb{Z}^{\mathbf{x}}) = \mathrm{pre}_R^{5^{2N}+1}(\mathbb{Z}^{\mathbf{x}}) = \mathrm{pre}_R^{5^{2N}+2}(\mathbb{Z}^{\mathbf{x}}) = \ \ldots$$

Then, the stability of the sequence can be checked by checking equality between the 5^{2N}-th and $(5^{2N}+1)$-th iterate. Since the size of the binary representation of the n-th power of an octagonal relation increases by a polynomial of $\log n$ (see the first condition of Theorem 10 for a formal statement) the formulae defining these sets are computable in $\textsc{Ptime}(|R|)$, using exponentiation by squaring (the $\textsc{Square}(R,n)$ function on lines 1 and 2 of the algorithm). As a direct consequence, we obtain a decision procedure for the termination problem with the same worst-case complexity, simply by testing the consistency of the computed $\mathrm{wrs}(R)$, which is an octagonal constraint as well.

For a $4N \times 4N$ DBM M, we denote by ${}^{\blacksquare}M$ its top left corner, which is a $2N \times 2N$ DBM. The image mapping $\varrho({}^{\blacksquare}M)$ returns the octagonal constraint involving only atomic propositions of the for $\pm x_i \pm x_j \leq c$, for $1 \leq i, j \leq 2N$.

Algorithm 2 Weakest Recurrent Sets for Octagonal Relations

input An octagonal constraint $R(\mathbf{x}, \mathbf{x}')$ where $\mathbf{x} = \{x_1, \ldots, x_N\}$
output An octagonal constraint representing $\mathrm{wrs}(R)$

1: $V(\mathbf{x}, \mathbf{x}') \leftarrow \textsc{Square}(R, 5^{2N})$
2: $W(\mathbf{x}, \mathbf{x}') \leftarrow \textsc{Square}(R, 5^{2N}+1)$
3: **if** $W \Leftrightarrow$ false **or** ${}^{\blacksquare}M_V^t > {}^{\blacksquare}M_W^t$ **then**
4: **return** false
5: **else**
6: **return** $\varrho({}^{\blacksquare}M_V^t)$

The following theorem proves that Algorithm 2 is correct.

Theorem 14. *Let $R \in \mathrm{OCT}(\mathbf{x})$ be an octagonal relation, where $\mathbf{x} = \{x_1, \ldots, x_N\}$. Then, Algorithm 2, with input a formula that defines R, returns an octagonal constraint that defines $\mathrm{wrs}(R)$. Moreover, the well-foundedness problem $\mathrm{wrs}(R) = \varnothing$ can be decided in $\textsc{Ptime}(|R|)$.*

Proof. See [BIK14a, Theorems 4.38 and 4.39]. □

For parametric octagonal relations, the decidability status and complexity of the termination problem remains open (Section 4.7).

4.7 Discussion and Open Problems

The first proof of Presburger-definability of closed forms for difference bounds relations has been given by Comon and Jurski [CJ98], using an overapproximation, called *folded graph*, of the set of paths in the unfolding of a constraint graph. Their proof is based on the fact that only certain paths in this graph are relevant for the definition of the closed form, namely those paths that do not change direction while traversing vertices from the same SCC of the folded graph.

Our first proof is based on the fact that these paths can be recognized by a finite weighted automaton [BIL09]. This proves the decidability of the reachability problem in a more general context, when the bounds are given by parametric linear terms. Moreover, using results from the theory of weighted automata and tropical algebra, we prove that the sequence of DBMs encoding the powers of a relation $\{R^n\}_{n=0}^{\infty}$ is periodic, giving simply-exponential bounds on the prefix and period of this sequence. In particular, the simple exponential upper bound on the period of such sequences requires an insight on the particular structure of cycles in the zigzag automaton [BIL09], and relies on the idea of restricting to a set of simple paths with a bounded number of direction changes, whose weights subsume the set of weights of the minimal paths in the unfolding graph [CJ98].

Developing further the idea of folded graphs [CJ98], Konečný showed that the closed form of the power sequence of a difference bounds (respectively, octagonal) relation can be defined by a quantifier-free Presburger formula which, moreover, can be built in polynomial time by a deterministic algorithm [Kon14]. This gives an alternative proof of the fact that the reachability problem for flat counter machines is in NPTIME, using a polynomial reduction to the satisfiability of quantifier-free Presburger arithmetic.

Open Problems

The following decidable problems currently require tight complexity bounds:

1. The reachability and termination problems for flat counter machines with parametric octagonal cycles.
2. The reachability and termination problems for flat counter machines with cycles labeled by affine relations of the form $\varphi(\mathbf{x}) \wedge \mathbf{x}' = A\mathbf{x} + \mathbf{b}$, where φ is a quantifier-free Presburger formula and the set of matrix powers $A, A^2, \ldots$ is finite (finite monoid [Boi99, FL02b]). As a remark, the model checking problem for Linear Temporal Logic, in the case where A is the identity matrix is coined to NP-complete [DDS12].

Chapter 5

Recursive Counter Machines

Counter machines (Chapter 4) are suitable for modeling intra-procedural executions of programs with integer variables, and can, incidentally, be used as a tool for reasoning about array logics and programs with dynamically allocated recursive data structures (Chapter 6). In this chapter we describe an extension of this model, by allowing counter machines to make recursive calls to other counter machines.

From an operational point of view, this is equivalent to having a stack on which the values of the local variables are pushed prior to a call and retrieved upon return from a call. In other words, a recursive counter machine may be viewed as a pushdown automaton equipped with an infinite stack alphabet. For technical convenience, we work with visibly pushdown grammars instead of pushdown automata, and to represent the execution of a recursive counter machine as a derivation of a grammar, directly inferred from the description of the machine.

Procedure summaries are relations between the input and return values of a procedure, resulting from its terminating executions. Computing summaries is important, as they are a key enabler for the development of modular verification techniques for inter-procedural programs, such as checking safety, termination or equivalence properties. Given a recursive counter machine, represented as a visibly pushdown grammar, we give an effective method to compute increasingly larger under-approximations of its summaries, by considering only derivations that do not exceed a given budget, called *index*, on the number of occurrences of nonterminals occurring at each derivation step.

The under-approximation method converges yielding the precise program summaries, provided that the language of actions generated by the

grammar is contained in the language of a regular expression $w_1^* \dots w_d^*$ where each w_i is a finite sequence of program statements. Finally, we show that the reachability problem for recursive counter machines, whose statements are described by octagonal relations, is in NEXPTIME, with an NP-hard lower bound [GI15a]. Despite our best efforts, we did not close this complexity gap yet. However, when setting the derivation index to a fixed constant, the complexity of the resulting reachability problem for programs with arbitrary call graphs becomes NP-complete. This is joint work with Pierre Ganty (IMDEA, Madrid) [GIK13, GIK12, GI15a].

5.1 Programs as Visibly Pushdown Grammars

In the following, we use the term *program* to denote a recursive counter machine, and *procedure* to denote one of its components. In other words, a program is a collection of procedures calling each other, possibly according to recursive schemes. Formally, a program is a tuple $\mathcal{P} = \langle P_1, \dots, P_n \rangle$ and each procedure is given by a *control flow graph* $P_i = \langle \mathbf{x}_i, \mathbf{x}_i^{in}, \mathbf{x}_i^{out}, S_i, q_i^{init}, F_i, \Delta_i \rangle$, where $\mathbf{x}_i$ are the *local* variables[1] of P_i ($\mathbf{x}_i \cap \mathbf{x}_j = \varnothing$ for all $i \neq j$), $\mathbf{x}_i^{in}, \mathbf{x}_i^{out} \subseteq \mathbf{x}_i$ are ordered sequences of input and output variables, S_i are the *states* of P_i ($S_i \cap S_j = \varnothing$, for all $i \neq j$), $q_i^{init} \in S_i \setminus F_i$ is the *initial*, and $F_i \subseteq S_i$ ($F_i \neq \varnothing$) are the *final* states, and Δ_i is a set of *transitions*:

- $q \xrightarrow{\varphi(\mathbf{x}_i, \mathbf{x}_i')} q'$ is an *internal transition*, where $q, q' \in S_i$, and $\varphi(\mathbf{x}_i, \mathbf{x}_i')$ is an arithmetic formula involving only the local variables of P_i,
- $q \xrightarrow{\mathbf{z}' = P_j(\mathbf{u})} q'$ is a *call*, where $q, q' \in S_i$, P_j is the callee, $\mathbf{u}$ is an ordered sequence of linear terms over $\mathbf{x}_i$, $\mathbf{z} \subseteq \mathbf{x}_i$ is an ordered sequence of variables, such that $|\mathbf{u}| = |\mathbf{x}_j^{in}|$ and $|\mathbf{z}| = |\mathbf{x}_j^{out}|$.

The *call graph* of a program $\mathcal{P} = \langle P_1, \dots, P_n \rangle$ is a directed graph with vertices $P_1, \dots, P_n$ and an edge (P_i, P_j), for each P_i and P_j, such that P_i has a call to P_j. A program is *recursive* if its call graph has at least one cycle, and *non-recursive* if its call graph is a dag. We denote by $\mathcal{F}(\mathcal{P}) = \bigcup_{i=1}^n F_i$ the set of final states of the program $\mathcal{P}$, by $n\mathcal{F}(P_i)$ the set $S_i \setminus F_i$ of non-final states of P_i, and by $n\mathcal{F}(\mathcal{P}) = \bigcup_{i=1}^n n\mathcal{F}(\mathcal{P})$ be the set of non-final states of $\mathcal{P}$.

Example 6. *Figure 5.1 shows a program $\mathcal{P} = \langle P \rangle$, where*

$$P = \langle \{x, z\}, \{x\}, \{z\}, \{q_1^{init}, q_2, q_3, \varepsilon\}, q_1^{init}, \{\varepsilon\}, \{t_1, t_2, t_3, t_4\} \rangle$$

[1] Observe that there are no global variables in the definition of integer program. Those can be encoded as input and output variables to each procedure.

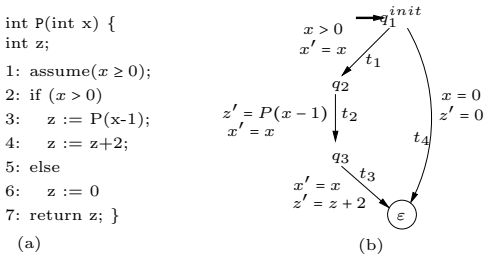


Figure 5.1: A recursive program that returns the input value multiplied by two (a) and its control flow graph (b)

is the only procedure. Since P calls itself once (by the call t_2), this program is recursive. ■

To model the control flow of procedural programs, we use languages generated by visibly pushdown grammars [AM09], a subset of context-free grammars. In this setting, words are defined over a *tagged alphabet* $\widehat{\Sigma} = \Sigma \cup \langle\!\langle \Sigma \cup \Sigma \rangle\!\rangle$, where $\langle\!\langle \Sigma = \{\langle\!\langle a \mid a \in \Sigma\}$ represents procedure *calls* and $\Sigma\rangle\!\rangle = \{a\rangle\!\rangle \mid a \in \Sigma\}$ represents procedure *returns*. Formally, a *visibly pushdown grammar* $G = \langle \Xi, \widehat{\Sigma}, \Delta \rangle$ consists of a set of nonterminals Ξ, a tagged alphabet $\widehat{\Sigma}$ and a set of productions Δ, of the following forms, for some $a, b \in \Sigma$:

$$\text{(a) } X \to a \qquad \text{(b) } X \to a\,Y \qquad \text{(c) } X \to \langle\!\langle a\,Y\,b \rangle\!\rangle\,Z$$

We are interested in computing the *summary relation* between the values of the input and output variables of a procedure. To this end, we give the semantics of a program $\mathcal{P} = \langle P_1, \ldots, P_n \rangle$ as a tuple of relations, denoted $[\![q]\!]$ in the following, describing, for each non-final state $q \in n\mathcal{F}(P_i)$ of a procedure P_i, the effect of the program when started in q upon reaching a state in F_i. The summary of a procedure $[\![P_i]\!] = [\![q_i^{init}]\!]$ is the relation corresponding to its unique initial state.

An *interprocedurally valid path* is represented by a tagged word over an alphabet $\widehat{\Theta}$, which maps each internal transition t to a symbol τ, and each call transition t to a pair of symbols $\langle\!\langle \tau, \tau \rangle\!\rangle \in \widehat{\Theta}$. In the sequel, we denote by Q the nonterminal corresponding to the state q, and by $\tau \in \Theta$ the alphabet symbol corresponding to the transition t of $\mathcal{P}$. Formally, we associate each program $\mathcal{P}$ a visibly pushdown grammar, denoted in the following by $G_{\mathcal{P}} = \langle \Xi, \widehat{\Theta}, \Delta \rangle$, such that $Q \in \Xi$ if and only if $q \in n\mathcal{F}(\mathcal{P})$ and:
(a) $Q \to \tau \in \Delta$ if and only if t is $q \xrightarrow{\varphi} q'$ and $q' \in \mathcal{F}(\mathcal{P})$,
(b) $Q \to \tau\ Q' \in \Delta$ if and only if t is $q \xrightarrow{\varphi} q'$ and $q' \in n\mathcal{F}(\mathcal{P})$,

(c) $Q \to \langle\!\langle \tau\ Q_j^{init}\ \tau \rangle\!\rangle\ Q' \in \Delta$ if and only if t is $q \xrightarrow{\mathbf{z}'=P_j(\mathbf{u})} q'$.

Example 7. *The visibly pushdown grammar for the program in Figure 5.1 is given below:*

$$\mathbf{p}_1^b : Q_1^{init} \to \tau_1 Q_2 \qquad \mathbf{p}_3^a : Q_3 \to \tau_3$$
$$\mathbf{p}_2^c : Q_2 \to \langle\!\langle \tau_2\ Q_1^{init}\ \tau_2 \rangle\!\rangle\ Q_3 \qquad \mathbf{p}_4^a : Q_1^{init} \to \tau_4$$

■

Each tagged word generated by visibly pushdown grammars is associated a *nested word*, i.e. a pair $(w, \rightsquigarrow)$, where $\rightsquigarrow \subseteq \{1, \ldots, |w|\} \times \{1, \ldots, |w|\}$ is a set of *nesting edges*, where:

1. $i \rightsquigarrow j$ only if $i < j$; edges only go forward;
2. $\|\{j \mid i \rightsquigarrow j\}\| \leq 1$ and $\|\{i \mid i \rightsquigarrow j\}\| \leq 1$; no two edges share a call/return position;
3. if $i \rightsquigarrow j$ and $k \rightsquigarrow \ell$ then it is not the case that $i < k \leq j < \ell$; edges do not cross.

Dually, we associate a nested word to a tagged word as follows: there is an edge between tagged symbols $\langle\!\langle a$ and $a \rangle\!\rangle$ if and only if both symbols are produced by the same derivation step. Finally, let w_nw denote the mapping which given a tagged word in the language of a visibly pushdown grammar returns the nested word thereof.

Example 8. *For the tagged word* $w = \tau_1\langle\!\langle\tau_2\tau_1\langle\!\langle\tau_2\tau_4\tau_2\rangle\!\rangle\tau_3\tau_2\rangle\!\rangle\tau_3$, $w\_nw(w) = (\tau_1\tau_2\tau_1\tau_2\tau_4\tau_2\tau_3\tau_2\tau_3, \{2 \rightsquigarrow 8, 4 \rightsquigarrow 6\})$ *is the associated nested word.* ■

The semantics of a program is the union of the relations (between initial and final valuations of the variables) induced by the nested words corresponding to its executions. To define the relational semantics of a nested word, we first associate to each $\tau \in \widehat{\Theta}$ an integer relation ρ_τ, defined as:

- for an internal transition $t{:}\, q \xrightarrow{\varphi} q' \in \Delta_i$, we define $\rho_\tau \equiv \varphi(\mathbf{x}_i, \mathbf{x}_i')$;
- for a call transition $t{:}\, q \xrightarrow{\mathbf{z}'=P_j(\mathbf{u})} q' \in \Delta_i$, we define a *call relation* $\rho_{\langle\!\langle\tau} \equiv (\mathbf{x}_j^{in'} = \mathbf{u}) \subseteq \mathbb{Z}^{\mathbf{x}_i} \times \mathbb{Z}^{\mathbf{x}_j}$, a *return relation* $\rho_{\tau\rangle\!\rangle} \equiv (\mathbf{z}' = \mathbf{x}_j^{out}) \subseteq \mathbb{Z}^{\mathbf{x}_j} \times \mathbb{Z}^{\mathbf{x}_i}$ and a *frame relation* $\phi_\tau \equiv \bigwedge_{x \in \mathbf{x}_i \setminus \mathbf{z}} x' = x$. Intuitively, the frame relation copies the values of all local variables, that are not involved in the call as return value receivers ($\mathbf{z}$), across the call.

We define the semantics of the program $\mathcal{P} = \langle P_1, \ldots, P_n \rangle$ in a top-down manner. Assuming a fixed ordering of the non-final states in the program, i.e. $n\mathcal{F}(\mathcal{P}) = \langle q_1, \ldots, q_m \rangle$, the semantics of the program $\mathcal{P}$ is the tuple of relations $[\![\mathcal{P}]\!] = \langle [\![q_1]\!], \ldots, [\![q_m]\!] \rangle$. For each non-final state $q \in n\mathcal{F}(P_i)$ where $1 \leq i \leq n$, we denote by $[\![q]\!] \subseteq \mathbb{Z}^{\mathbf{x}_i} \times \mathbb{Z}^{\mathbf{x}_i}$ the relation (over the local variables

of procedure P_i) defined as $[\![q]\!] = \bigcup_{\alpha \in L_Q(G_\mathcal{P})} [\![\alpha]\!]$, where $L_Q(G_\mathcal{P})$ is the language generated by the grammar $G_\mathcal{P}$ with axiom Q. Then, for each procedure P_i in the program, we have $[\![P_i]\!] = \bigcup_{\alpha \in L_{Q_i^{init}}(G_\mathcal{P})} [\![\alpha]\!]$.

It remains to define $[\![\alpha]\!]$, the semantics of the tagged word (or equivalently interprocedural valid path) α. Out of convenience, we define the semantics of its corresponding nested word $w\_nw(\alpha) = (\theta, \rightsquigarrow)$ over the alphabet Θ, and define $[\![\alpha]\!] = [\![w\_nw(\alpha)]\!]$. For a nesting relation $\rightsquigarrow \subseteq \{1, \ldots, |\theta|\} \times \{1, \ldots, |\theta|\}$, we define $\rightsquigarrow_{i,j} = \{(s-(i-1), t-(i-1)) \mid (s,t) \in \rightsquigarrow \cap \{i, \ldots, j\} \times \{i, \ldots, j\}\}$, for some $i, j \in \{1, \ldots, \ell\}$, $i < j$. Finally, we define:

$$[\![(\theta, \rightsquigarrow)]\!] = \begin{cases} \rho_{(\theta)_1} & \text{if } |\theta| = 1 \\ \rho_{(\theta)_1} \circ [\![((\theta)_{2\ldots|\theta|}, \rightsquigarrow_{2,|\theta|})]\!] & \text{if } |\theta| > 1,\ 1 \not\rightsquigarrow j \text{ for no } j \\ CaRet^j_\theta \circ [\![((\theta)_{j+1\ldots|\theta|}, \rightsquigarrow_{j+1,|\theta|})]\!] & \text{if } |\theta| > 1,\ 1 \rightsquigarrow j \text{ for some } j \end{cases}$$

where, in the last case, which corresponds to call transition $t \in \Delta_i$, we have $(\theta)_1 = (\theta)_j = \tau$ and define $CaRet^j_\theta = \left(\rho_{\langle\tau} \circ [\![(\theta)_{2\ldots j-1}, \rightsquigarrow_{2,j-1})]\!] \circ \rho_{\tau\rangle}\right) \cap \phi_\tau$.

Example 9. *The semantics of the nested word* $\theta = (\tau_1\tau_2\tau_1\tau_2\tau_4\tau_2\tau_3\tau_2\tau_3, \{2 \rightsquigarrow 8, 4 \rightsquigarrow 6\})$ *is the relation defined by:*

$$[\![\theta]\!] \;=\; \rho_{\tau_1} \circ \left((\rho_{\langle\tau_2} \circ \rho_{\tau_1} \circ \left((\rho_{\langle\tau_2} \circ \rho_{\tau_4} \circ \rho_{\tau_2\rangle}) \cap \phi_{\tau_2}\right) \circ \rho_{\tau_3} \circ \rho_{\tau_2\rangle}) \cap \phi_{\tau_2}\right) \circ \rho_{\tau_3}$$

One can verify that $[\![\theta]\!] \equiv x = 2 \wedge z' = 4$, *i.e. the result of calling* P *with input valuation* $x = 2$ *is an output valuation* $z = 4$. ■

5.2 Underapproximating Summaries

In what follows we define under-approximations of context-free languages, by filtering out derivations. The outcome is the definition of an under-approximation sequence of the semantics $[\![\mathcal{P}]\!]$ of a program $\mathcal{P}$, called *k-index under-approximations*, where $k > 0$ is an integer constant. Intuitively, each iterate of the under-approximation sequence corresponds to the execution of a recursive program, with unbounded stack usage, that proceeds along those traces produced by the visibly pushdown grammar $G_\mathcal{P}$ using at most k nonterminals at each derivation step. In particular, we show that the summary semantics of such a program can be inferred from the semantics of a non-recursive program, obtained directly from the program $\mathcal{P}$ and the constant $k > 0$ by a linear-time source-to-source program transformation.

First, we require several technical definitions. Given a grammar $G = \langle \Xi, \Sigma, \Delta \rangle$, two strings $u, v \in (\Sigma \cup \Xi)^*$, a production $(X, w) \in \Delta$ and $1 \leq$

$j \leq |u|$, we define a *step* $u \xRightarrow{(X,w)/j}_G v$ if, and only if, $(u)_j = X$ and $v = (u)_1 \cdots (u)_{j-1} \cdot w \cdot (u)_{j+1} \cdots (u)_{|u|}$. We omit (X, w) or j above the arrow and the subscript G when they are not important. *Step sequences* are defined using the reflexive and transitive closure of the step relation $\Rightarrow_G$, denoted $\Rightarrow_G^*$. For instance, $X \Rightarrow_G^* w$ means there exists a sequence of steps that produces the word $w \in (\Sigma \cup \Xi)^*$, starting from X. We call any *step sequence* $v \Rightarrow_G^* w$ *a derivation* whenever $v \in \Xi$ and $w \in \Sigma^*$. The language produced by G, starting with a nonterminal X is the set $L_X(G) = \{w \in \Sigma^* \mid X \Rightarrow_G^* w\}$.

A *control word* is a sequence of productions $\gamma_1 \ldots \gamma_n \in \Delta^*$. Given a control word γ of length n we write $u \xRightarrow{\gamma}_G v$ whenever there exist words $w_0, \ldots, w_n \in (\Xi \cup \Sigma)^*$ such that $u = w_0 \xRightarrow{\gamma_1}_G w_1 \ldots w_{n-1} \xRightarrow{\gamma_n}_G w_n = v$. For a nonterminal $X \in \Xi$ and a set $\Gamma \subseteq \Delta^*$ of control words, also referred to as a *control set*, we denote by $\hat{L}_X(\Gamma, G) = \{w \in \Sigma^* \mid \exists \gamma \in \Gamma : X \xRightarrow{\gamma} w\}$ the language generated by G using only control words in Γ.

A step sequence (derivation) is said to be *depth-first* if it has the following informal property: if X and Y are two nonterminals produced by the application of one rule, then the steps corresponding to a full derivation of the form $X \Rightarrow^* u$ will be applied *without interleaving* with the steps corresponding to a derivation of the form $Y \Rightarrow^* v$. In other words, once the derivation of X has started, it will be finished before the derivation of Y begins. It is well-known that restricting the set of derivations of a grammar to depth-first derivations only, does not change the produced language.

Example 10. *Consider the grammar* $G = \langle \{X, Y, Z\}, \{a, b\}, \Delta \rangle$ *where* $\Delta = \{X \to YZ,\ Y \to aY \mid \varepsilon,\ Z \to Zb \mid \varepsilon\}$. *Then* $X \xRightarrow{(X,YZ)} YZ \xRightarrow{(Y,aY)} aYZ \xRightarrow{(Z,Zb)} aYZb \xRightarrow{(Y,\epsilon)} aZb \xRightarrow{(Z,\epsilon)} ab$ *is not a depth-first derivation, whereas* $X \xRightarrow{(X,YZ)} YZ \xRightarrow{(Y,aY)} aYZ \xRightarrow{(Y,\epsilon)} aZ \xRightarrow{(Z,Zb)} aZb \xRightarrow{(Z,\epsilon)} ab$ *is a depth-first derivation.* ■

The central notion of this section are *index-bounded derivations*, i.e. derivations in which each step has a *limited budget* of nonterminals. This notion is the key to our under-approximation method. For a given integer constant $k > 0$, a word $u \in (\Sigma \cup \Xi)^*$ is said to be of index k, if u contains at most k occurrences of nonterminals. A step $u \Rightarrow v$ is said to be k-indexed, denoted $u \underset{(k)}{\Rightarrow} v$, if and only if both u and v are of index k. As expected, a depth-first step sequence is k-indexed if all its steps are k-indexed, denoted $u \xRightarrow[\mathbf{df}(k)]{}^* v$. For any nonterminal(s) $X \in \Xi$, $Y \in \Xi \cup \{\varepsilon\}$, and $k > 0$, we define:

$$L_X^{(k)}(G) = \left\{ w \in \Sigma^* \mid X \xRightarrow[\mathbf{df}(k)]{}^* w \right\} \quad \Gamma_{X,Y}^{\mathbf{df}(k)}(G) = \left\{ \gamma \in \Delta^* \mid \exists u, v \in \Sigma^* : X \xRightarrow[\mathbf{df}(k)]{\gamma} uYv \right\}$$

We write $\Gamma_X^{\mathbf{df}(k)}(G)$ for $\Gamma_{X,\varepsilon}^{\mathbf{df}(k)}(G)$ in the following.

Example 11. *Inspecting the grammar $G_{\mathcal{P}}$ for the program $\mathcal{P}$ in Figure 5.1 reveals that $L_{Q_1^{init}}(G_{\mathcal{P}}) = \{(\tau_1 \langle\!\langle \tau_2)^n \tau_4 (\tau_2 \rangle\!\rangle \tau_3)^n \mid n \in \mathbb{N}\}$. For each value of n we give a 2-index derivation capturing the word $(\tau_1 \langle\!\langle \tau_2)^n \tau_4 (\tau_2 \rangle\!\rangle \tau_3)^n$: repeat n times the steps $Q_1^{init} \overset{p_1^b p_2^c}{\Longrightarrow} \tau_1 \langle\!\langle \tau_2 Q_1^{init} \tau_2 \rangle\!\rangle Q_3 \overset{p_3^a}{\Longrightarrow} \tau_1 \langle\!\langle \tau_2 Q_1^{init} \tau_2 \rangle\!\rangle \tau_3$ followed by $Q_1^{init} \overset{p_4^a}{\Longrightarrow} \tau_4$. Therefore the 2-index under-approximation of $G_{\mathcal{P}}$ shows that $L_{Q_1^{init}}(G_{\mathcal{P}}) = L^{(2)}_{Q_1^{init}}(G_{\mathcal{P}})$.* ■

Given a grammar G and a nonterminal X, for any $k > 0$ we have $L_X^{(k)}(G) \subseteq L_X(G)$. The k-index under-approximation of the semantics of a program $\mathcal{P} = \langle P_1, \ldots, P_n \rangle$ is $[\![P]\!]^{(k)} = \langle [\![P_1]\!]^{(k)}, \ldots, [\![P_n]\!]^{(k)} \rangle$, where:

$$[\![P_i]\!]^{(k)} = \bigcup_{\alpha \in L^{(k)}_{Q_i^{init}}(G_{\mathcal{P}})} [\![\alpha]\!], \text{ for each } i = 1, \ldots, n \ .$$

It is not hard to prove that the summary $[\![\mathcal{P}]\!]$ is the limit of the increasing sequence of under-approximations $[\![\mathcal{P}]\!]^{(1)} \subseteq [\![\mathcal{P}]\!]^{(2)} \subseteq \ldots$ This sequence can be seen as the Newton fixpoint iteration, defined by Esparza, Kieffer and Luttenberger [EKL10], applied to programs with integer parameters, return values, and local variables.

The under-approximation sequence $\left\{[\![\mathcal{P}]\!]^{(k)}\right\}_{k=1}^{\infty}$ is thus defined by filtering out derivations of $G_{\mathcal{P}}$ of index more than k. The remaining question is how to compute the iterates of this sequence, given the grammar $G_{\mathcal{P}}$ and the constant $k > 0$. A first observation is that the relation $[\![\alpha]\!]$ induced by a tagged word α can be equivalently computed by looking at the control word γ such that $Q_i^{init} \overset{\gamma}{\Longrightarrow} \alpha$, for some procedure index $i = 1, \ldots, n$. To avoid cluttering the presentation, we do not give the details of the definition of control word semantics here, and refer the reader to [GIK12, Section 3.3].

Second, the control set $\Gamma^{\mathbf{df}(k)}_{Q_i^{init}}(G)$ is the language of an effectively constructible finite automaton (Lemma 4). Hence each iterate $[\![\mathcal{P}]\!]^{(k)}$ can be thus computed by applying an intra-procedural analysis algorithm to the procedure-less program obtained by annotating this automaton with appropriate relations, given by the semantics of control words. For efficiency reasons, we give a modular construction of the k-th iterate of the under-approximation sequence, that uses the $(k-1)$-th iterate, for all $k > 1$. That is, the previously computed summaries are reused in the current iterate.

We are now ready to describe a source-to-source program transformation, from a recursive program to a non-recursive program, in which all

computation traces correspond to words generated by index-bounded derivations of the visibly pushdown grammar associated with the original program. Let $\mathcal{P}$ be a recursive program and $G_{\mathcal{P}} = \langle \Xi, \widehat{\Theta}, \Delta \rangle$ be its associated pushdown grammar, where each non-final state q of $\mathcal{P}$ is associated a nonterminal $Q \in \Xi$. Then, for a given constant $K > 0$, we define a *nonrecursive* program $\mathcal{H}^K$ that captures only the traces of $\mathcal{P}$ corresponding to K-index depth-first derivations of $G_{\mathcal{P}}$ (Algorithm 3). Formally, we define $\mathcal{H}^K = \langle \mathit{query}^0, \mathit{query}^1, \ldots, \mathit{query}^K \rangle$, i.e. the program is structured in $K+1$ procedures, such that:

- query^0 consists of a single statement **assume**(false), i.e. no execution going through a call of query^0 is possible,
- each procedure query^k, with the exception of query^0, issues calls only to query^{k-1}, for all $k > 1$,
- all executions of query^k, for each $1 \leq k \leq K$ correspond to k-index depth-first derivations of $G_{\mathcal{P}}$.

Each procedure query^k has five sets of local variables, all of the same cardinality as $\mathbf{x}$: two sets, named $\mathbf{x}_I$ and $\mathbf{x}_O$, are used as input variables, whereas the other three sets, named $\mathbf{x}_J, \mathbf{x}_K$ and $\mathbf{x}_L$ are used locally by query^k. Besides, each query^k has local variables called PC, τ, Y, Z and input variable X. There are no output variables in query^k.

First, the procedure query^k matches the current production rule to be simulated. To this end, we distinguish between productions of type (a) $Q \to \tau$, (b) $Q \to \tau Q'$ and (c) $Q \to \langle\!\langle \tau, Q_j^{init} \tau \rangle\!\rangle\, Q'$ (see Example 7) of the visibly pushdown grammar $G_{\mathcal{P}}$. Since Ξ and $\widehat{\Theta}$ are finite sets, we associate each nonterminal $Q \in \Xi$ an index $\mathcal{I}_Q \in \{1, \ldots, \|\Xi\|\}$, each symbol $\tau \in \widehat{\Theta}$ an index $\mathcal{I}_\tau \in \{1, \ldots, \|\widehat{\Theta}\|\}$, and identify the productions of $G_{\mathcal{P}}$ by the formulae:

$$
\begin{array}{rcl}
\pi_a(x,y) & \equiv & \bigvee_{(Q \to \tau) \in \Delta} x = \mathcal{I}_Q \wedge y = \mathcal{I}_\tau \\
\pi_b(x,y,z) & \equiv & \bigvee_{(Q \to \tau Q') \in \Delta} x = \mathcal{I}_Q \wedge y = \mathcal{I}_\tau \wedge z = \mathcal{I}_{Q'} \\
\pi_c(x,y,z,t,s) & \equiv & \bigvee_{(Q \to \langle\!\langle \tau Q_j^{init} \tau \rangle\!\rangle Q') \in \Delta} x = \mathcal{I}_Q \wedge y = \mathcal{I}_{\langle\!\langle \tau} \wedge z = \mathcal{I}_{Q_j^{init}} \wedge t = \mathcal{I}_{\tau \rangle\!\rangle} \wedge s = \mathcal{I}_{Q'}
\end{array}
$$

If the current production is of type (a) ($Q \to \tau$), the relation $\rho_\tau(\mathbf{x}_I, \mathbf{x}_O)$ must be checked before returning to the caller (query^{k+1}). For productions of type (b) ($Q \to \tau Q'$), the relation $\rho_\tau(\mathbf{x}_I, \mathbf{x}_J)$ is checked, before passing the values of $\mathbf{x}_J$ to $\mathbf{x}_I$ and moving on to the next production, that consumes the nonterminal Q'. The case of productions of type (c) ($Q \to \langle\!\langle \tau, Q_j^{init} \tau \rangle\!\rangle\, Q'$) is slightly more involved. The working variables $\mathbf{x}_J$, $\mathbf{x}_K$ and $\mathbf{x}_L$ are used to check the call $\rho_{\langle\!\langle \tau}(\mathbf{x}_I, \mathbf{x}_J)$, return $\rho_{\tau \rangle\!\rangle}(\mathbf{x}_K, \mathbf{x}_L)$ and frame $\phi_\tau(\mathbf{x}_I, \mathbf{x}_L)$ relations. Then the nondeterministic execution of query^k chooses between the (i) in-order execution, i.e. the derivation of Q_j^{init} before Q', corresponding to

the program label asgn_c^k, and (ii) out-of-order execution, i.e. the derivation of Q' before Q_j^{init}, corresponding to the program label swap^k.

Formally, each interprocedurally valid path of $query^k$, started with parameter $X = Q_i^{init}$, corresponds (modulo a language homomorphism) to a control word from $\Gamma^{\mathbf{df}(k)}_{Q_i^{init}}(G_{\mathcal{P}})$, labeling an index-bounded depth-first derivation of Q_i^{init}. For simplicity's sake, we do not give the details of this proof here, and refer the interested reader to [GIK12, Lemma 5].

Algorithm 3 proc $query^k(\mathrm{X}, \mathbf{x}_I, \mathbf{x}_O)$ for $1 \le k \le K$

var $\mathbf{x}_J, \mathbf{x}_K, \mathbf{x}_L$
var PC, τ, Y, Z
PC $\leftarrow X$
start^k: **goto** prod_a^k **or** prod_b^k **or** prod_c^k
prod_a^k: **havoc**(τ)
assume $\pi_a(\text{PC}, \tau)$ ▷ rule of type (a): $Q \to \tau$
asgn_a^k: **assume** $\rho_\tau(\mathbf{x}_I, \mathbf{x}_O)$
return
prod_b^k: **havoc**(τ, Y)
assume $\pi_b(\text{PC}, \tau, \text{Y})$ ▷ rule of type (b): $Q \to \tau\, Q'$
havoc($\mathbf{x}_J$)
assume $\rho_\tau(\mathbf{x}_I, \mathbf{x}_J)$
$\mathbf{x}_I \leftarrow \mathbf{x}_J$
asgn_b^k: PC $\leftarrow$ Y
goto start^k
prod_c^k: **havoc**(τ, Y, Z)
assume $\pi_c(\text{PC}, \langle\!\langle \tau, \text{Y}, \tau \rangle\!\rangle, \text{Z})$ ▷ rule of type (c): $Q \to \langle\!\langle \tau\, Q_j^{init}\, \tau \rangle\!\rangle\, Q'$
havoc($\mathbf{x}_J, \mathbf{x}_K, \mathbf{x}_L$)
assume $\rho_{\langle\!\langle \tau}(\mathbf{x}_I, \mathbf{x}_J)$ ▷ call relation
assume $\rho_{\tau \rangle\!\rangle}(\mathbf{x}_K, \mathbf{x}_L)$ ▷ return relation
assume $\phi_\tau(\mathbf{x}_I, \mathbf{x}_L)$ ▷ frame relation
goto swap^k **or** asgn_c^k
swap^k: **swap**(Y, Z)
swap($\mathbf{x}_J, \mathbf{x}_L$)
swap($\mathbf{x}_K, \mathbf{x}_O$)
asgn_c^k: $\mathbf{x}_I \leftarrow \mathbf{x}_L$
PC $\leftarrow$ Z
$query^{k-1}(\text{Y}, \mathbf{x}_J, \mathbf{x}_K)$
goto start^k

For example, let us consider the execution of the call $query^2(Q_1^{init}, (\,1\;0\,), (\,1\;2\,))$ following $Q_1^{init} \xRightarrow{p_1^b p_2^b p_3^c} \tau_1\tau_2\langle\tau_3 Q_1^{init}\tau_3\rangle Q_4 \xRightarrow{p_4^a} \tau_1\tau_2\langle\tau_3 Q_1^{init}\tau_3\rangle\tau_4 \xRightarrow{p_1^b p_5^b p_6^b p_7^a} \tau_1\tau_2\langle\tau_3\tau_1\tau_5\tau_6\tau_7\tau_3\rangle\tau_4$. In the table below, the first row (labelled PC) gives the value of local variable PC when control hits the labelled statement given at the second row (labelled ip). The third row (labelled $\mathbf{x}_I/\mathbf{x}_O$) represents the content of the two arrays. $\mathbf{x}_I/\mathbf{x}_O = (\,a\;b\,)(\,c\;d\,)$ says that, in $\mathbf{x}_I$, x has value a and z has value b; in $\mathbf{x}_O$, x has value c and z has value d.

PC	Q_1^{init}	–	Q_2	–	Q_3	–	–
ip	$\mathbf{start}^2$	$\mathbf{prod}_b^2\,(p_1^b)$	$\mathbf{start}^2$	$\mathbf{prod}_b^2\,(p_2^b)$	$\mathbf{start}^2$	$\mathbf{prod}_c^2\,(p_3^c)$	$\mathbf{swap}^2$
$\mathbf{x}_I/\mathbf{x}_O$	$(\,1\;0\,)(\,1\;2\,)$	$(\,1\;0\,)(\,1\;2\,)$	$(\,1\;0\,)(\,1\;2\,)$	$(\,1\;0\,)(\,1\;2\,)$	$(\,1\;0\,)(\,1\;2\,)$	$(\,1\;0\,)(\,1\;2\,)$	$(\,1\;0\,)(\,1\;2\,)$
PC	Q_4	–	Q_1^{init}	–	Q_2	–	Q_5
ip	$\mathbf{start}^1$	$\mathbf{prod}_a^1\,(p_4^a)$	$\mathbf{start}^2$	$\mathbf{prod}_b^2\,(p_1^b)$	$\mathbf{start}^2$	$\mathbf{prod}_b^2\,(p_5^b)$	$\mathbf{start}^2$
$\mathbf{x}_I/\mathbf{x}_O$	$(\,1\;0\,)(\,1\;2\,)$	$(\,1\;0\,)(\,1\;2\,)$	$(\,0\;0\,)(\,42\;0\,)$	$(\,0\;0\,)(\,42\;0\,)$	$(\,0\;0\,)(\,42\;0\,)$	$(\,0\;0\,)(\,42\;0\,)$	$(\,0\;0\,)(\,42\;0\,)$
PC	–	Q_6	–				
ip	$\mathbf{prod}_b^2\,(p_6^b)$	$\mathbf{start}^2$	$\mathbf{prod}_a^2\,(p_7^a)$				
$\mathbf{x}_I/\mathbf{x}_O$	$(\,0\;0\,)(\,42\;0\,)$	$(\,0\;0\,)(\,42\;0\,)$	$(\,0\;0\,)(\,42\;0\,)$				

The execution of $query^2(Q_1^{init}, (\,1\;0\,), (\,1\;2\,))$ starts on row 1, column 1 and proceeds until the call to $query^1(Q_4, (\,1\;0\,), (\,1\;2\,))$ at row 2, column 1 (the out of order case). The latter ends at row 2, column 2, where the execution of $query^2(Q_1^{init}, (\,1\;0\,), (\,1\;2\,))$ resumes. Since the execution is out of order, and the previous $\mathbf{havoc}(\mathbf{x}_J, \mathbf{x}_K, \mathbf{x}_L)$ results into $\mathbf{x}_J = (\,0\;0\,)$, $\mathbf{x}_K = (\,42\;0\,)$ and $\mathbf{x}_L = (\,1\;0\,)$ (this choice complies with the call relation), the values of $\mathbf{x}_I/\mathbf{x}_O$ are updated to $(\,0\;0\,)/(\,42\;0\,)$. The choice for equal values (0) of z in both $\mathbf{x}_I$ and $\mathbf{x}_0$ is checked in row 3, column 3.

The following theorem summarizes the result of this section, namely that any K-index under-approximation of the semantics of a recursive program $\mathcal{P}$ can be computed by looking at the semantics of a non-recursive program $\mathcal{H}^K$, obtained from $\mathcal{P}$ by a syntactic source-to-source transformation. For a valuation $\nu \in \mathbb{Z}^{\mathbf{x}}$ and a set of variables $\mathbf{y} \subset \mathbf{x}$, we denote by $\nu\!\downarrow_{\mathbf{y}}$ the restriction of ν to the variables in $\mathbf{y}$.

Theorem 15. *Let $\mathcal{P} = \langle P_1, \ldots, P_n\rangle$ be a program, $\mathbf{x} = \mathbf{x}_1 \cdot \ldots \cdot \mathbf{x}_n$ be the tuple of variables in $\mathcal{P}$, and let $q \in n\mathcal{F}(P_i)$ be a non-final control state of $P_i = \langle \mathbf{x}_i, \mathbf{x}_i^{in}, \mathbf{x}_i^{out}, S_i, q_i^{init}, F_i, \Delta_i\rangle$. Moreover, let $\mathcal{H}^K = \langle query^0, \ldots, query^K\rangle$ be the program defined by Algorithm 3. For any $1 \le k \le K$, we have:*

$$[\![\mathcal{P}]\!]_q^{(k)} = \{\langle (\widetilde{I}\!\downarrow_{\mathbf{x}_I}[\mathbf{x}/\mathbf{x}_I])\!\downarrow_{\mathbf{x}_i}, (\widetilde{I}\!\downarrow_{\mathbf{x}_O}[\mathbf{x}/\mathbf{x}_O])\!\downarrow_{\mathbf{x}_i}\rangle \mid \widetilde{I}\cdot\widetilde{O} \in [\![\mathcal{H}^K]\!]_{query^k}, \widetilde{I}(X) = Q\} \; .$$

Proof. See [GIK12, Theorem 1]. □

Reducing the computation of a program summary $[\![\mathcal{P}]\!]$ to that of computing a sequence of under-approximations $\left\{[\![\mathcal{P}]\!]^{(k)}\right\}_{k=1}^{\infty}$, that are defined

by a sequence of non-recursive programs $\left\{[\![\mathcal{H}^k]\!]\right\}_{k=1}^{\infty}$ does not yet solve the problem of computing each iterate $[\![\mathcal{P}]\!]^{(k)}$. In general, the summary of a non-recursive integer program is not expressible in a decidable fragment of arithmetic, due to the undecidability of the reachability problem for counter machines [Min67]. In the next section we identify a non-trivial case in which this computation is possible and, moreover, the under-approximation sequence is bound to stabilize after a number of iterations that is linear in the size (number of states) of $\mathcal{P}$.

5.3 Interprocedural Reachability Problems

In this section, we define a restriction of recursive integer programs, for which the k-index under-approximation sequence converges, and, moreover, computing each iterate of this sequence reduces to computing the set of reachable configurations of a flat counter machine (Chapter 4). We restrict the class of input programs by considering that all updates to the integer variables $\mathbf{x}$ are defined by octagonal constraints (Definition 4) and that every execution of the program belongs to the language of a given regular expression $w_1^* \dots w_d^*$, where the w_i's are finite sequences of program statements. These (not necessarily regular) sets are called *bounded languages* by Ginsburg and Spanier [GS64].

The reachability problem for this class of recursive programs is called *flat-octagonal reachability* (fo-reachability, for short) in the following. Concretely, given: a program $\mathcal{P}$ with procedures and local/global variables, whose statements are specified by octagonal constraints, and a bounded expression $\mathbf{b} = w_1^* \dots w_d^*$, where w_i's are sequences of statements of $\mathcal{P}$, the fo-reachability problem $\mathrm{REACH}_{fo}(\mathcal{P}, \mathbf{b})$ asks: can $\mathcal{P}$ run to completion by executing a sequence of program statements $w \in \mathbf{b}$? Studying the complexity of this problem provides the theoretical foundations for implementing efficient decision procedures, of practical interest in areas of software verification, such as bug-finding [EG11], or counterexample-guided abstraction refinement [KLW13, HIK+12].

The main idea is that, when considering only execution traces of $\mathcal{P}$ that belong to the language of $\mathbf{b}$, the set of control words producing these traces can be captured by a *bounded control set* $\Gamma_{\mathbf{b}} = \gamma_1^* \dots \gamma_e^*$, where the γ_i's are sequences of productions of $G_{\mathcal{P}}$. The control set $\Gamma_{\mathbf{b}}$ is thus the language of a flat finite automaton. By labeling the transitions of this automaton with octagonal relations (as in Algorithm 3) we obtain a flat counter machine with octagonal cycles. In the rest of this section, we show that the flat finite

automaton recognizing $\Gamma_{\mathbf{b}}$ can be built in time $|G_{\mathcal{P}}|^{\mathcal{O}(k)}$, where $|G_{\mathcal{P}}|$ is the size (number of productions) of $G_{\mathcal{P}}$ and k is the maximal index among the derivations induced by $\Gamma_{\mathbf{b}}$. Since, in general, it is the case that $L(G_{\mathcal{P}}) \cap \mathbf{b} = L^{(k)}(G_{\mathcal{P}}) \cap \mathbf{b}$ for $k = \mathcal{O}(|G_{\mathcal{P}}|)$ [Luk78], the size of the flat automaton recognizing $\Gamma_{\mathbf{b}}$ is simply exponential. As shown earlier (Theorem 10) the reachability problem can be solved in NPTIME in the size of this automaton, hence in NEXPTIME in the size of the recursive program. The NP-hardness follows from Theorem 10, considering an instance of the reachability problem for a non-recursive program.

5.3.1 Regular k-Index Depth-first Control Sets

It is known that the set $\Gamma_X^{\mathbf{df}(k)}(G)$ of k-index depth-first derivations of a grammar G, with axiom X, is recognizable by a finite automaton [Luk80, Lemma 5]. Below we give a formal definition of this automaton, that will be used to produce bounded control sets for covering the language of G. Given $k > 0$ and a grammar $G = \langle \Xi, \Sigma, \Delta \rangle$, we define a labeled graph $A_G^{\mathbf{df}(k)}$ such that its paths defines the set of k-index depth-first step sequences of G. This representation of a k-index depth-first control set is a crucial ingredient in the decidability proof for the fo-reachability problem, which is given in the next section.

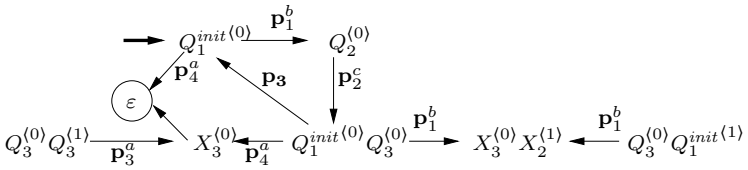
For a d-dimensional vector $\mathbf{v} \in \mathbb{N}^d$, we write $(\mathbf{v})_i$ for its i-th element $(1 \leq i \leq d)$. A vector $\mathbf{v} \in \mathbb{N}^d$ is said to be *contiguous* if $\{(\mathbf{v})_1, \ldots, (\mathbf{v})_d\} = \{0, \ldots, k\}$, for some $k \geq 0$. Given an alphabet Σ define the ranked alphabet $\Sigma^{\mathbb{N}}$ to be the set $\{\sigma^{\langle i \rangle} \mid \sigma \in \Sigma, i \in \mathbb{N}\}$. A ranked word is a word over a ranked alphabet. Given a word w of length n and an n-dimensional vector $\alpha \in \mathbb{N}^n$, the *ranked word* w^α is the sequence $(w)_1{}^{\langle(\alpha)_1\rangle} \ldots (w)_n{}^{\langle(\alpha)_n\rangle}$, in which the i-th element of α annotates the i-th symbol of w. We also denote $w^{\langle\langle c \rangle\rangle} = (w)_1{}^{\langle c \rangle} \ldots (w)_{|w|}{}^{\langle c \rangle}$ as a shorthand. Let $A_G^{\mathbf{df}(k)} = \langle Q, \Delta, \rightarrow \rangle$ be the following labeled graph, where:

$$Q = \{w^\alpha \mid w \in \Xi^*, |w| \leq k, \alpha \in \mathbb{N}^{|w|} \text{ is contiguous}, (\alpha)_1 \leq \cdots \leq (\alpha)_{|w|}\}$$

is the set of vertices, the edges are labeled by the set Δ of productions of G, and the edge relation is defined next. For all vertices $q, q' \in Q$ and labels $(X, w) \in \Delta$, we have $q \xrightarrow{(X,w)} q'$ if and only if:

- $q = u\, X^{\langle i \rangle}\, v$ for some u, v, where i is the maximum rank in q, and
- $q' = u\, v\, (w{\downarrow_\Xi})^{\langle\langle i' \rangle\rangle}$, where $|u\, v\, (w{\downarrow_\Xi})^{\langle\langle i' \rangle\rangle}| \leq k$ and $i' = \begin{cases} 0 & \text{if } u\, v = \varepsilon \\ i & \text{else if } (u\, v){\downarrow_{\Xi^{\langle i \rangle}}} = \varepsilon \\ i+1 & \text{else} \end{cases}$

Below we show the automaton $A^{\mathbf{df}(2)}_{G_\mathcal{P}}$ corresponding to the visibly pushdown grammar $G_\mathcal{P}$ that models the program $\mathcal{P}$ in Figure 5.1:



We denote by $|A^{\mathbf{df}(k)}_G| = \|Q\|$ the size (number of vertices) of $A^{\mathbf{df}(k)}_G$ and omit the subscript G, when the grammar is clear from the context. The size of a grammar $G = \langle \Xi, \Sigma, \Delta\rangle$ is defined as $|G| = \sum_{(X,w)\in\Delta}(|w|+1)$.

Lemma 4. *Given $G = \langle \Xi, \Sigma, \Delta\rangle$, and $k > 0$, for each $X \in \Xi$, $Y \in \Xi \cup \{\varepsilon\}$ and $\gamma \in \Delta^*$, we have $\gamma \in \Gamma^{\mathbf{df}(k)}_{X,Y}(G)$ if and only if $X^{\langle 0\rangle} \xrightarrow{\gamma} Y^{\langle 0\rangle}$ is a path in $A^{\mathbf{df}(k)}_G$. Moreover, we have $|A^{\mathbf{df}(k)}_G| \leq |G|^{2k}$.*

Proof. See [GI15a, Lemma 1]. □

5.3.2 A Decision Procedure for FO-reachability Problems

Let $G_\mathcal{P} = \langle \Xi, \widehat{\Theta}, \Delta\rangle$ be the visibly pushdown grammar that describes the program $\mathcal{P}$, $Q \in \Xi$ be the nonterminal corresponding to the initial state of $\mathcal{P}$, and let $\mathbf{b} = w_1^* \dots w_d^*$ be a bounded expression, where $w_1, \dots, w_d \in \widehat{\Theta}^*$. The general fo-reachability problem $\mathrm{REACH}_{fo}(\mathcal{P}, \mathbf{b})$ is equivalent to the question $\bigcup_{\alpha\in L_Q(G_\mathcal{P})\cap\mathbf{b}} [\![\alpha]\!] = \varnothing$. In addition, we define a k-index fo-reachability problem $\mathrm{REACH}^{(k)}_{fo}(\mathcal{P}, \mathbf{b})$ as: $\bigcup_{\alpha\in L^{(k)}_Q(G_\mathcal{P})\cap\mathbf{b}} [\![\alpha]\!] = \varnothing$. In this section we provide an algorithm that solves both problems, and provide complexity upper bounds. These bounds are tight for $\mathrm{REACH}^{(k)}_{fo}(\mathcal{P}, \mathbf{b})$, when k is a constant, not part of the input.

First, we compute, in time polynomial in the sizes of $\mathcal{P}$ and $\mathbf{b}$, a grammar $G^\cap = \langle \Xi^\cap, \widehat{\Theta}, \Delta^\cap\rangle$, such that $L_Q(G_\mathcal{P}) = \bigcup_{i=1}^{\ell} L_{X_i}(G^\cap)$, for some nonterminals $X_1, \dots, X_\ell \in \Xi^\cap$. The grammar $G^\cap$ is an automata-theoretic product between the grammar G and the bounded expression $\mathbf{b}$. To avoid clutter, we defer the formal definition of $G^\cap$ to the technical report [GI15b] and explain the construction by means of the example below.

Example 12. *Let us consider the bounded expression $\mathbf{b} = (ac)^*(ab)^*(db)^*$. Consider the grammar $G^\mathbf{b}$ with the following productions: $\mathrm{Q}_1^{(1)} \to a\,\mathrm{Q}_2^{(1)} \mid \varepsilon$, $\mathrm{Q}_1^{(2)} \to a\,\mathrm{Q}_2^{(2)} \mid \varepsilon$, $\mathrm{Q}_1^{(3)} \to d\,\mathrm{Q}_2^{(3)} \mid \varepsilon$, $\mathrm{Q}_2^{(1)} \to c\,\mathrm{Q}_1^{(1)} \mid c\,\mathrm{Q}_1^{(2)} \mid c\,\mathrm{Q}_1^{(3)}$, $\mathrm{Q}_2^{(2)} \to b\,\mathrm{Q}_1^{(2)} \mid b\,\mathrm{Q}_1^{(3)}$, $\mathrm{Q}_2^{(3)} \to b\,\mathrm{Q}_1^{(3)}$. It is easy to check that $\mathbf{b} = \bigcup_{i=1}^{3} L_{\mathrm{Q}_1^{(i)}}(G^\mathbf{b})$. Let*

$G = \langle \{X, Y, Z, T\}, \{a, b, c, d\}, \Delta \rangle$ *where* $\Delta = \{X \to aY,\ Y \to Zb,\ Z \to cT,\ Z \to \varepsilon,\ T \to Xd\}$, *i.e. we have* $L_X(G) = \{(ac)^n ab\,(db)^n \mid n \in \mathbb{N}\}$. *The following productions define a grammar* $G^{\cap}$*:*

$$
\begin{array}{llll}
[\mathrm{Q}_1^{(j)} X \mathrm{Q}_1^{(3)}] & \stackrel{p_1}{\to} & a\,[\mathrm{Q}_2^{(j)} Y \mathrm{Q}_1^{(3)}] & \\
[\mathrm{Q}_2^{(1)} Z \mathrm{Q}_2^{(3)}] & \stackrel{p_3}{\to} & c\,[\mathrm{Q}_1^{(j)} T \mathrm{Q}_2^{(3)}] & \\
[\mathrm{Q}_1^{(j)} T \mathrm{Q}_2^{(3)}] & \stackrel{p_5}{\to} & [\mathrm{Q}_1^{(j)} X \mathrm{Q}_1^{(3)}]\,d, \quad j = 1, 2 & \\
[\mathrm{Q}_2^{(2)} Y \mathrm{Q}_1^{(3)}] & \stackrel{p_7}{\to} & [\mathrm{Q}_2^{(2)} Z \mathrm{Q}_2^{(2)}]\,b & \\
[\mathrm{Q}_2^{(1)} Y \mathrm{Q}_1^{(3)}] & \stackrel{p_2}{\to} & [\mathrm{Q}_2^{(1)} Z \mathrm{Q}_2^{(3)}]\,b & \\
[\mathrm{Q}_2^{(2)} Z \mathrm{Q}_2^{(2)}] & \stackrel{p_4}{\to} & \varepsilon & \\
[\mathrm{Q}_1^{(2)} X \mathrm{Q}_1^{(3)}] & \stackrel{p_6}{\to} & a\,[\mathrm{Q}_2^{(2)} Y \mathrm{Q}_1^{(3)}] &
\end{array}
$$

One checks $L_X(G) = L_X(G) \cap \mathbf{b} = L_{[\mathrm{Q}_1^{(1)} X \mathrm{Q}_1^{(3)}]}(G^{\cap}) \cup L_{[\mathrm{Q}_1^{(2)} X \mathrm{Q}_1^{(3)}]}(G^{\cap})$. ■

A bounded expression $\mathbf{b} = w_1^* \dots w_d^*$ over alphabet Σ is said to be *d-letter-bounded* (or simply letter-bounded, when d is not important) when $|w_i| = 1$, for all $i = 1, \dots, d$. A letter-bounded expression $\widetilde{\mathbf{b}}$ is *strict* if all its symbols are distinct. A language $L \subseteq \Sigma^*$ is (strict, letter-) bounded iff $L \subseteq \mathbf{b}$, for some (strict, letter-) bounded expression $\mathbf{b}$.

Second, we reduce the problem from $\mathbf{b} = w_1^* \dots w_d^*$ to the strict letter-bounded case $\widetilde{\mathbf{b}} = a_1^* \dots a_d^*$, by building a grammar $G^{\bowtie}$, with the same non-terminals as $G^{\cap}$, such that, for each $X_i \in \Xi^{\cap}$, we have:

1. $L_{X_i}(G^{\bowtie}) \subseteq \widetilde{\mathbf{b}}$,
2. $w_1^{i_1} \dots w_d^{i_d} \in L_{X_i}^{(k)}(G^{\cap})$ iff $a_1^{i_1} \dots a_d^{i_d} \in L_{X_i}^{(k)}(G^{\bowtie})$, for all $k > 0$
3. from each control set $\widetilde{\Gamma}$ such that $L_{X_i}^{(k)}(G^{\bowtie} \subseteq \hat{L}_{X_i}(\widetilde{\Gamma}, G^{\bowtie})$, we compute, in polynomial time, a control set Γ such that $L_{X_i}^{(k)}(G^{\cap}) \subseteq \hat{L}_{X_i}(\Gamma, G^{\cap})$.

Example 13 (continued from Example 12)**.** *Let* $\mathcal{A} = \{a_1, a_2, a_3\}$, $\widetilde{\mathbf{b}} = a_1^* a_2^* a_3^*$ *and* $h\colon \mathcal{A} \to \Sigma^*$ *be the homomorphism given by* $h(a_1) = ac, h(a_2) = ab$ *and* $h(a_3) = db$. *The grammar* $G^{\bowtie}$ *results from deleting* a*'s and* d*'s in* $G^{\cap}$ *and replacing* b *in* p_2 *by* a_3, b *in* p_7 *by* a_2 *and* c *by* a_1. *Then, it is easy to check that* $h^{-1}(L_X(G)) \cap \widetilde{\mathbf{b}} = L_{[\mathrm{Q}_1^{(1)} X \mathrm{Q}_1^{(3)}]}(G^{\bowtie}) \cup L_{[\mathrm{Q}_1^{(2)} X \mathrm{Q}_1^{(3)}]}(G^{\bowtie}) = \{a_1^n\, a_2\, a_3^n \mid n \in \mathbb{N}\}$. ■

Third, for the strict letter-bounded grammar $G^{\bowtie}$, we compute a control set $\widetilde{\Gamma} \subseteq (\Delta^{\bowtie})^*$ using the result of the theorem below:

Theorem 16. *For a grammar* $G = \langle \Xi, \mathcal{A}, \Delta \rangle$*, and* $X \in \Xi$*, such that* $L_X(G) \subseteq \widetilde{\mathbf{b}}$*, where* $\widetilde{\mathbf{b}}$ *is the strict* d*-letter bounded expression for* $L_X(G)$*, for each* $k > 0$*, there exists a finite set of bounded expressions* $\mathcal{S}_{\widetilde{\mathbf{b}}}$ *over* Δ *such that* $L_X^{(k)}(G) \subseteq \hat{L}_X(\bigcup \mathcal{S}_{\widetilde{\mathbf{b}}} \cap \Gamma_X^{\mathbf{df}(k+1)}, G)$*. Moreover,* $\mathcal{S}_{\widetilde{\mathbf{b}}}$ *can be constructed in time* $|G|^{\mathcal{O}(k)+d}$ *and each* $\widetilde{\Gamma} \in \mathcal{S}_{\widetilde{\mathbf{b}}}$ *can be constructed in time* $|G|^{\mathcal{O}(k)}$.

Proof. See [GI15a, Theorem 3]. □

The main ingredient of the proof is a decomposition[2] of a k-index depth-first derivation with control word γ into a $(k+1)$-index depth-first derivation, consisting of a prefix $\gamma^\sharp$ producing a word in $a_1^* a_d^*$ and a production (X_i, w) followed by two control words γ' and γ'' that produce words contained within two bounded expressions $a_\ell^* \dots a_m^*$ and $a_m^* \dots a_r^*$, respectively, where $\max(m-\ell, r-m) < d-1$. This reduces the problem of computing a bounded control set for $\widetilde{\mathbf{b}} = a_1^* \dots a_d^*$ to the constant case $d = 2$, for which we prove that the control set can be computed in time $|G|^{\mathcal{O}(k)}$.

The time needed to build a representation of each control set $\Gamma_{i,j}$ is of the order of $|G|^{\mathcal{O}(|G|)} = 2^{\mathcal{O}(|G| \log |G|)}$. To decide the fo-reachability problem $\mathrm{REACH}_{fo}(\mathcal{P}, \mathbf{b})$, a nondeterministic algorithm first chooses such a control set $\Gamma_{i,j}$, then decides the ReachFlat(OCT) problem for the flat CM obtained by labeling the finite automaton recognizing $\Gamma_{i,j}$ with octagonal constraints, as described in Section 5.2. The fo-reachability problem for this CM is decidable in Nptime in the size of the CM, thus $\mathrm{REACH}_{fo}(\mathcal{P}, \mathbf{b})$ is in NExptime. The restricted k-index fo-reachability problem $\mathrm{REACH}_{fo}^{(k)}(\mathcal{P}, \mathbf{b})$ becomes Np-complete, when k is constant.

Theorem 17. *Let $\mathcal{P}$ be an octagonal program, $G_{\mathcal{P}} = \langle \Xi, \widehat{\Sigma}, \Delta \rangle$ be its corresponding visibly pushdown grammar and $\mathbf{b} = w_1^* \dots w_d^*$ be a bounded expression, where $w_1, \dots, w_d \in \widehat{\Sigma}^*$. Then the problem $\mathrm{REACH}_{fo}(\mathcal{P}, \mathbf{b})$ is decidable in NExptime with an Np-hard lower bound, and $\mathrm{REACH}_{fo}^{(k)}(\mathcal{P}, \mathbf{b})$ is Np-complete if $k > 0$ is a constant.*

Proof. See [GI15b, Theorem 2]. □

5.4 Discussion and Open Problems

The under-approximation method described in this chapter can be seen as a generalization of the acceleration technique to recursive integer programs. Moreover, the completeness result based on the restriction of the set of interprocedurally valid execution traces to a bounded language is nothing but the generalization of the flatness restriction from Chapter 4 to the recursive case. The main open problem is, currently, closing the gap between the Nexptime and Np bounds for the reachability problem for bounded programs with octagonal constraints. Another, equally interesting, problem is considering other classes of relations, such as affine relations with the finite monoid condition [Boi99, FL02b].

[2] Inspired by a decomposition given by Ginsburg [Gin66, Chapter 5.3, Lemma 5.3.3].

Chapter 6

Program Verification

In this chapter we describe several applications of counter machines to program verification. We consider programs that manipulate higher-order data structures such as lists, trees and arrays, by developing verification methods targeted to the particular data types.

First, we distinguish programs with dynamically allocated data structures and low-level pointer manipulations from programs with statically declared arrays. Within the former class, we consider programs with singly-linked lists, that are independent of the data stored in the list cells, and reduce their verification problems (safety and termination) to equivalent (reachability and termination) problems on counter machines. The idea of this reduction is that the set of reachable memory configurations can be captured by a finite-range abstraction (a finite number of shape graphs). This translation allows to use available tools for the verification of counter machines to deal with programs with dynamic memory and, moreover, to derive several decidability results for the verification of programs with lists.

Second, we consider programs with dynamically allocated tree-shaped data structures. In this case, the set of reachable memory configurations can be over-approximated by a tree automaton [BHRV06]. The main limitation of using tree automata is that they cannot capture sets of balanced trees (AVL and RED-BLACK search trees), that are widely used within programs such as operating system kernels and databases. We tackle this problem by defining a new class of tree automata, called *tree automata with size constraints* (TASC) which are closed under boolean operations (union, intersection and complement) and whose emptiness problem is decidable. Based on this novel class of tree automata, we develop a Hoare-style deductive verification approach, based on a strongest post-condition calculus.

Finally, we consider programs handling array structures that store integer values. Reasoning about this class of programs requires expressive logics that allow to encode universal properties of unbounded array segments. Our focus here is on defining such logics that have nice algorithmic properties, such as decidability of entailments. We describe two such logics, called *singly-indexed logic* (SIL) and *logic of integer arrays* (LIA), whose decidability is established by reduction to the reachability problems of flat counter machines with octagonal constraints (Chapter 4).

Let us start by coining a few notions formally. For a partial function $f : A \rightharpoonup B$, and $\bot \notin B$, we denote by $f(x) = \bot$ the fact that f is undefined at some point $x \in A$. The domain of f is denoted $\mathrm{dom}(f) = \{x \in A \mid f(x) \neq \bot\}$. By $f : A \rightharpoonup_{fin} B$, we denote any partial function whose domain is finite.

We consider $\mathsf{Var} = \{u, v, w, \ldots\}$ and $\mathsf{Loc} = \{\ell, m, n, \ldots\}$ to be countably infinite sets of *program variables* and *memory locations*, respectively. Let $\mathsf{nil} \in \mathsf{Var}$ be a designated variable, $\mathsf{null} \in \mathsf{Loc}$ be a designated location and $\mathsf{Sel} = \{1, \ldots, k\}$ be a finite set of natural numbers, called *selectors*.

Definition 9. *A* state *is a pair* $\langle s, h \rangle$ *where* $s : \mathsf{Var} \rightharpoonup_{fin} \mathsf{Loc}$ *is a partial function mapping program variables into locations such that* $s(\mathsf{nil}) = \mathsf{null}$*, and* $h : \mathsf{Loc} \rightharpoonup_{fin} \mathsf{Sel} \rightharpoonup_{fin} \mathsf{Loc}$ *is a finite partial function such that* $\mathsf{null} \notin \mathrm{dom}(h)$ *and for all* $\ell \in \mathrm{dom}(h)$ *there exists* $i \in \mathsf{Sel}$ *such that* $(h(\ell))(i) \neq \bot$.

For a state $\langle s, h \rangle$, s is called the *store* and h the *heap*. We write $\ell \xrightarrow{i} \ell'$ instead of $(h(\ell))(i) = \ell'$ and call this a *selector edge* of the state $\langle s, h \rangle$.

We distinguish heaps based on the number of selectors, which is determined by the type of the data structure used. If $\|\mathsf{Sel}\| = 1$, i.e. each location has at most one outgoing selector edge, the heap consists of *singly-linked lists* only, possibly with sharing and circularities. Otherwise, if $\|\mathsf{Sel}\| > 1$ and there is no sharing (no two edges $\ell_1 \xrightarrow{k_1} \ell$ and $\ell_2 \xrightarrow{k_2} \ell$), the heap consists of *trees* only. Finally, if $\|\mathsf{Sel}\| > 1$ and sharing is allowed, we are in the general case of unrestricted graph structures.

6.1 Programs with Lists

In this section, we define a model for imperative programs manipulating dynamic singly-linked list data structures. We assume that lists are implemented using data types with one selector, typically called next, as it is the case in most common imperative programming languages. We consider programs without function calls and concurrency constructs, therefore all variables are assumed to be global. In addition to the list data structures,

the programs we consider can have integer variables. Simple examples of such programs include list reversion, list insertion, sorting procedures, programs counting the elements in a list, etc.

The abstract syntax of the programs considered is given in Figure 6.1 (a). We use Lab to denote a finite set of program labels (control locations), Var to denote a finite set of pointer variables, and IVar to represent a finite set of integer variables (counters). Pointers can be used in assignments such as u := nil, u := w, and u := w.next; selector updates u.next := w and u.next := nil; and new cell creation u := new. Counters can be incremented $i := i + 1$, decremented $i := i - 1$, and reset $i := 0$. The control structure is composed of iteration (`while`) statements and conditional (`if-then-else`) statements. The guards of the control statements are pointer equality u = w and u = nil, zero tests for counters $i = 0$, and boolean combinations of the above. An example of a simple program from the considered class is the list reversal program in Figure 6.1 (b).

$$
\begin{array}{rcl}
\ell & \in & \mathsf{Lab} \\
\mathsf{u}, \mathsf{v} & \in & \mathsf{Var} \\
i & \in & \mathsf{IVar} \\
\mathit{Prog} & := & \{\ell : \mathit{Stm};\}^* \\
\mathit{Stm} & := & \mathit{While} \mid \mathit{IfElse} \mid \mathit{Asgn} \\
\mathit{While} & := & \texttt{while}\ \mathit{Guard}\ \texttt{do}\ \{\mathit{Stm};\}^*\ \texttt{od} \\
\mathit{IfElse} & := & \texttt{if}\ \mathit{Guard}\ \texttt{then}\ \{\mathit{Stm};\}^*\ \big[\texttt{else}\ \{\mathit{Stm};\}^*\big]\ \texttt{fi} \\
\mathit{Asgn} & := & \mathsf{u} := \mathsf{nil} \mid \mathsf{u} := \mathsf{new} \mid \mathsf{u} := \mathsf{v} \mid \mathsf{u} := \mathsf{v.next} \mid \\
 & & \mathsf{u.next} := \mathsf{nil} \mid \mathsf{u.next} := \mathsf{v} \mid i := 0 \mid i := i \pm 1 \\
\mathit{Guard} & := & \mathsf{u} = \mathsf{v} \mid \mathsf{u} = \mathsf{nil} \mid i = 0 \mid \neg \mathit{Guard} \mid \mathit{Guard} \wedge \mathit{Guard}
\end{array}
$$

(a)

```
1: while u ≠ nil do
2:     w := u.next;
3:     u.next := v;
4:     v := u;
5:     u := w;
6:     do
```

(b)

Figure 6.1: Abstract syntax of programs with lists

The semantics of a program is described as a stepwise transformation of a state (Definition 9). To keep the presentation succint, we do not give the rules of the operational semantics here and point the reader to [BBH+11, Section 2.2] for a formal definition thereof.

Given a state $\langle s, h \rangle$, a *cut point* is a location $\ell \in \mathsf{Loc}$ such that either (i) there exist two distinct locations $\ell_1, \ell_2 \in \mathsf{Loc}$ such that $\ell_1 \xrightarrow{\mathsf{next}} \ell$ and $\ell_2 \xrightarrow{\mathsf{next}} \ell$, or (ii) there exists a variable $\mathsf{u} \in \mathsf{Var}$ such that $s(\mathsf{u}) = \ell$. This notion induces an equivalence relation on locations, defined as follows. Let $\sim$ be the reflexive, symmetric and transitive closure of the relation $\lhd \subseteq \mathsf{Loc} \times \mathsf{Loc}$, where $\ell \lhd \ell'$ iff $\ell \xrightarrow{\mathsf{next}} \ell'$ and ℓ' is not a cut point. For a location ℓ, we denote by $[\ell]$ its equivalence class with respect to $\sim$ (also called a *list segment*) and

define the *quotient state* $\langle s_{/\sim}, h_{/\sim}\rangle$ as follows:
- for all $\mathsf{u} \in \mathsf{Var}$, we have $s_{/\sim}(\mathsf{u}) = [\ell]$ if $s(\mathsf{u}) = \ell$,
- for all $\ell, \ell \in \mathsf{Loc}$, we have $[\ell] \xrightarrow{\mathsf{next}} [\ell']$ if there exist $\ell_0 \in [\ell]$ and $\ell'_0 \in [\ell']$ such that $\ell_0 \xrightarrow{\mathsf{next}} \ell'_0$ and ℓ'_0 is a cut point of the state $\langle s, h\rangle$.

For technical convenience, we assume that each location in the domain of the heap is reachable, via a set of edges, from a program variable, i.e. the heap does not contain garbage nodes. We are now ready to define a finite-range abstraction of states:

Definition 10. *A* symbolic shape graph *(SSG) is a directed graph* $S = \langle N, E, V\rangle$*, where* N *is a set of vertices,* $E : N \rightharpoonup_{fin} N$ *is a successor mapping and* $V : \mathsf{Var} \rightharpoonup_{fin} N$ *is a variable mapping.*

In analogy with concrete states (Definition 9), we say that a vertex $v \in N$ of an SSG $S = \langle N, E, V\rangle$ is a cut point if (i) there exist two distinct vertices $n_1, n_2 \in N$ such that $E(n_1) = E(n_2) = n$, or (ii) there exist a variable $\mathsf{u} \in \mathsf{Var}$ such that $V(\mathsf{u}) = n$. An SSG is in *normal form* if each vertex is a cut point and it is reachable from a variable. The main idea is that the number of SSGs in normal form is finite:

Lemma 5. *If* $\langle N, E, V\rangle$ *is an SSG in normal form, then* $\|N\| \leq 2\,\|\mathrm{dom}(V)\|$*. Moreover, for a given constant* $n > 0$*, the number of SSGs in normal form such that* $\|\mathrm{dom}(V)\| \leq n$ *is of the order of* $2^{\Theta(n \log n)}$*.*

Proof. See [BBH+11, Lemma 1]. □

A SSG $\langle N, E, V\rangle$ is an *abstraction* of a state $\langle s, h\rangle$ if there exists a bijection $\beta : \mathrm{dom}(h_{/\sim}) \cup \{\bot\} \to N \cup \{\bot\}$ such that $\beta(\bot) = \bot$ and:
- for all $\ell \in \mathsf{Loc}$, we have $E(\beta([\ell])) = \beta(h_{/\sim}([\ell]))$, and
- for all $\mathsf{u} \in \mathsf{Var}$, we have $V(\mathsf{u}) = \beta(s_{/\sim}(\mathsf{u}))$.

In other words, each list segment is mapped into a vertex of the SSG and this mapping is an isomorphism between the quotient state and the SSG. This abstraction comes, however, with loss of information. Namely, we lose track of the number of locations in each list segment of the concrete state.

To recover this information, we associate an integer counter with each vertex and build, for each program written using the syntax from Figure 6.1 (a), a counter machine whose semantics is in bisimulation with the concrete semantics of the program. This entails a strong preservation of temporal logic properties. In particular, safety and termination are strongly preserved by the resulting counter machine, meaning that one can prove (disprove) safety and termination properties of such programs based on the behavior of their counter machines, obtained by the following translation.

Consider a program with lists that uses a subset $\mathcal{V} \subseteq \mathsf{Var}$ of pointer variables, such that $\|\mathcal{V}\| = K$, and N integer variables $i_1, \ldots, i_N \in \mathsf{IVar}$. We build a counter machine $M = \langle \mathbf{x}, Q, \iota, F, \Delta \rangle$, where:

- $\mathbf{x}$ contains a variable x_n for each vertex $n \in N$ of an SSG $S = \langle N, E, V \rangle$ in normal form, such that $\mathrm{dom}(V) \subseteq \mathcal{V}$ and, moreover, $i_1, \ldots, i_N \in \mathbf{x}$,
- Q is the set of pairs $\langle \ell, S \rangle$, where $\ell \in \mathsf{Lab}$ is a label of the program and $S = \langle N, E, V \rangle$ is an SSG in normal form such that $\mathrm{dom}(V) \subseteq \mathcal{V}$,
- $\iota = \langle \ell_0, S_0 \rangle$ where ℓ_0 is the initial label of the program and S_0 is the SSG representing the set of initial states,
- F is the set of pairs $\langle \ell_f, S \rangle$, where ℓ_f is a final label of the program,
- Δ is defined by several rules (see [BBH+11, Section 4.1] for a complete formalization). Given a state $q = \langle \ell, S \rangle$ the set of successor states $q' = \langle \ell', S' \rangle$ is defined based on the type of the current program statement at ℓ and the structure of S. Due to the rather large number of possible cases, we chose not to give the formal description here and rely on the following example to illustrate the principle.

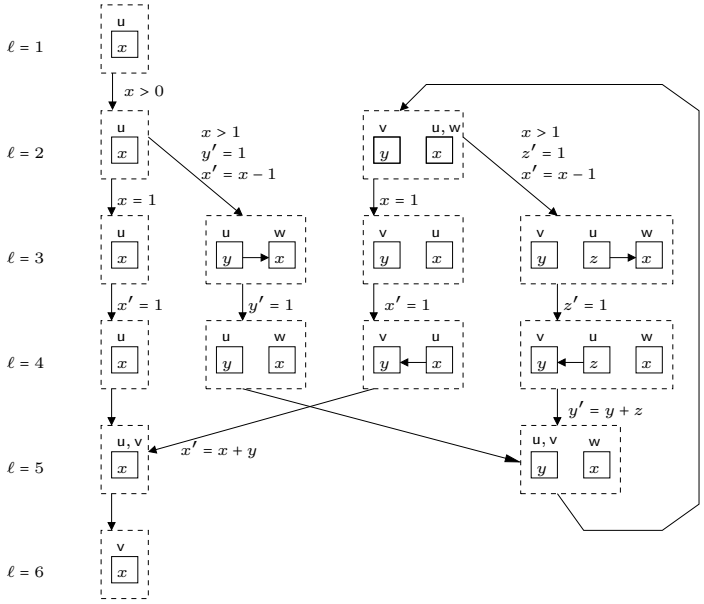


Figure 6.2: Non-circular list reversal

Example 14. *Figure 6.2 shows the counter automaton obtained from the list reversal program from Figure 6.1 (b), started with a non-circular list pointed to by* u *as its input. The counter variable corresponding to each abstract node is depicted inside the node itself. Variables pointing to* null *are*

omitted. For each label of the program the corresponding control states are those reached after the execution of the statement at that label. ■

The first statement (`while u ≠ nil`) enters the main cycle if the number of nodes in the list segment pointed to by `u` is greater than 0 ($x > 0$). The assignment `w := u.next` at line 2 causes a case split based on the size of this list segment. If $x = 1$ (the left branch in Figure 6.2) `w` is assigned to `null` and we do not represent it in the shape graph in line 2 (left). Otherwise, if $x > 1$, this assignment causes a split of the list segment pointed to by `u` into two parts: one pointed to by `u`, of size $y = 1$ and the rest, pointed to by `w`, of size $x - 1$ (line 3). The assignment `v := u` on line 4 merges two list segments, pointed to by `u` and `v`, respectively, in case the two list segments are in the successor relation, according to the SSG (line 4, 3rd column). In this case, we also add their sizes ($x' = x + y$). The rest of the transitions of the counter machine can be understood among the same lines. The main property of this transformation is given by the next theorem:

Theorem 18 ([BBH+06, BBH+11])**.** *The semantics of a list program is in bisimulation with the semantics of its corresponding counter machine.*

Proof. See [BBH+11, Theorem 2]. □

As a direct consequence, a safety (reachability) or termination property of a program with singly-linked lists can be decided on the counter machine obtained using this transformation, using available tools for reachability and termination of counter machines. Observe, however, that the resulting counter machine is not flat, even when the input program is flat, as shown by Example 14. The problem is caused by the assignment statements `u := v.next` that cause the control of the counter machine to branch, according to whether the counter of the list segment pointed to by `v` is one or greater than one. In the next section we tackle this problem by introducing a different translation scheme, for a restriction of list programs, that preserves the flatness of the control structure thereof.

6.1.1 Undecidability Results

We focus next on establishing a boundary between decidability and undecidability for several classes of decision problems concerning programs working on singly-linked lists. We consider essentially *safety* properties, specified by inserting assertions `assert(`*Guard*`)` in the text of the program and *termination*, i.e. the absence of infinite computations. Moreover, we consider *flat programs*, whose control structure does not contain nested `while` cycles and `if-then-else` statements inside `while` cycles, and restrict the set of program

statements, by prohibiting *destructive updates*, namely the statements of the form $\mathsf{u} := \mathsf{new}$, $\mathsf{u.next} := \mathsf{v}$ and $\mathsf{u.next} := \mathsf{nil}$. Observe that these programs can traverse the input structure, by moving several iterator variables down the lists, but cannot change the structure itself.

Surprisingly, even with these restrictions, safety and termination properties remain undecidable for programs with lists. The source of the undecidability is the complexity of the input heap structure. We show that, as soon as the input structure contains more than one cycle. The next section shows that safety and termination become decidable, if the input heap is restricted to having at most one cycle, which establishes a sharp decidability boundary for this class of programs.

The undecidability results are by reduction from the satisfiability problem for the quantifier-free fragment of the theory $\langle \mathbb{N}, +, \mathrm{lcm} \rangle$, where $\mathrm{lcm}(x, y)$ denotes the least common multiple of x and y. This fragment is undecidable, as a consequence of the undecidability of Hilbert's Tenth Problem [Mat70] and the following encoding of multiplication using addition and the least common multiple: $x^2 = y \Leftrightarrow y + x = \mathrm{lcm}(x, x+1)$ and $x \cdot y = z \Leftrightarrow 2z + x^2 + y^2 = (x+y)^2$. Since this reduction does not introduce quantifiers, any quantifier-free formula of $\langle \mathbb{N}, +, \cdot \rangle$ can be translated into a quantifier-free formula of $\langle \mathbb{N}, +, \mathrm{lcm} \rangle$, possibly by introducing additional variables. Observe moreover that the validity problem $\forall \mathbf{x} \,.\, \phi(\mathbf{x})$ is also undecidable, when ϕ is a quantifier-free formula of $\langle \mathbb{N}, +, \mathrm{lcm} \rangle$, because this problem is equivalent to the satisfiability of $\neg\phi(\mathbf{x})$.

Definition 11. *A* parametric shape graph *(PSG) is a directed graph* $S = \langle \mathbf{x}, N, E, V, X \rangle$*, where* $\langle N, E, V \rangle$ *is a symbolic shape graph,* $\mathbf{x}$ *is a set of integer variables and* $X : N \to \mathbf{x}$ *is a one-to-one mapping of vertices with integer variables.*

Given a PSG $S = \langle \mathbf{x}, N, E, V, X \rangle$ and a valuation $\nu : \mathbf{x} \mapsto \mathbb{N}_+$, we denote by $[\![S]\!]_\nu$ the state obtained by replacing each vertex $v \in N$ with a list segment of length $\nu(X(v))$. Moreover, we define $[\![S]\!] = \{[\![S]\!]_\nu \mid \nu : \mathbf{x} \mapsto \mathbb{N}_+\}$.

For any quantifier-free formula $\phi(\mathbf{x})$ in the theory $\langle \mathbb{N}, +, \mathrm{lcm} \rangle$, we build a flat program with lists P_ϕ and a parametric shape graph S_ϕ such that $\forall \mathbf{x} \,.\, \phi(\mathbf{x})$ holds if and only if P_ϕ, started on any state $\langle s, h \rangle \in [\![S_\phi]\!]$, does not violate any of its assertions and terminates. The reduction uses the gadgets from Figure 6.3, as we explain next.

First, notice that the gadget programs $P_{x=y}$, $P_{x+y=z}$ and $P_{z=\mathrm{lcm}(x,y)}$ terminate. In particular, when P_φ terminates and C_φ holds, then it must be the case that the formula $\varphi(x, y, z)$ holds as well.

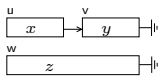
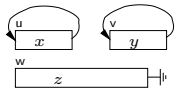
φ	P_φ	C_φ	S_φ
$x = y$	1: i := u; 2: j := w; 3: while i ≠ null ∧ j ≠ null do 4: i := i.next; 5: j := j.next; 6: od	i = null ∧ j = null	
$x + y = z$	1: i := u; 2: j := w; 3: while i ≠ null ∧ j ≠ null do 4: i := i.next; 5: j := j.next; 6: od;	i = null ∧ j = null	
$z = \text{lcm}(x, y)$	1: i := u.next; 2: j := v.next; 3: k := w.next; 4: while (i ≠ u ∨ j ≠ v) ∧ k ≠ null do 5: i := i.next; 6: j := j.next; 7: k := k.next; 8: od;	k = null ∧ i = u ∧ j = v	

Figure 6.3: Gadgets for proving undecidability

We assume w.l.o.g. that the formula $\phi(\mathbf{x})$ is given in conjunctive normal form and we check the validity of each of its clauses. Let $\bigvee_{i=1}^{n} \psi_i(\mathbf{x})$ be such a clause, i.e. a disjunction of atomic propositions of the form $x = y$, $x + y = z$ or $z = \text{lcm}(x, y)$, or their negations. Moreover, we consider a separate PSG S_{ψ_i} as in Figure 6.3. The program checking the validity of the clause is the following:

P_{ψ_1}; if $C_{\neg\psi_1}$ then P_{ψ_2};
if $C_{\neg\psi_2}$ then P_{ψ_3};
...
assert(false);
...
fi
fi

where, for all $1 \leq i \leq n$ we have:
- if ψ_i is a positive literal, P_{ψ_i} and C_{ψ_i} are as in Figure 6.3,
- if ψ_i is a negative literal, P_{ψ_i} is $P_{\neg\psi_i}$ and C_{ψ_i} is $\neg C_{\neg\psi_i}$.

Moreover, the program has to test that all list segments encoding occurrences of the same variable are of the same length. This can be done in the beginning, using a sequence of flat programs of the same kind as $P_{x=y}$, and is skipped for brevity reasons. It is not hard to see that the assertion

`assert(false)` is reached if and only if the clause $\bigvee_{i=1}^{n} \psi_i$ is not valid.

To show undecidability of the termination problem, we use the same reduction, with the only difference that the `assert(false)` statement is replaced by a non-terminating loop `while(true) do...od`. The program then terminates if and only if the clause $\bigvee_{i=1}^{n} \psi_i$ is valid.

Observe that the least common multiple relation has been encoded using an input heap with at least two separate cycles. The above considerations lead to the following theorem:

Theorem 19 ([BI07])**.** *The safety and termination problems for flat list programs without destructive updates, running on input heaps with more than one cycle are undecidable.*

Proof. See [BI07, Theorem 2]. □

6.1.2 Decidability Results

Assuming that the input heap of the program has at most one cycle, we show that the safety and termination problems become decidable, by reduction to the reachability and termination problems of a class of counter machines, with deterministic affine transitions and guards defined by $\mathsf{L}^1_{\text{div}}$ formulae (Chapter 3). The translation scheme used to produce these counter machines is different from the one described previously, in that it preserves the flatness of the control structure.

The idea of the encoding is to fix a set of *root* variables, call them $\mathsf{r}_1, \ldots, \mathsf{r}_p$, that are not changed by the program, such that every node in the input PSG is reachable from at least one root variable. The rest of the variables in the program, called *iterators*, are denoted as $\mathsf{u}_1, \ldots, \mathsf{u}_r$. At each step during the execution of the program, the position of an iterator u_i is uniquely determined by a root variable r_j and the distance (number of successor edges) from r_j to u_i. This distance is kept in a counter variable y_i. The resulting counter machine is $M = \langle \mathbf{y}, Q, \iota, F, \Delta \rangle$, where:

- $\mathbf{y} = \{y_1, \ldots, y_r\}$, where y_j is the counter associated with the iterator u_j,
- $Q = \mathsf{Lab} \times \{\mathsf{r}_1, \ldots, \mathsf{r}_p, \bot\}^r$, each state is a tuple $\langle \ell, \mathsf{r}_{i_1}, \ldots, \mathsf{r}_{i_r} \rangle$, where r_{i_j} is the current root of the iterator u_j, or $\bot$ is u_j is null,
- $\iota = \langle \ell_0, \mathsf{r}_{i_1^0}, \ldots, \mathsf{r}_{i_r^0} \rangle$, ℓ_0 is the initial label and $\mathsf{r}_{i_j^0}$ is the initial root of u_j,
- $F = \{\ell_1^f, \ldots, \ell_k^f\} \times \{\mathsf{r}_1, \ldots, \mathsf{r}_p, \bot\}^r$, where $\ell_1^f, \ldots, \ell_k^f$ are the final labels,
- Δ is the set of rules $\langle \ell, \mathsf{r}_{i_1}, \ldots, \mathsf{r}_{i_r} \rangle \xrightarrow{\varphi(\mathbf{y}, \mathbf{y}')} \langle \ell', \mathsf{r}_{i'_1}, \ldots, \mathsf{r}_{i'_r} \rangle$ corresponding to program statements, as follows:
  - $\mathsf{u}_j := \mathsf{nil}$ yields $\varphi \equiv y'_j = 0 \wedge \mathrm{Id}_j$, $\mathsf{r}_{i'_j} = \bot$ and $i'_k = i_k$ for all $k \neq j$,

– $\mathsf{u}_j := \mathsf{u}_h$ yields $\varphi \equiv y'_j = y_h \wedge \mathrm{Id}_j$, $i'_j = i_h$ and $i'_k = i_k$ for all $k \neq j$,
– $\mathsf{u}_i := \mathsf{u}_h.\mathsf{next}$ yields $\varphi \equiv y'_j = y_h + 1 \wedge \mathrm{Id}_j$, $i'_j = i_h$ and $i'_k = i_k$ for all $k \neq j$.

where $\mathrm{Id}_j \equiv \bigwedge_{k \neq j} y'_k = y_k$.

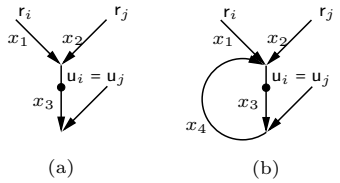


Figure 6.4: Iterator equalities

A guard (a boolean combination of propositions of the form $\mathsf{u}_i = \mathsf{nil}$ or $\mathsf{u}_i = \mathsf{u}_j$) has a more complex encoding, due to the fact that the position on an iterator in the heap can be represented in several ways. Let us consider the more interesting case $\mathsf{u}_i = \mathsf{u}_j$, depicted in Figure 6.4, where $V(\mathsf{u}_i) = V(\mathsf{u}_j) = n_0 \in N$ is the node pointed to by both u_i and u_j in the current PSG $S = \langle \mathbf{x}, N, E, V, X \rangle$. Let r_i and r_j denote the roots of u_i and u_j, respectively. We distinguish the following cases:

1. n_0 does not belong to a cycle. In this case there are unique paths $\pi_i : V(\mathsf{r}_i) = n^i_1, n^i_2, \ldots, n^i_{k_i} = V(\mathsf{u}_i) = n_0$ and $\pi_j : V(\mathsf{r}_j) = n^j_1, n^j_2, \ldots, n^j_{k_j} = V(\mathsf{u}_j) = n_0$ in S. The equality $\mathsf{u}_i = \mathsf{u}_j$ can be encoded as:

$$y_i - (\sum_{n \in \pi_i} X(n)) = y_j - (\sum_{n \in \pi_j} X(n)) \ .$$

   For instance, the condition $\mathsf{u}_i = \mathsf{u}_j$ in the PSG from Figure 6.4 (a) translates to $x_i - x_1 = x_j - x_2$.
2. n_0 belongs to the only cycle in S, call this γ, and let π_i and π_j be the paths from r_i and r_j to a designated common vertex on γ. In this case, the condition $\mathsf{u}_i = \mathsf{u}_j$ is encoded as:

$$(\sum_{n \in \gamma} X(n)) \mid ((y_i - \sum_{n \in \pi_i} X(n)) - (y_j - \sum_{n \in \pi_j} X(n))) \ .$$

   For instance, in Figure 6.4 (b) we obtain $(x_3 + x_4) \mid (y_i - x_1) - (y_j - x_2)$. Observe, moreover, that, since γ is the only cycle in S, and the structure of S does not change during the program execution, all terms to the left of the divisibility sign are the same, for all guards.

The resulting counter machine will thus have transition rules labeled by formulae of the form: $\psi(\mathbf{x}, \mathbf{y}) \wedge \mathbf{x}' = \mathbf{x} \wedge \mathbf{y}' = A\mathbf{y} + \mathbf{b}$ where ψ is a formula of the $\mathsf{L}^1_{\mathrm{div}}$ fragment of integer arithmetic, $A \in \{0, 1\}^{r \times r}$ is an affine transformation

matrix whose set of powers is finite (A meets the finite monoid condition [Boi99]) and $\mathbf{b} \in \mathbb{Z}^r$ is an integer vector. Using acceleration, it can be proved that reachability and termination are decidable for this class of automata, by reduction to the validity problem for the $\mathsf{L}^1_{\text{div}}$ fragment (Chapter 3, Section 3.1), leading to the following result:

Theorem 20 ([BI07])**.** *The safety and termination problems for flat list programs without destructive updates, running on input heaps with at most one cycle are decidable.*

Proof. See [BI07, Corollary 3]. □

6.2 Programs with Trees

In this section we consider programs with tree-shaped data structures, in which a heap cell may have several outgoing edges (usually, we consider two edges, called left and right) and each node is reachable by exactly one path from a unique root node. A typical example of a tree structure widely used in practice are RED-BLACK trees, in which (i) every node is colored either red or black, (ii) the root and all leaves are black, (iii) if a node is red, both its children are black, and (iv) each path from the root to a leaf contains the same number of black nodes. An example is given in Figure 6.5 (a).

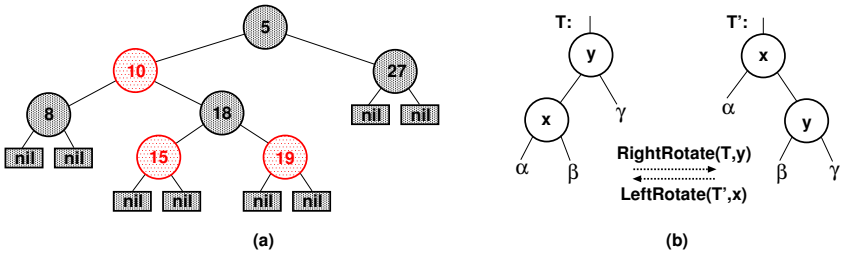


Figure 6.5: (a) Red-black tree, (b) left and right rotations

The main operations on balanced trees (and hence also RED-BLACK trees) are searching, insertion, and deletion. When implementing the last two operations, one has to make sure that the trees remain balanced. This is usually done using tree rotations, cf. Figure 6.5 (b), which can change the number of black nodes on a given path and requires re-balancing. If we consider rotations as a primitive program statement (instead of the low-level pointer manipulations used to implement rotations) every heap configuration can be represented by a tree, thus tree automata [CDG+05] are the natural way to encode infinite sets thereof.

Let us first introduce trees and tree automata formally. Let $\mathbb{N}^*$ be the set of sequences of positive integers. We denote by ε the empty sequence and by $p.q$ the concatenation of two sequences $p, q \in \mathbb{N}^*$. We say that q is a *prefix* of p, denoted $q \preceq p$ if $p = q.r$, for some sequence $r \in \mathbb{N}^*$. A *prefix-closed* set S has the property that, for all $p \in S$, $q \preceq p$ implies $q \in S$. A *ranked alphabet* is a countable set of symbols Σ with an associated arity function $\#(\sigma) \geq 0$, for all $\sigma \in \Sigma$. A *tree* is a finite partial function $t : \mathbb{N}^* \rightharpoonup_{fin} \Sigma$, whose domain, denoted $\mathrm{dom}(t)$, is a finite prefix-closed subset of $\mathbb{N}^*$. For each position $p \in \mathrm{dom}(t)$, a position $p.i \in \mathrm{dom}(t)$ is called a *child* of p, for some $i \in \mathbb{N}$. Let $\mathrm{Fr}(t) = \{p \in \mathrm{dom}(t) \mid \forall i \in \mathbb{N} \,.\, p.i \notin \mathrm{dom}(t)\}$ be the set of *leaves* (frontier) of the tree t. The *subtree* of t starting at position $p \in \mathrm{dom}(t)$ is the tree $t_{|p}$ defined as $t_{|p}(q) = t(p.q)$ if and only if $p.q \in \mathrm{dom}(t)$.

A (finite, nondeterministic, bottom-up) *tree automaton* (TA) [CDG+05] is a tuple $A = \langle Q, \Sigma, \Delta, F \rangle$, where Σ is a finite ranked alphabet, Q is a finite set of *states*, $F \subseteq Q$ is a set of *final states* and Δ is a set of *transition rules* of the form $\sigma(q_1, \ldots, q_n) \to q$, for some $\sigma \in \Sigma$ such that $\#(\sigma) = n$, and $q, q_1, \ldots, q_n \in Q$. The TA is *deterministic* if, for each $\sigma \in \Sigma$ such that $\#(\sigma) = n$, and each $q_1, \ldots, q_n \in Q$, there exists at most one rule in Δ with left hand side $\sigma(q_1, \ldots, q_n)$. A *run* of A over a tree $t : \mathbb{N}^* \rightharpoonup_{fin} \Sigma$ is a function $\pi : dom(t) \to Q$ such that, for each node $p \in dom(t)$, we have $q = \pi(p)$ only if $q_i = \pi(p.i)$ for all $i = 0, \ldots, \#(t(p)) - 1$, and there exists a rule $t(p)(q_0, \ldots, q_{\#(t(p))-1}) \to q \in \Delta$. We write $t \overset{\pi}{\Longrightarrow}_A q$ to denote that π is a run of A over t such that $\pi(\varepsilon) = q$, and omit π and A when they are clear from the context. The *language* of a state q of A is defined as $\mathcal{L}_q(A) = \{t \mid t \Longrightarrow_A q\}$, and the language of A is defined as $\mathcal{L}(A) = \bigcup_{q \in F} \mathcal{L}_q(A)$.

The main limitation of classical tree automata is that they cannot recognize sets of trees that are defined by arithmetic constraints involving the lengths of all their paths, such as the balancing constraint from the definition of RED-BLACK trees: all paths have the same number of black nodes. To cope with this limitation, we introduce a new class of tree automata, called *tree automata with size constraints* (TASC) [HIV10] and study their boolean closure properties and their decision problems. These tree automata are capable of recognizing balanced trees and can be, moreover, used to define the successor set of a set of TASC-recognizable trees. In this thesis we focus on the former aspects and refer the reader to [HIV10], for a definition of strongest post-conditions using TASC.

Definition 12. *A* size function *associates every tree* $t : \mathbb{N}^* \rightharpoonup_{fin} \Sigma$ *with an integer* $|t|$*, defined inductively on the structure of the tree. For each* $f \in \Sigma$*,*

if $\#(f) = 0$*, then* $|f|$ *is a constant, otherwise, for* $\#(f) = n > 0$*, we have:*

$$|f(t_1, \ldots, t_n)| = \begin{cases} b_1|t_1| + c_1 & \text{if} \vDash \delta_1(|t_1|, \ldots, |t_n|) \\ \cdots & \\ b_n|t_n| + c_n & \text{if} \vDash \delta_n(|t_1|, \ldots, |t_n|) \end{cases}$$

where $b_1, \ldots, b_n \in \{0, 1\}$*,* $c_1, \ldots, c_n \in \mathbb{Z}$*, and* $\delta_1(x_1, \ldots, x_n), \ldots, \delta_n(x_1, \ldots, x_n)$ *are difference bounds constraints defining a partition of* $\mathbb{N}^n$*, depending on* f*.*

A *sized alphabet* is a ranked alphabet equipped with a size function.

Example 15. *Let* $\Sigma = \{\mathsf{red}, \mathsf{black}, \mathsf{null}\}$ *be a ranked alphabet, with* $\#(\mathsf{red}) = \#(\mathsf{black}) = 2$ *and* $\#(\mathsf{null}) = 0$*. We define the size function to be the maximal number of black nodes from the root to a leaf:* $|\mathsf{null}| = 1$*,* $|\mathsf{red}(t_1, t_2)| = \max(|t_1|, |t_2|)$ *and* $|\mathsf{black}(t_1, t_2)| = \max(|t_1|, |t_2|) + 1$*.* ■

A *tree automaton with size constraints* (TASC) over a sized alphabet $(\Sigma, |.|)$ is a tuple $A = \langle Q, \Sigma, \Delta, F\rangle$ where Q is a finite set of states, $F \subseteq Q$ is a designated set of final states, and Δ is a set of transition rules of the form $f(q_1, \ldots, q_n) \xrightarrow{\varphi(|1|, \ldots, |n|)} q$ where $f \in \Sigma$, $\#(f) = n$, and $\varphi(x_1, \ldots, x_n)$ is difference bounds constraint. For constant symbols $a \in \Sigma$, $\#(a) = 0$, the automaton has unconstrained rules of the form $a \to q$.

A *run* of A over a tree $t : \mathbb{N}^* \rightharpoonup_{fin} \Sigma$ is a mapping $\pi : dom(t) \rightharpoonup_{fin} Q$, labeling each position of t by a state. Formally, for each position $p \in \mathrm{dom}(t)$, such that $q = \pi(p)$, we have:

- if $\#(t(p)) = n > 0$ and $q_i = \pi(p.i)$, for all $1 \leq i \leq n$, then Δ has a rule $t(p)(q_1, \ldots, q_n) \xrightarrow{\varphi(|1|, \ldots, |n|)} q$ and $\vDash \varphi(|t_{|p.1}|, \ldots, |t_{|p.n}|)$,
- otherwise, if $\#(t(p)) = 0$, then Δ has a rule $t(p) \to q$.

A run π is said to be *accepting* if $\pi(\varepsilon) \in F$. The *language* of A, denoted as $\mathcal{L}(A)$ is the set of all trees over which A has an accepting run.

Example 16. *(contd. from Example 15) A TASC recognizing the set of balanced red-black trees is* $A_{rb} = \langle \{q_b, q_r\}, \Sigma, \Delta, \{q_b\}\rangle$*, where:*

$$\Delta = \{\mathsf{null} \to q_b, \mathsf{black}(q_{b/r}, q_{b/r}) \xrightarrow{|1|=|2|} q_b, \mathsf{red}(q_b, q_b) \xrightarrow{|1|=|2|} q_r\} \ .$$

By writing $q_{x/y}$ *within the left-hand side of a transition rule, we mean the set of rules in which either* q_x *or* q_y *take the place of* $q_{x/y}$*.* ■

6.2.1 Closure Properties

We prove first that TASC are closed under determinization and the boolean operations of union, intersection and complement. The decidability of their emptiness problem is the subject of the next section.

A TASC is said to be *deterministic* (*complete*) if it has at most (at least) one run for every input tree. For a given TASC A, we define a deterministic and complete TASC A^d such that $\mathcal{L}(A^d) = \mathcal{L}(A)$, by a generalisation of the classical subset construction for tree automata [CDG+05]. Given $A = \langle Q, \Sigma, \Delta, F\rangle$, we define $A^d = \langle 2^Q, \Sigma, \Delta^d, \{S \subseteq Q \mid S \cap F \neq \varnothing\}\rangle$, where for each symbol $f \in \Sigma$ such that $\#(f) = n > 0$, we have $f(S_1, \ldots, S_n) \xrightarrow{\phi} S \in \Delta^d$ iff:

$$\begin{array}{rcl} S & \subseteq & \left\{q \mid f(q_1, \ldots, q_n) \xrightarrow{\psi} q \in \Delta, q_1 \in S_1, \ldots, q_n \in S_n\right\} \text{ and} \\ \psi & \equiv & \bigwedge\left\{\psi \mid f(q_1, \ldots, q_n) \xrightarrow{\psi} q \in \Delta, q_1 \in S_1, \ldots, q_n \in S_n, q \in S\right\} \wedge \\ & & \bigwedge\left\{\neg\psi \mid f(q_1, \ldots, q_n) \xrightarrow{\psi} q \in \Delta, q_1 \in S_1, \ldots, q_n \in S_n, q \notin S\right\} . \end{array}$$

Observe that the constraints labeling the rules of A^d are difference bounds constraints as well, because difference bounds constraints are closed under conjunctions and negation can be eliminated. For constant symbols $\#(f) = 0$, we have $f \to S \in \Delta^d$ iff $S = \{q \mid f \to q \in \Delta\}$.

Lemma 6. *For any given TASC A, A^d is deterministic and complete and, moreover, $\mathcal{L}(A^d) = \mathcal{L}(A)$.*

Proof. See [HIV10, Theorem 1]. □

The complement of a TASC $A = \langle Q, \Sigma, \Delta, F\rangle$ is $\overline{A} = \langle Q^d, \Sigma, \Delta^d, Q^d \setminus F^d\rangle$, where $A^d = \langle Q^d, \Sigma, \Delta^d, F^d\rangle$ is the deterministic and complete TASC obtained by applying the above construction to A. For two TASC $A_1 = \langle Q_1, \Sigma, \Delta_1, F_1\rangle$ and $A_2 = \langle Q_2, \Sigma, \Delta, F_2\rangle$, with disjoint sets of states, i.e. $Q_1 \cap Q_2 = \varnothing$, we define their union as $A_1 \cup A_2 = \langle Q_1 \cup Q_2, \Sigma, \Delta_1 \cup \Delta_2, F_1 \cup F_2\rangle$. Consequently, the intersection can be defined as $A_1 \cap A_2 = \overline{\overline{A_1} \cup \overline{A_2}}$.

6.2.2 Decision Problems

In this section we prove that the emptiness problem is decidable for TASC. Since TASC are closed under the boolean operations of complement, union and intersection, it follows that the universality and the inclusion problems are also decidable for TASC.

We show that all runs of a TASC are in a one-to-one correspondence to the accepting runs of an effectively constructed *Alternating Pushdown System* (APDS), whose emptiness problem is decidable [BEM97]. Namely,

it is shown that, given a regular set of configurations C, the set $\text{pre}^*_S(C)$ of predecessors w.r.t. the semantics of S is effectively regular, i.e. recognized by an alternating automaton that can be built in polynomial time in the size of S and exponential time in the size of the automaton recognizing C.

Given a TASC, we translate it into an APDS whose stack encodes the value of one integer counter, denoted by y from now on. An APDS is a tuple $S = \langle Q, \Gamma, \delta, F\rangle$ where Q is a finite set of control locations, Γ is a finite stack alphabet, $F \subseteq Q$ is a set of final control locations, and δ is a mapping from $Q \times \Gamma$ into $\mathcal{P}(\mathcal{P}(Q \times \Gamma^*))$. Notice that an APDS does not have an input alphabet since we are interested in the behaviors it generates, rather than in the accepted language. A run of an APDS is a tree $t : \mathbb{N}^* \rightharpoonup_{fin} (Q \times \Gamma^*)$, such that, for any $p \in \text{dom}(t)$, if $t(p) = \langle q, \gamma w\rangle$, then $\{t(p.i) \mid 1 \leq i \leq \#(t(p))\} = \{\langle q_1, w_1 w\rangle, \ldots, \langle q_n, w_n w\rangle\}$, where $\{\langle q_1, w_1\rangle, \ldots, \langle q_n, w_n\rangle\} \in \delta(q, \gamma)$. The run is accepting if all control locations occurring on its frontier are final.

For a TASC $A = \langle Q, \Sigma, \Delta, F\rangle$, let $S_A = \langle Q_A, \Gamma, \delta_A, F_A\rangle$ be the APDS where $Q_A = Q \times \Sigma \cup \Pi$, $\Gamma = \{-, 0, 1\}$, and $F_A = \{q_f\} \subset \Pi$. Here, Π is an additional set of states that are needed in the construction of S_A from A and that are not of the form $\langle q, f\rangle$. We use 0 as the bottom of the stack marker, $-$ on top of the stack denotes a negative value, and 1 is used for the unary encoding of the absolute value of the counter. We represent an integer value $n \in \mathbb{Z}$ using the unary encoding $(n)_1 = 1^n 0$ if $n \geq 0$ and $(n)_1 = -1^{-n} 0$ if $n < 0$. The primitive operations on the counter y, i.e. increment, decrement, and zero test, are encoded by the moves given below. For example, if the value of y in a control state q is -2, a transition that increments y and moves into q' is simulated by the sequence of moves: $\langle q, -110\rangle \rightsquigarrow \langle q^-, 110\rangle \rightsquigarrow \langle q'^-, 10\rangle \rightsquigarrow \langle q', -10\rangle$, where $(-2)_1 = -110$ and $(-1)_1 = -10$.

$q \xrightarrow{y'=y+1} q'$			$q \xrightarrow{y'=y-1} q'$			$q \xrightarrow{y=0} q'$		
$\langle q, 1\rangle$	$\hookrightarrow$	$\langle q', 11\rangle$						
$\langle q, 0\rangle$	$\hookrightarrow$	$\langle q', 10\rangle$						
$\langle q, -\rangle$	$\hookrightarrow$	$\langle q^-, \epsilon\rangle$	$\langle q, 1\rangle$	$\hookrightarrow$	$\langle q', \epsilon\rangle$			
$\langle q^-, 1\rangle$	$\hookrightarrow$	$\langle q'^-, \epsilon\rangle$	$\langle q, 0\rangle$	$\hookrightarrow$	$\langle q', -10\rangle$	$\langle q, 0\rangle$	$\hookrightarrow$	$\langle q', 0\rangle$
$\langle q'^-, 1\rangle$	$\hookrightarrow$	$\langle q', -1\rangle$	$\langle q, -\rangle$	$\hookrightarrow$	$\langle q', -1\rangle$			
$\langle q'^-, 0\rangle$	$\hookrightarrow$	$\langle q', 0\rangle$						

We shall encode a move of A as a series of moves of S_A. As A moves bottom-up on the tree, S_A will perform a series of alternating top-down transitions, simulating the move of A in reverse. The stack (counter) of S_A is intended to encode the value of the size function $|.|$ at the current tree node. Suppose that A has a transition rule $f(q_1, \ldots, q_n) \xrightarrow{\varphi} q$ and that the current node is of the form $f(t_1, \ldots, t_n)$ with $|f(t_1, \ldots, t_n)| = b_r|t_r| + c_r$, and δ_r is the disjunctive condition such that $\vDash \delta_r(|t_1|, \ldots, |t_n|)$ (Definition 12). W.l.o.g., we consider that φ and δ_r have the same set of free variables, denoted $x_1, \ldots, x_n$. In what follows, we consider the case $b_r = 1$, i.e., $|f(t_1, \ldots, t_n)| =$

$|t_r| + c_r$. The case $b_r = 0$ can be treated in a similar way, by guessing the value $|t_r|$. The position r is called the *reference position* of the subtree $f(t_1, \ldots, t_n)$. The value $|t_r|$ is called the *reference value* of $f(t_1, \ldots, t_n)$.

Without losing generality, we consider that the difference bounds constraint formula $\varphi \wedge \delta_r$ is written as a finite disjunction of difference bounds constraints of the form [HIV10, Lemma 4]:

$$\bigwedge_{k=1}^{n-1} x_{I(k)} - x_{I(k+1)} \diamond_k d_k \;\wedge \bigwedge_{m \in M \subseteq \{1,\ldots,n\}} x_m \leq e_m \;\wedge \bigwedge_{p \in P \subseteq \{1,\ldots,n\}} x_p \geq \ell_p$$

where $\diamond_k \in \{\leq, =\}$, $d_k, e_m, \ell_p \in \mathbb{Z}$ are constants, and I is a permutation of $\{1, \ldots, n\}$. For the rest of this section, let us fix one such disjunct.

After each sequence of universal moves, S_A creates n copies of its counter y, let us name them $y_1, \ldots, y_n$. Each counter y_i holds the value $|t_{I(i)}|$ for $1 \leq i \leq n$, and the counter y holds the value $|f(t_1, \ldots, t_n)|$. Let $i_r = I^{-1}(r)$ be the index of the counter y_{i_r} that holds the reference value of the given transition, i.e., $y = y_{i_r} + c_r$. With this notation, Figure 6.6 (a) shows the alternating moves of S_A that simulate the A-transition considered, for one disjunct of $\varphi \wedge \delta_r$. Figure 6.6 (b) shows the moves for transitions $a \to q$.

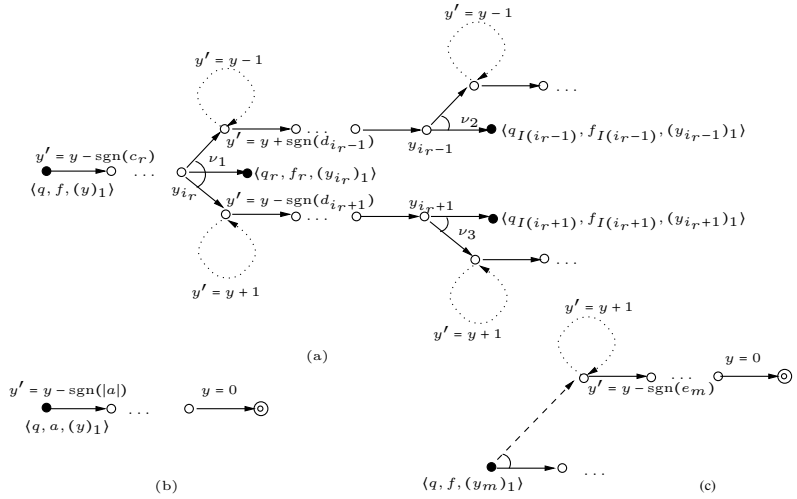


Figure 6.6: Simulation of a TASC by an APDS

Filled circles in Figure 6.6 represent states from $Q \times \Sigma$, and empty circles are additional states from Π. The only accepting state of S_A, named q_f, is marked by a double circle. The notation sgn(.) denotes the sign function, i.e. $\mathrm{sgn}(n) = 1$ if $n > 0$, $\mathrm{sgn}(0) = 0$, and $\mathrm{sgn}(n) = -1$ if $n < 0$. Next,

$\nu_1, \nu_2, \ldots$ are symbolic names for the universal moves performed by S_A. The configurations of S_A from $Q \times \Sigma \times \Gamma^*$ are labeled by triples of the form $\langle q, f, (y)_1 \rangle$, where $(y)_1$ denotes the unary encoding of the value of the y counter. Moreover, for simplicity, configurations from $\Pi \times \Gamma^*$ are label-led only with $(y)_1$ in Figure 6.6.

When simulating the A-transition $f(q_1, \ldots, q_n) \xrightarrow{\varphi} q$, S_A starts with the configuration $\langle q, f, (y)_1 \rangle$ (cf. Figure 6.6 (a)). In order to derive the reference value y_{i_r} from y, S_A performs $\mathrm{abs}(c_r)$ decrement or increment actions, depending on whether the sign of c_r is positive or negative. Then S_A performs the universal move ν_1 making three copies of itself (unless $i_r = 1$ when the upper branch is omitted and/or $i_r = n$ when the lower branch is omitted). The middle branch simply moves to the appropriate control state $\langle q_r, f_r \rangle$ with stack $(y_{i_r})_1$. The upper and lower branches are used to produce the values y_{i_r-1} and y_{i_r+1}, if needed.

The upper branch of the universal move ν_1 depicted in Figure 6.6 depends on $\diamond_r \in \{\leq, =\}$. If $\diamond_r$ is $=$, then S_A performs a sequence of increment/decrement operations of length d_{i_r-1} in order to obtain the value y_{i_r-1} from y_{i_r} (since $y_{i_r-1} = y_{i_r} + d_{i_r-1}$). If $\diamond_r$ is $\leq$, then there is an additional existential (nondeterministic) transition—depicted using a dotted arrow in Figure 6.6 (a)—which decrements the counter an arbitrary number of times in order to obtain a smaller value (since $y_{i_r-1} \leq y_{i_r} + d_{i_r-1}$).

In order to simulate moves of the form $a \to q$ (Figure 6.6 (b)), S_A simply decrements/increments the counter, depending on the sign of $|a|$, a number of times equal to the absolute value of $|a|$. The condition $y = 0$ ensures that S_A accepts only with the empty stack. The universal dotted branch in Figure 6.6 (c) is used to test that $y_m \leq e_m$ for some $1 \leq m \leq n$. A similar test for $y_p \geq l_p$ can be issued by replacing $y' = y + 1$ with $y' = y - 1$ on the cycle. The following theorem states the result of this section:

Theorem 21. *The emptiness, universality and inclusion problems are decidable for TASC.*

Proof. For the emptiness problem, see [HIV10, Theorem 2]. The other decidability results are consequences of the fact that TASC are closed under complement and intersection. □

The decidability of the emptiness problem for TASC can also be proved via a reduction to the class of *tree automata with one memory* (TAOM) [CC05] by encoding the size of a tree as a unary term. The inequality constraints from the guards of the TASC can be simulated analogously by adding increment/decrement self loops to the tree automata with one memory. Both the emptiness problems for APDS and TAOM are shown to belong to DEXPTIME. It is currently an open problem whether the same complexity bound applies to TASC.

6.3 Logics of Integer Arrays

In this section we shift the focus from programs with dynamic memory and recursively defined data structures and concentrate on programs with statically declared arrays. Arrays are important, mostly for safety-critical embedded control software, for which the standards forbid the use of pointers and dynamic memory allocation, thus all program data must be contained within static arrays. Another application are *parametric systems* [APR+01, BJS07] consisting of N replicated processes running concurrently. Here one uses arrays of size N to represent the local data of each process. The properties typically expressed about arrays in a program involve universally quantified index variables, such as, e.g.:

- the array a is sorted: $\forall i \ . \ a[i+1] \geq a[i]$,
- all elements of a are bigger than all elements of b: $\forall i \forall j \ . \ a[i] \geq b[j]$,
- all elements of a on even positions are positive: $\forall i \ . \ i \equiv_2 0 \rightarrow a[i] > 0$.

In general, we consider boolean combinations of formulae of the form $\forall \mathbf{i} \ . \ \gamma(\mathbf{i}) \rightarrow \upsilon(\mathbf{a}, \mathbf{i})$, where $\gamma(\mathbf{i})$ is a *guard* defining a constraint over the universally quantified index variables $\mathbf{i}$ and $\upsilon(\mathbf{a}, \mathbf{i})$ is a *value expression* that defines a constraint over array read terms $a[i+n]$, where n is an integer constant.

Without specific syntactic restrictions, a logic of such expressive power is shown to be undecidable, by encoding the history of computations of a 2-counter machine [Min67] as models of an array formula. From this reduction, one can derive two restrictions leading to decidability. The first restriction, considered by Bradley, Manna, and Sipma [BMS06], forbids using $a[i]$ and $a[i+1]$ in the same value expression. The second restriction, considered in the following, allows only conjunctive array expressions, allowing to reason about consecutive arrays elements $a[i]$ and $a[i+1]$, which appears to be important in program verification.

We define two novel logics, called *Singly-Indexed Logic* (SIL) [HIV08a] and *Logic of Integer Arrays* (LIA) [HIV08b], for which we proved the decidability of their satisfiability problems by reduction to the reachability problem for flat counter machines with octagonal constraints (Chapter 4). Essentially, in SIL we consider array properties $\forall i \ . \ \gamma(i) \rightarrow \upsilon(\mathbf{a}, i)$ with only one universally quantified index variable, whereas in LIA we lift this restriction and consider any number of universally quantified index variables.

The advantage of the single index restriction (SIL) is that the flat counter machines produced are *deterministic*, thus the translation can be defined compositionally, by induction on the boolean structure of the formula. If we lift this restriction (LIA), eliminating negations of array properties requires an expensive normalization step, that avoids the complementation of the produced counter machines.

6.3.1 A Singly-Indexed Logic

We consider three disjoint sets of variables, namely *bound* variables BVar, *index* variables IVar and *array* variables AVar. The bound variables are used to define intervals in which some universal property is required to hold. The index variables are the only variables that may occur under the scope of a universal quantifier, as in $\exists \ell \exists u \;.\; \ell < u \wedge \forall i \;.\; \ell \leq i \leq u \rightarrow a[i] = 0$, where $\ell, u \in \mathsf{BVar}$, $i \in \mathsf{IVar}$ and $a \in \mathsf{AVar}$. The abstract syntax of the logic SIL is given below:

i	$\in$	IVar	index variables
a, b	$\in$	AVar	array variables
$\sim$	$\in$	$\{\leq, \geq\}$	
B	$:=$	$n \in \mathbb{Z} \mid \ell \in \mathsf{BVar} \mid \lvert a \rvert \mid B \pm B$	bound terms
A	$:=$	$a[i + d]$	array reads ($d \in \{0, 1\}$)
G	$:=$	$\top \mid i \sim B \mid i \equiv_s t \mid G \wedge G \mid G \vee G$	guard expressions ($0 \leq t < s$)
V	$:=$	$A \sim B \mid A \pm A \sim n \mid i \pm A \sim n \mid V \wedge V$	value expressions
C	$:=$	$B \sim n \mid B \equiv_s t$	bound constraints ($0 \leq t < s$)
P	$:=$	$\forall i \;.\; G \rightarrow V$	array properties
F	$:=$	$P \mid C \mid \neg F \mid F \;\wedge\; F \mid F \;\vee\; F$	formulae

Observe that, a value expression is a comparison between (i) array read terms $a[i + d]$ (within a window $d \in \{0, 1\}$) and bound terms, (ii) two array read terms, or (iii) an array read term and an index variable. Allowing comparisons between more than two terms and/or disjunctions within value expressions would lead to undecidability, by allowing to encode histories of 2-counter machines within the arrays [Min67].

We use $|a|$ for the length of the array a and define, for a value expression υ, the sanity condition $\mathcal{B}(\upsilon)$, ensuring that no out-of-bounds reads occur:

$$\mathcal{B}(a[i + d] \sim B) \equiv 0 \leq i + d < |a| \qquad \mathcal{B}(i \pm a[i + d] \sim n) \equiv 0 \leq i + d < |a|$$
$$\mathcal{B}(a[i + d] \pm b[i + e] \sim n) \equiv 0 \leq i + d < |a| \wedge 0 \leq i + e < |b|$$
$$\mathcal{B}(\upsilon_1 \wedge \upsilon_2) \equiv \mathcal{B}(\upsilon_1) \wedge \mathcal{B}(\upsilon_2)$$

The semantics of the SIL logic is given in terms of a forcing relation $\iota, \mu \vDash \phi$, where $\iota : \mathsf{BVar} \cup \mathsf{IVar} \rightharpoonup \mathbb{Z}$ is a partial interpretation of bound and index variables and $\mu : \mathsf{AVar} \rightarrow \mathbb{Z}^*$ associates each array variable a finite sequence of integers. Then the interpretation $[\![t]\!]_{\iota,\mu}$ of a term t is defined inductively, where, in particular, we have $[\![|a|]\!]_{\iota,\mu} = |\mu(a)|$ is the length and $[\![a[i+d]]\!]_{\iota,\mu} = \mu(a)_{\iota(i)+d}$ is the $(\iota(i) + d)$-th element of the sequence $\mu(a)$. The semantics

of an array property is defined as:

$$\iota, \mu \models \forall i \;.\; \gamma \rightarrow \upsilon \text{ iff for all } n \in \mathbb{Z}\text{: } \iota[i \leftarrow n], \mu \models \gamma \wedge \mathcal{B}(\upsilon) \Rightarrow \iota[i \leftarrow n], \mu \models \upsilon \;\; .$$

A pair (ι, μ) such that $\iota, \mu \models \varphi$ is a *model* of φ. The *satisfiability problem* asks, given φ, for the existence of a model.

We address the satisfiability problem for the logic SIL using a compositional translation of formulae into counter machines. In fact, we show that the set of array valuations $\mu : \mathsf{AVar} \rightarrow \mathbb{Z}^*$ that are models of a SIL formula corresponds to the accepting runs of a deterministic flat counter machine, with octagonal constraints, defined by induction on the structure of the formula. Since the reachability problem for these counter machines is decidable (as proved in Chapter 4), we can infer that the satisfiability problem of the SIL logic is also decidable.

A counter machine $M = \langle \mathbf{x}, Q, \iota, F, \Delta \rangle$ is *deterministic* if for every two transition rules $q \xrightarrow{\phi_1} q_1$ and $q \xrightarrow{\phi_2} q_2$, we have $\phi_1 \wedge \phi_2 \rightarrow \bot$. The *trace language* $\mathcal{L}(M)$ of a counter machine is the set of all finite sequences of counter valuations $\nu_1, \ldots, \nu_k \in (\mathbb{Z}^k)^*$ for which there exists a sequence of transition rules $\iota \xrightarrow{\phi_1} q_1 \xrightarrow{\phi_2} \ldots \xrightarrow{\phi_k} q_{k+1}$ such that $q_{k+1} \in F$ and $(\nu_i, \nu_{i+1}) \models \phi_i$, for all $1 \leq i \leq k$. It is easy to prove that a deterministic counter machine has exactly one run for a given finite sequence of counter valuations $\nu_1, \nu_2, \ldots, \nu_k \in \mathcal{L}(M)$. Moreover, the class of deterministic flat counter machines with octagonal relations is closed under the boolean operations of union, intersection and complement, with respect to their trace languages [HIV08a, Lemmas 1, 2].

Let $\phi(\mathbf{k}, \mathbf{a})$ be a SIL formula with free variables $\mathbf{k} \subseteq \mathsf{BVar}$ and $\mathbf{a} \subseteq \mathsf{AVar}$. The counter machine M_ϕ will have the set $\mathbf{x}$ of counters:

- for each $k \in \mathbf{k}$ and each $a \in \mathbf{a}$, we have parameters x_k and $x_{|a|}$, whose values remain unchanged during the execution of M_ϕ,
- for each $a \in \mathbf{a}$, we have a counter x_a whose value, at step i during the execution of M_ϕ is $a[i]$,
- τ is a special counter, initially 0, incremented by each transition of M_ϕ, indicating the current position in the arrays.

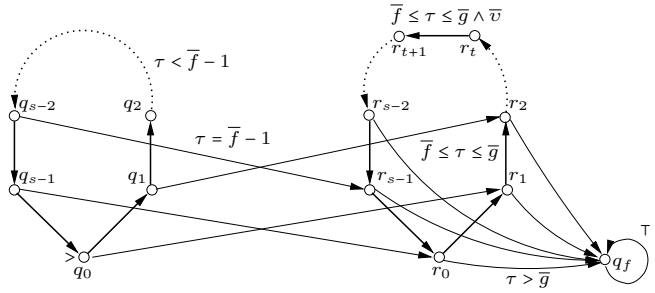
M_ϕ is the intersection $M_\tau \cap \mathcal{M}_\phi$, where $M_\tau = \langle \mathbf{x}, \{q_0, q_\tau\}, \{q_0\}, \{q_\tau\}, \rightarrow_\tau \rangle$ and:

$$q_0 \xrightarrow{\tau=0 \wedge \tau'=\tau+1 \wedge \bigwedge_{k \in \mathbf{k}} x'_k = x_k \wedge \bigwedge_{a \in \mathbf{a}} x'_{|a|} = x_{|a|}} q_\tau$$

$$q_\tau \xrightarrow{\tau'=\tau+1 \wedge \bigwedge_{k \in \mathbf{k}} x'_k = x_k \wedge \bigwedge_{a \in \mathbf{a}} x'_{|a|} = x_{|a|}} q_\tau$$

are the only transition rules of M_τ. The definition of $\mathcal{M}_\phi$ is by induction on the structure of ϕ:

- $\mathcal{M}_{\neg\phi} = \overline{\mathcal{M}_\phi}$, where $\mathcal{L}(\overline{\mathcal{M}_\phi}) = (\mathbb{Z}^{\mathbf{x}})^* \setminus \mathcal{L}(\mathcal{M}_\phi)$,
- $\mathcal{M}_{\phi_1 \wedge \phi_2} = \mathcal{M}_{\phi_1} \cap \mathcal{M}_{\phi_2}$, where $\mathcal{L}(\mathcal{M}_{\phi_1} \cap \mathcal{M}_{\phi_2}) = \mathcal{L}(\mathcal{M}_{\phi_1}) \cap \mathcal{L}(\mathcal{M}_{\phi_2})$,
- $\mathcal{M}_{\phi_1 \vee \phi_2} = \mathcal{M}_{\phi_1} \cup \mathcal{M}_{\phi_2}$, where $\mathcal{L}(\mathcal{M}_{\phi_1} \cup \mathcal{M}_{\phi_2}) = \mathcal{L}(\mathcal{M}_{\phi_1}) \cup \mathcal{L}(\mathcal{M}_{\phi_2})$,
- if ϕ is a bound constraint (generated by the abstract syntax with axiom C) then $\mathcal{M}_\phi = \langle \mathbf{x}, Q, q_0, \{q_1\}, \rightarrow \rangle$, with transition rules $q_0 \xrightarrow{\overline{\phi}} q_1$ and $q_1 \xrightarrow{\top} q_1$, where $\overline{\phi}$ is obtained from ϕ, by replacing all occurrences of $k \in \mathbf{k}$ by x_k and all occurrences of $|a|, a \in \mathbf{a}$ by $x_{|a|}$,
- if ϕ is the array property $\forall i \,.\, f \leq i \leq g \wedge i \equiv_s t \rightarrow v$, M_ϕ is shown below:



The terms $\overline{f}$, $\overline{g}$ and $\overline{v}$ are obtained from f, g and v by replacing all occurrences of $k \in \mathbf{k}$ by x_k, all occurrences of $|a|, a \in \mathbf{a}$ by $x_{|a|}$, each occurrence of i not inside an array read by τ and each occurrence of array reads $a[i]$ and $a[i+1]$ by x_a and x'_a, for all $a \in \mathbf{a}$, respectively. The control structure of M_ϕ keeps track of the current value of τ modulo s. Observe that the counter machine is flat and deterministic, due to the treatment of τ within the guards of the transitions. In the first cycle $q_0 \rightarrow q_1 \rightarrow \ldots \rightarrow q_{s-1} \rightarrow q_0$ the machine waits until the variable τ reaches the lower bound $\overline{f} - 1$. Then the control is transferred to the second cycle $r_0 \rightarrow r_1 \rightarrow \ldots \rightarrow r_{s-1} \rightarrow r_0$ that checks the value expression $\overline{v}$ when $\tau \equiv_s t$, on the transition $r_t \rightarrow r_{t+1}$. Finally, this cycle is left towards the final state q_f, when $\tau > \overline{g}$. The following theorem states the main result of this section:

Theorem 22. *The satisfiability problem is decidable for the logic* SIL.

Proof. See [HIV08a, Theorem 2]. □

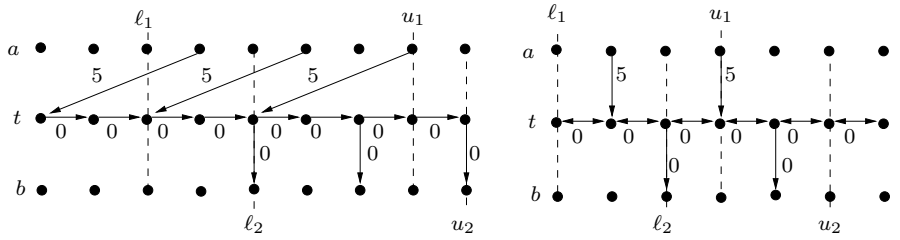
6.3.2 A Logic of Integer Arrays

In the following, we lift the single-index restriction and consider the *logic of integer arrays* (LIA), which is the set of quantifier-free boolean combinations of array properties of the following form:

$$\forall i \forall j \,.\, \ell_1 \leq i \leq u_1 \wedge \ell_2 \leq j \leq u_2 \wedge i \equiv_{s_1} t_1 \wedge j \equiv_{s_2} t_2 \wedge m \leq i - j \leq n \rightarrow a[i] \pm b[j] \sim p$$

where $\ell_1, u_1, \ell_2, u_2 \in \mathbb{Z}_\infty$ are the lower and upper bounds for i and j, respectively and $m, n \in \mathbb{Z}_\infty$ are the lower and upper bounds on the difference between i and j. Observe that these array properties relate elements of a and b situated at a distance which can be arbitrarily large.

Because of this issue, such array properties cannot be directly encoded as counter machines checking the validity of a value expression at certain moments during its computation. This limitation can be overcome by introducing additional array variables, whose adjacent elements are related by transitive inequality ($\leq$), that capture the non-local constraints $a[i]-b[j] \leq p$ by the transitive closure of a finite set of local constraints. For instance, for the array property $\forall i \forall j \ . \ i \leq j \rightarrow a[i] - b[j] \leq 5$, the non-local constraint $a[0] - b[3] \leq 5$ is encoded as $a[0] - t[0] \leq 5 \wedge t[0] - t[1] \leq 0 \wedge t[1] - t[2] \leq 0 \wedge t[2] - t[3] \leq 0 \wedge t[3] - b[3] \leq 0$, where t is an additional array variable. This step is best understood by looking at the constraint graphs defined by the array property formulae, as in the example below:



(a) $\forall i \forall j \ . \ \ell_1 \leq i \leq u_1 \wedge \ell_2 \leq j \leq u_2 \wedge i - j \leq 3 \wedge i \equiv_2 0 \wedge j \equiv_2 1 \rightarrow a[i] - b[j] \leq 5$

(b) $\forall i \forall j \ . \ \ell_1 \leq i \leq u_1 \wedge \ell_2 \leq j \leq u_2 \wedge i \equiv_2 0 \wedge j \equiv_2 1 \rightarrow a[i] - b[j] \leq 5$

Figure 6.7: Constraint graphs for array properties

However, introducing new array variables is not possible for the array properties occurring under an odd number of negations, because this would mean introducing universally quantified array variables in the logic, leading to undecidability. To work around the negation problem, we normalize any quantifier-free boolean combination of array properties to formulae of the form: $\phi \equiv \bigvee_c \bigwedge_d \phi_{cd}(\mathbf{a}, \mathbf{k}) \wedge \theta_c(\mathbf{k})$, where $\mathbf{a}$ is a set of array variables, $\mathbf{k}$ is a set of integer variables, and

- each θ_c is a conjunction of terms of the forms (i) $g(\mathbf{k}) \geq 0$ or (ii) $g(\mathbf{k}) \equiv_s t$ where $g(\mathbf{k})$ is a linear term and $0 \leq t < s$ are constants,
- each ϕ_{cd} is of one of the following forms, for $\sim \in \{\leq, \geq\}$, linear terms f_k, g_ℓ, f_k^1, g_ℓ^1, f_k^2, g_ℓ^2 over $\mathbf{k}$ and constants $m \in \mathbb{N}$, $p, q \in \mathbb{Z}_\infty$, $0 \leq t < s$, $0 \leq v < u$:

$$\forall i \ . \ \bigwedge_{k=1}^{K} f_k \leq i \ \wedge \ \bigwedge_{l=1}^{L} i \leq g_l \ \wedge \ i \equiv_s t \rightarrow a[i] \sim h(\mathbf{k}) \qquad \text{(F1)}$$

(F1) formulae bind all values of a in some interval by some linear combination h of variables in $\mathbf{k}$.

$$\forall i \;.\; \bigwedge_{k=1}^{K} f_k \leq i \;\wedge\; \bigwedge_{l=1}^{L} i \leq g_l \;\wedge\; i \equiv_s t \rightarrow a[i] - b[i+p] \sim q \tag{F2}$$

(F2) formulae relate all values of a and b in the same interval such that the distance between the indices of a and b, respectively, is constant.

$$\begin{array}{c} \forall i, j \;.\; \bigwedge_{k=1}^{K_1} f_k^1 \leq i \;\wedge\; \bigwedge_{l=1}^{L_1} i \leq g_l^1 \;\wedge\; \bigwedge_{k=1}^{K_2} f_k^2 \leq j \;\wedge\; \bigwedge_{l=1}^{L_2} j \leq g_l^2 \;\wedge \\ i - j \leq p \;\wedge\; i \equiv_s t \;\wedge\; j \equiv_u v \rightarrow a[i] - b[j] \sim q \end{array} \tag{F3}$$

(F3) formulae relate all values of a with all values of b within two intervals. If $\phi \equiv \bigvee_c \bigwedge_d \phi_{cd}(\mathbf{a}, \mathbf{k}) \wedge \theta_c(\mathbf{k})$ is in normal form, the counter machine whose trace language is the set of array valuations of ϕ is defined as $\mathcal{M}_\phi = M_\tau \cap (\bigcup_c \bigcap_d \mathcal{M}_{\phi_{cd}} \cap \mathcal{M}_{\theta_c})$, where M_τ is the counter machine that increments the step variable τ and copies the values of the parameters $\{x_k \mid k \in \mathbf{k}\}$ (Section 6.3.1), $\mathcal{M}_{\theta_c}$ consists of a single transition that checks the condition θ_c on the parameters and $\mathcal{M}_{\phi_{cd}}$ is defined according to the type (F1)-(F3) of the array property. The most interesting case is (F3), explained next.

The definition of the counter machines is simplified, by observing that the constraint graphs that define the models of the formulae in the logic, after elimination of the non-local constraints by introducing new array variables, consists of edges that span a fixed size window and are either vertical, horizontal or diagonal. The counter machine for an array property of type (F1)-(F3) is the intersection of several counter machines of a fixed structure.

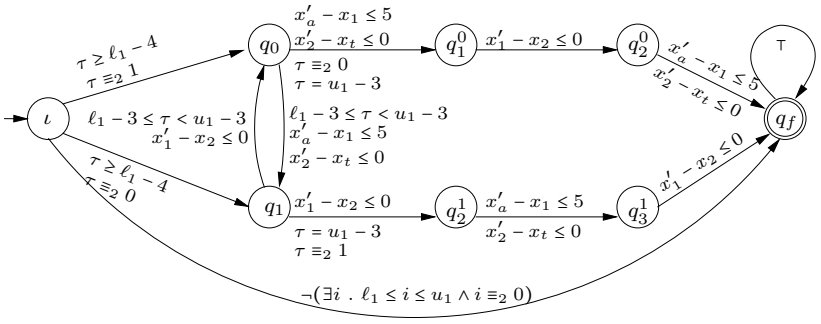


Figure 6.8: Counter machine for diagonal edges in Fig. 6.7 (a)

For instance, the counter machine in Figure 6.8 defines the diagonal edges of the constraint graph in Figure 6.7 (a). Observe that the constraints $a[i] - t[i-3] \leq 5$, that occur for $i \equiv_2 0$, span a window of size 3. We represent

these constraints in the counter machine by introducing additional variables x_1 and x_2 and firing the transitions $q_0 \xrightarrow{x'_a - x_1 \leq 5} q_1 \xrightarrow{x'_1 - x_2 \leq 0} q_0 \xrightarrow{x'_2 - x_t \leq 0} 1$ instead, if $\tau \equiv_2 0$ (a similar sequence is possible for the case $\tau \equiv_2 1$).

The counter machine for each array property is flat and flatness is preserved by union and intersection of counter machines. We have thus reduced the satisfiability problem for a LIA formula to the reachability of a flat counter machine with octagonal constraints.

Theorem 23. *The satisfiability problem is decidable for the logic* LIA.

Proof. See [HIV08b, Corollary 1]. □

6.4 Discussion and Open Problems

The Turing-completeness result for 2-counter machines [Min67] proves, in principle, that every program or computer system can be analyzed using a reduction to counter machines with only increment, decrement and zero test. However, the effective definitions of these reductions, even for specific classes of programs, requires hard work. On the other hand, one can build up on existing complexity results for various decidable classes of counter machines to derive decidability and complexity bounds for certain program verification problems, which can, over time, improve the state of the art of existing program verifiers. Counter machines provide also effective ways of deciding satisfiability within logics over infinite data domains, e.g. array logics (Section 6.3).

An interesting research direction consists in applying other known decidability and complexity results for counter machines, such as reversal-bounded or (branching) vector addition systems to design more expressive array logics and study their computational complexities. For instance, the logics to automata translations from Section 6.3.1 and 6.3.2 build flat counter machines with octagonal constraints of size which is likely to be simply exponential in the size of the input formulae[1]. Given the result of Theorem 10, the satisfiability problems of SIL and LIA could potentially belong to NEXPTIME. Finding a matching lower bound is currently an open problem. Another problem is defining an array logic matching the expressive power of the recursive counter machine model, with the restriction that the set of interprocedurally valid paths belong to a bounded language (Section 5.3).

[1] The intersection operation used to define conjunctions of array properties already causes an exponential blowup, even in the absence of negations.

Chapter 7

Separation Logic

Separation Logic (initially called BI, for *logic of bunched implications* [OP99]) is a logical framework centered on the notion of *resource*. Generally speaking, resources are entities that can be distributed among certain populations. They can be also split into disjoint parts and combined as a whole. From an abstract perspective, resources form a monoid with their join operation, e.g. the set of words over a given alphabet with word concatenation as join.

In program verification, Separation Logic (SL) [IO01, Rey02] is mainly used to describe dynamically allocated recursive data structures and develop modular verification techniques, based on the principle of *local reasoning*: analyzing different parts (functions, threads) of the program that work on separate sections of the heap and combine the analysis results a-posteriori. Using Separation Logic as a specification formalism is attractive because of the possibility of writing higher-order inductive definitions that naturally describe recursive data structures, such as singly- or doubly-linked lists, skip lists, trees and more complex variations, such as nested and overlaid data structures, e.g. trees with linked leaves, hash maps, etc.

However, the expressive power of SL comes with a high price: the satisfiability problem is undecidable, even without inductive definitions. On the other hand, the satisfiability problem for the quantifier-free fragment of SL without inductive definitions is shown to be PSPACE-complete [CYO01].

In this chapter we define a fragment of SL with general inductive definitions, for which both satisfiability of an assertion (predicate) and the validity of entailments between assertions (predicates) are decidable, with elementary complexity. This result relies on an embedding into the Monadic Second Order Logic (MSO) of graphs with bounded treewidth [Cou90]. By applying a further restriction, we define a fragment in which the entailment

problem is EXPTIME-complete, by reduction to the inclusion problem for tree automata.

We consider a set of variables Var. The syntax of SL, without inductive definitions, is given below:

$$\begin{array}{lcl} \mathsf{u}, \mathsf{v} & \in & \mathsf{Var} \\ \mathsf{P} & \in & \mathsf{Pred} \\ \phi & := & \mathsf{u} = \mathsf{v} \mid \mathsf{u} = \mathsf{nil} \mid \mathsf{u} \mapsto (\mathsf{v}_1, \ldots, \mathsf{v}_k) \mid \mathsf{emp} \\ & & \phi_1 * \phi_2 \mid \phi_1 \mathrel{-\!\!*} \phi_2 \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \mid \exists \mathsf{u} \ . \ \phi_1 \end{array}$$

We recall the definition of states (Definition 9, Chapter 6), where the set of selectors is $\mathsf{Sel} = \{1, \ldots, k\}$ and Loc denotes an infinite countable set of memory locations. The semantics of SL is given by a forcing relation between states $\langle s, h \rangle$ and formulae, as follows:

$$\begin{array}{lcl} \langle s, h \rangle \models_{\mathsf{SL}} \mathsf{u} = \mathsf{v} & \Leftrightarrow & s(\mathsf{u}) = s(\mathsf{v}) \\ \langle s, h \rangle \models_{\mathsf{SL}} \mathsf{u} = \mathsf{nil} & \Leftrightarrow & s(\mathsf{u}) = \mathsf{null} \\ \langle s, h \rangle \models_{\mathsf{SL}} \mathsf{u} \mapsto (\mathsf{v}_1, \ldots, \mathsf{v}_k) & \Leftrightarrow & s(\mathsf{u}) = \ell_0, s(\mathsf{v}_1) = \ell_1, \ldots s(\mathsf{v}_k) = \ell_k \text{ and} \\ & & h = \{(\ell_0, \lambda i \ . \text{ if } 1 \leq i \leq k \text{ then } \ell_i \text{ else } \bot)\}, \\ & & \text{for some } \ell_0, \ell_1, \ldots, \ell_n \in \mathsf{Loc} \\ \langle s, h \rangle \models_{\mathsf{SL}} \mathsf{emp} & \Leftrightarrow & h = \varnothing \\ \langle s, h \rangle \models_{\mathsf{SL}} \phi * \varphi & \Leftrightarrow & \langle s, h_1 \rangle \models_{\mathsf{SL}} \phi \text{ and } \langle s, h_2 \rangle \models_{\mathsf{SL}} \varphi, \text{ for some } h_1, h_2 \\ & & \text{such that } \mathrm{dom}(h_1) \cap \mathrm{dom}(h_2) = \varnothing \text{ and } h_1 \cup h_2 = h \\ \langle s, h \rangle \models_{\mathsf{SL}} \phi \mathrel{-\!\!*} \varphi & \Leftrightarrow & \text{for all } h', \mathrm{dom}(h') \cap \mathrm{dom}(h) = \varnothing \text{ and } \langle s, h' \rangle \models_{\mathsf{SL}} \phi \\ & & \text{implies } \langle s, h' \cup h \rangle \models_{\mathsf{SL}} \varphi \\ \langle s, h \rangle \models_{\mathsf{SL}} \exists \mathsf{u} \ . \ \phi & \Leftrightarrow & \langle s[\mathsf{u} \leftarrow \ell], h \rangle \models_{\mathsf{SL}} \varphi, \text{ for some } \ell \in \mathsf{Loc} \end{array}$$

The semantics of the boolean connectives $\wedge$ and $\neg$ is as in classical first-order logic. Observe that the *points-to* proposition $\mathsf{u} \mapsto (\mathsf{v}_1, \ldots, \mathsf{v}_k)$ holds in a state whose heap has exactly one location in the domain. Consequently, $\mathsf{u}_1 \mapsto \mathsf{v}_1 \wedge \mathsf{u}_2 \mapsto \mathsf{v}_2$ holds only if $\mathsf{u}_1 = \mathsf{u}_2$ and $\mathsf{v}_1 = \mathsf{v}_2$. On the other hand, the separating conjunction $*$ asks that the heap can be split into two disjoint heaps, each satisfying one of its subformulae. For instance, $\mathsf{u}_1 \mapsto \mathsf{v}_1 * \mathsf{u}_2 \mapsto \mathsf{v}_2$ holds only for states $\langle s, h \rangle$ such that $\|\mathrm{dom}(h)\| = 2$.

Given a set Pred of predicate names, with arities denoted as $\#(\mathsf{P})$, for each $\mathsf{P} \in \mathsf{Pred}$, an *inductive system* is a set of definitions:

$$\left\{ \mathsf{P}_i(\mathsf{x}_{i,1}, \ldots, \mathsf{x}_{i,\#(\mathsf{P}_i)}) \equiv \bigvee_{i=1}^{m_i} \mathsf{r}_{ij}(\mathsf{x}_{i,1}, \ldots, \mathsf{x}_{i,\#(\mathsf{P}_i)}) \right\}_{i=1}^{n}$$

where $\mathsf{x}_{i,1}, \ldots, \mathsf{x}_{i,\#(\mathsf{P}_i)} \in \mathsf{Var}$ are called *parameters*, and $\mathsf{r}_{ij}(\mathsf{x}_{i,1}, \ldots, \mathsf{x}_{i,\#(\mathsf{P}_i)})$ are open SL formulae, called *rules*, which may contain atomic propositions

of the form $\mathsf{P}(\mathsf{u}_1, \ldots, \mathsf{u}_{\#(\mathsf{P})})$, for some $\mathsf{P} \in \mathsf{Pred}$ and variables $\mathsf{u}_1, \ldots, \mathsf{u}_{\#(\mathsf{P})} \in \mathsf{Var}$. We extend the forcing relation to predicates, as follows:

$$\langle s, h \rangle \vDash_{\mathsf{SL}} \mathsf{P}_i \Leftrightarrow \langle s, h \rangle \vDash_{\mathsf{SL}} \exists \mathsf{x}_{i,1} \ldots \exists \mathsf{x}_{i,\#(\mathsf{P}_i)} \cdot \mathsf{r}_{ij}(\mathsf{x}_{i,1}, \ldots, \mathsf{x}_{i,\#(\mathsf{P}_i)}), \text{ for some } 1 \leq j \leq m_i$$

For a formula ϕ, possibly containing predicates, we define the set of models $[\![\phi]\!] = \{\langle s, h \rangle \mid \langle s, h \rangle \vDash_{\mathsf{SL}} \phi\}$. This set is the least fixed point of a (provably) monotonic and continuous function mapping sets of states into sets of states.

The *satisfiability problem* asks, given a predicate P, whether $[\![\mathsf{P}]\!] \neq \emptyset$. The *entailment problem* $\mathsf{P}_i \vDash_{\mathsf{SL}} \mathsf{P}_j$ asks whether $[\![\mathsf{P}_i]\!] \subseteq [\![\mathsf{P}_j]\!]$. Both problems are undecidable even when the rules of the system do not contain predicates or the separation logic connectives $*$ and $-\!\!*$. In fact, only first-order quantification and binary points-to propositions $\mathsf{x} \mapsto (\mathsf{y}, \mathsf{z})$ are sufficient to encode the computation of a Turing machine as a satisfiability problem [CYO01]. Next, we define a decidable fragment of SL with inductive definitions, that allows to reason about rather general recursive data structures.

7.1 A Decidable Inductive Fragment of SL

To begin with, we restrict the rules of the inductive system to formulae of the form $\mathsf{r}(\mathsf{x}_1, \ldots, \mathsf{x}_p) \equiv \exists \mathsf{y}_1 \ldots \exists \mathsf{y}_q \cdot \Sigma \wedge \Pi$ where:

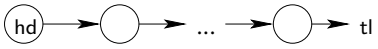
- Σ is a separating conjunction of atomic propositions emp, $\mathsf{u}_0 \mapsto (\mathsf{v}_1, \ldots, \mathsf{v}_k)$ and predicates $\mathsf{P}_i(\mathsf{u}_1, \ldots, \mathsf{u}_{\#(\mathsf{P}_i)})$ with $\mathsf{u}_0, \ldots, \mathsf{u}_{\#(\mathsf{P}_i)}, \mathsf{v}_1, \ldots, \mathsf{v}_k \in \{\mathsf{x}_1, \ldots, \mathsf{x}_p\} \cup \{\mathsf{y}_1, \ldots, \mathsf{y}_q\}$, and
- Π is a conjunction of equalities and disequalities between the variables in $\{\mathsf{x}_1, \ldots, \mathsf{x}_p\} \cup \{\mathsf{y}_1, \ldots, \mathsf{y}_q\}$.

This restriction has little impact on the expressive power of the logic, because many useful recursive data structures can still be encoded as inductive predicates of this form, as in Figure 7.1. However, even with these restrictions, the entailment problem remains undecidable (Lemma 7). To work around this problem, we assume that each rule $\mathsf{r}(\mathsf{x}_1, \ldots, \mathsf{x}_p)$ in the given inductive system meets the following conditions:

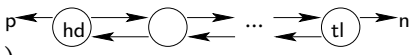
1. *Progress.* The rule r contains exactly one occurrence of a points-to proposition $\mathsf{u} \mapsto (\mathsf{v}_1, \ldots, \mathsf{v}_k)$. We say that the rule r *allocates* the variable u. A predicate allocates a parameter x_i if each rule in its definition allocates x_i.
2. *Connectivity.* There exists at least one points-to edge (possibly involving equalities) between the variable allocated in r and the variable allocated in each of the predicates occurring in r.

3. *Establishment.* All existentially quantified variables in r are eventually allocated, in every unfolding of the predicates occurring in r.

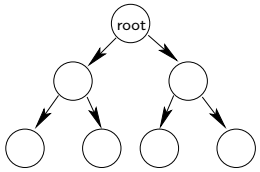
$$\mathsf{list}(\mathsf{hd},\mathsf{tl}) \equiv \mathsf{emp} \wedge \mathsf{hd} = \mathsf{tl} \vee \exists \mathsf{x} \,.\, \mathsf{hd} \mapsto \mathsf{x} * \mathsf{list}(\mathsf{x},\mathsf{tl})$$



$$\mathsf{dll}(\mathsf{hd},\mathsf{p},\mathsf{tl},\mathsf{n}) \equiv \mathsf{hd} \mapsto (\mathsf{n},\mathsf{p}) \wedge \mathsf{hd} = \mathsf{tl} \vee \exists \mathsf{x} \,.\, \mathsf{hd} \mapsto (\mathsf{x},\mathsf{p}) * \mathsf{dll}(\mathsf{x},\mathsf{hd},\mathsf{tl},\mathsf{n})$$



$$\mathsf{tree}(\mathsf{root}) \equiv \mathsf{emp} \wedge \mathsf{root} = \mathsf{nil} \vee \exists \mathsf{l} \exists \mathsf{r} \,.\, \mathsf{root} \mapsto (\mathsf{l},\mathsf{r}) * \mathsf{tree}(\mathsf{l}) * \mathsf{tree}(\mathsf{r})$$



$$\mathsf{tll}(\mathsf{root},\mathsf{ll},\mathsf{lr}) \equiv \begin{array}{ll} \mathsf{root} \mapsto & (\mathsf{nil},\mathsf{nil},\mathsf{lr}) \wedge \mathsf{root} = \mathsf{ll} \vee \\ \exists \mathsf{l} \exists \mathsf{r} \exists \mathsf{z} \,. & \mathsf{root} \mapsto (\mathsf{l},\mathsf{r},\mathsf{nil}) * \mathsf{tll}(\mathsf{l},\mathsf{ll},\mathsf{z}) * \\ & \mathsf{tll}(\mathsf{r},\mathsf{z},\mathsf{lr}) \end{array}$$

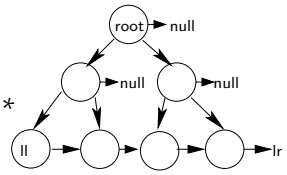


Figure 7.1: Inductive Data Structures Defined in SL.

We call $\mathsf{SL}_{\mathrm{btw}}$ the fragment of SL involving predicates defined according to conditions (1), (2) and (3). In the following, we formalize these conditions and prove that the satisfiability and entailment problems become decidable in their presence. Given a conjunction of equalities and disequalities Π, we write $\mathsf{x} =_\Pi \mathsf{y}$ if $\mathsf{x} = \mathsf{y}$ is a consequence of Π.

Definition 13. *A rule* $\mathsf{r}(\mathsf{x}_1,\ldots,\mathsf{x}_n) \equiv \exists \mathsf{y}_1 \ldots \exists \mathsf{y}_m \,.\, \mathsf{u} \mapsto (\mathsf{v}_1,\ldots,\mathsf{v}_k) * \mathsf{P}_{i_1}(\overrightarrow{\mathsf{y}_1}) * \ldots * \mathsf{P}_{i_s}(\overrightarrow{\mathsf{y}_m}) \wedge \Pi$ *is* connected *if, for each* $j = 1,\ldots,s$*, there exists a variable* $(\overrightarrow{\mathsf{y}_j})_\ell$ *such that* $\mathsf{v}_p =_\Pi (\overrightarrow{\mathsf{y}_j})_\ell$ *and the parameter* $\mathsf{x}_{j,\ell}$ *of* $\mathsf{P}_j(\mathsf{x}_{j,1},\ldots,\mathsf{x}_{j,m_j})$ *is allocated by* P_j*, for some* $1 \leq \ell \leq m_j$*.*

An inductive system is *connected* if each of its rules is connected and *disconnected* otherwise. Observe that all inductive systems from Figure 7.1 are connected. The following lemma shows that connectivity is a necessary condition for decidability of entailments:

Lemma 7. *The entailment problem is undecidable for disconnected inductive systems.*

Proof. By reduction from the undecidability of the universality problem for context-free languages. Given a context-free grammar $G = \langle \Xi, \mathcal{A}, \Delta \rangle$ and a nonterminal $X \in \Xi$, we build an inductive system $\mathcal{P}_G$ having a predicate P_Y for each nonterminal $Y \in \Xi$, such that $L_Y(G)$ and $[\![P_Y]\!]$ are in one-to-one relation (each word corresponds to a singly-linked list with appropriate selectors). Moreover, we consider a self-recursive predicate P^* that encodes the universal language $\mathcal{A}^*$. Then $L_X(G)$ is universal iff $P^* \models_{\mathsf{SL}} P_X$. □

In the following we aim at proving that, given a predicate P of an inductive SL_{btw} system, all states in $[\![P]\!]$ are represented by graphs of *bounded treewidth*, hence the name of the logic. Let us first introduce this notion.

For a state $S = \langle s, h \rangle$, let $\mathrm{loc}(S) = \mathrm{dom}(h) \cup \bigcup_{\ell \in \mathrm{dom}(h), i \in \mathsf{Sel}} (h(\ell))(i) \cup \{s(\mathsf{u}) \mid \mathsf{u} \in \mathrm{dom}(s)\}$ be the set of locations that occur in S, either in the domain of the heap, referred to by a selector edge or pointed to by a variable in the store. For a tree t and $p, q, r \in \mathrm{dom}(t)$, we say that q is between p and r if there exists a sequence $p = p_0, \ldots, p_n = r \in \mathrm{dom}(t)$ of pairwise distinct positions, such that p_{i+1} is either the parent, or a child of p_i, for all $i = 0, \ldots, n-1$ and $p_j = q$ for some $0 \leq j \leq n$.

Definition 14. *Given a state $S = \langle s, h \rangle$, a* tree decomposition *of S is a tree $t : \mathbb{N}^* \rightharpoonup_{fin} 2^{\mathrm{loc}(S)}$ such that:*

1. $\mathrm{loc}(S) = \bigcup_{p \in \mathrm{dom}(t)} t(p)$,
2. *for each edge $\ell \xrightarrow{i} \ell'$ in S there exists $p \in \mathrm{dom}(t)$ such that $\ell, \ell' \in t(p)$,*
3. *for each $p, q, r \in \mathrm{dom}(t)$, q is between p and r only if $t(p) \cap t(r) \subseteq t(q)$.*

The width of the decomposition is $w(t) = \max_{p \in \mathrm{dom}(t)} \{\|t(p)\| - 1\}$. The treewidth of S is $tw(S) = \min \{w(t) \mid t \text{ is a tree decomposition of } S\}$.

For instance, the treewidth of a state representing a tree data structure is 1. The optimal tree decomposition is an isomorphic tree t such that $t(p.i) = \{p, p.i\}$, for every $p \in \mathrm{dom}(t)$ and $i \in \mathbb{N}$ such that $p.i \in \mathrm{dom}(t)$. On the other hand, the set of treewidths of $n \times n$ squared grids, for $n > 0$, does not have an upper bound [See91]. The following condition is required to ensure that the set $\{tw(S) \mid S \in [\![P]\!]\}$ is bounded, for any predicate P of an inductive system.

Definition 15. *Given a rule $\mathsf{r}(\mathsf{x}_1, \ldots, \mathsf{x}_n) \equiv \exists \mathsf{y}_1 \ldots \exists \mathsf{y}_m \ . \ \mathsf{u} \mapsto (\mathsf{v}_1, \ldots, \mathsf{v}_k) * \mathsf{P}_{i_1}(\vec{\mathsf{y}_1}) * \ldots * \mathsf{P}_{i_m}(\vec{\mathsf{y}_s}) \wedge \Pi$, the parameter x_i, for $1 \leq i \leq n$ is* allocated in r *if and only if: (i) $\mathsf{x}_i =_\Pi \mathsf{u}$, or (ii) there exists $1 \leq j \leq s$ such that $\mathsf{x}_i =_\Pi (\vec{\mathsf{y}_j})_q$ and the corresponding parameter $\mathsf{x}_{j,q}$ is allocated in every rule of $P_j(\mathsf{x}_{j,1}, \ldots, \mathsf{x}_{j,m_j})$, for some $1 \leq q \leq m_j$. Moreover, the rule r is* established *is for every $j = 1, \ldots, m$ there exists $1 \leq \ell \leq s$ and $1 \leq q \leq m_\ell$ such that $\mathsf{y}_j =_\Pi (\vec{\mathsf{y}_\ell})_q$ and the corresponding parameter $\mathsf{x}_{i_j,q}$ is allocated.*

An inductive system is *established* if each of its rules is established. Consider, for instance, the predicate $\mathsf{le}(\mathsf{x}) \equiv \mathsf{x} \mapsto (\mathsf{nil}, \mathsf{nil}) \vee \exists \mathsf{y}, \mathsf{z} \,.\, \mathsf{x} \mapsto (\mathsf{y}, \mathsf{z}) * \mathsf{le}(\mathsf{y})$. The set of models $[\![\mathsf{le}]\!]$ consists of singly-linked lists in which the first selector points to the successor node and the second selector may point to any node in the heap. Then $[\![\mathsf{le}]\!]$ contains a square grid of size $n \times n$ for each list of length n^2, thus the set $\{tw(S) \mid S \in [\![\mathsf{le}]\!]\}$ is not bounded.

The following lemma proves that conditions (1), (2) and (3) are sufficient to ensure boundedness of this set, for each $\mathsf{SL}_{\mathrm{btw}}$-definable inductive system. For a rule $\mathsf{r}(\mathsf{x}_1, \ldots, \mathsf{x}_n) \equiv \exists \mathsf{y}_1 \ldots \exists \mathsf{y}_m \,.\, \Sigma \wedge \Pi$, we define $|\mathsf{r}|_{\mathrm{nvar}} = n + m$, i.e. the number of variables, both parameters and existentially quantified, that occur in r. For an inductive system $\mathcal{P} = \left\{\mathsf{P} \equiv \bigvee_{j=1}^{m_i} \mathsf{r}_{ij}\right\}_{i=1}^{n}$, we define $|\mathcal{P}|_{\mathrm{nvar}} = \max_{i=1}^{n} \max_{j=1}^{m_i} |\mathsf{r}|_{\mathrm{nvar}}$ to be the maximum such number.

Lemma 8. *Given $\mathcal{P}$ a connected and established inductive system, which, moreover satisfies the progress condition (1) and $\mathsf{P} \in \mathcal{P}$ a predicate, for each state $S \in [\![\mathsf{P}]\!]$, we have $tw(S) = \mathcal{O}(|\mathcal{P}|_{\mathrm{nvar}})$.*

Proof. For $S = \langle s, h \rangle \in [\![\mathsf{P}]\!]$, the connectivity condition defines the structure of the tree decomposition, which is a tree t that connects all locations in $\mathrm{loc}(S)$, such that each edge $(p, p.i) \in \mathrm{dom}(t) \times \mathrm{dom}(t)$ corresponds to an existing selector edge $\ell \xrightarrow{i} \ell'$ in S. The bags of t are defined as follows. First $\ell, \ell' \in t(p)$ for each selector edge $\ell \xrightarrow{i} \ell'$ of S that maps to some edge $(p, p.i) \in \mathrm{dom}(t) \times \mathrm{dom}(t)$. Let $\ell \xrightarrow{i} \ell'$ be a selector edge in S, not already covered by the edges of t such that $\ell \in t(p)$ and $\ell' \in t(p')$. Then we add ℓ' to all bags $t(q)$, where q is between p and p' in t. It remains to show that $w(t) \leq |\mathcal{P}|_{\mathrm{nvar}}$. This is the case because for each selector edge $\ell \xrightarrow{i} \ell'$ of S, not covered by an edge of t, $\ell' \in t(q)$ only if there exists a chain $\mathsf{y}_1 = \ldots = \mathsf{y}_n$ of equalities between pairwise distinct existentially quantified variables, which corresponds to a path from p to q. Since each y_i is created by a distinct application of a rule in $\mathcal{P}$ and the number of existentially quantified variables in a rule is bounded by $|\mathcal{P}|_{\mathrm{nvar}}$, we have that $tw(S) = \mathcal{O}(|\mathcal{P}|_{\mathrm{nvar}})$. □

Example 17. *Let us consider the model of the* tll *predicate (Figure 7.1) shown in Figure 7.2 (a). A possible tree decomposition follows the unfolding structure of the* tll *definition. Each rule allocates a variable to a location ℓ^p, where p is the corresponding tree position. The bags of the tree decomposition contain the locations $\ell^p, \ell^{p.0}$ and $\ell^{p.1}$ for each $p \in \mathrm{dom}(t) \setminus \mathrm{Fr}(t)$. For each non-local edge, the destination location is added to all bags between the position where the source and the destination are allocated. For instance, for the edge $\ell^{01} \xrightarrow{3} \ell^{10}$, we add ℓ^{10} to the bags at positions $1, \varepsilon, 0$ and 01.* ■

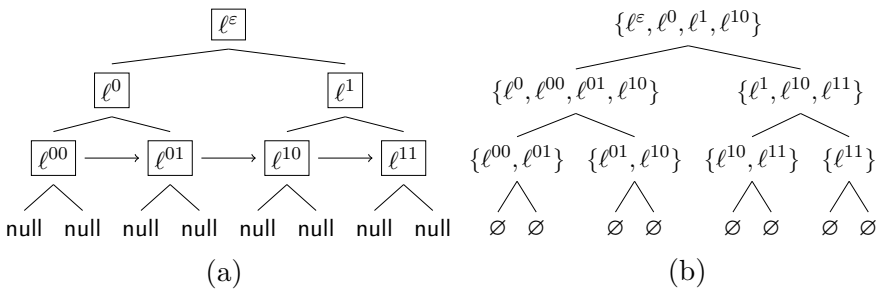


Figure 7.2: Tree Decomposition of a Tree with Linked Leaves

The proofs of decidability for the satisfiability and entailment problems in $\mathsf{SL}_{\mathrm{btw}}$ are based on an encoding of the semantics of $\mathsf{SL}_{\mathrm{btw}}$ into Monadic Second Order logic (MSO) interpreted over states. That is, for each predicate $\mathsf{P} \in \mathsf{Pred}$, we have an MSO formula Ψ_{P} such that $S \in [\![\mathsf{P}]\!] \Leftrightarrow S \models_{\mathsf{MSO}} \Psi_{\mathsf{P}}$. Since, moreover, the set $[\![\mathsf{P}]\!]$ has bounded treewidth (Lemma 8) the satisfiability and entailment problems for $\mathsf{SL}_{\mathrm{btw}}$ reduce to the satisfiability problem for MSO over states (graphs) of bounded treewidth. This problem is known to be decidable, by Courcelle's Theorem [Cou90].

In particular, the establishment condition (Definition 15) is not necessary to check satisfiability of a $\mathsf{SL}_{\mathrm{btw}}$ predicate. In this case, it is sufficient to show that, if P has a model, then it also has a model of bounded treewidth (using the same idea as in the proof of Lemma 8).

On the other hand, the entailment problem requires this condition, as explained next. Given the entailment $\mathsf{P} \models_{\mathsf{SL}} \mathsf{Q}$, we build MSO formulae Ψ_{P} and Ψ_{Q} and reduce the problem to checking the satisfiability of the formula $\Psi_{\mathsf{P}} \wedge \neg\Psi_{\mathsf{Q}}$. Since any model S of this formula is also a model of P, by Lemma 8 its treewidth is bounded by a linear function in the maximum number of variables in the inductive system $|\mathcal{P}_{\mathrm{nvar}}|$. By Courcelle's Theorem [Cou90], the satisfiability problem is decidable for any formula $\Psi_{\mathsf{P}} \wedge \neg\Psi_{\mathsf{Q}}$ derived from an entailment problem $\mathsf{P} \models_{\mathsf{SL}} \mathsf{Q}$, which proves that the entailment problem is decidable for $\mathsf{SL}_{\mathrm{btw}}$.

In the rest of this section, we sketch the translation of $\mathsf{SL}_{\mathrm{btw}}$ into MSO. Let $\mathsf{Var}_{\mathrm{mso}}$ be a set of first- and second-order variables. The first-order variables $x, y, \ldots$ range over locations and the second-order variables $X, Y, \ldots$

range over sets of locations. The syntax of MSO is given below:

$$\begin{array}{rcl} x, y, X & \in & \mathsf{Var}_{\mathrm{mso}} \\ i & \in & \mathsf{Sel} \\ \varphi & := & x = y \mid \mathrm{edge}_i(x, y) \mid \mathrm{null}(x) \mid X(x) \mid \varphi \wedge \varphi \mid \neg\varphi \mid \exists x \,.\, \varphi \mid \exists X \,.\, \varphi \end{array}$$

An interpretation $\iota : \mathsf{Var}_{\mathrm{mso}} \rightharpoonup \mathsf{Loc} \cup 2^{\mathsf{Loc}}$ associates first-order variables with locations and second-order variables with sets of locations. The semantics of MSO is given by a forcing relation $\langle s, h\rangle, \iota \vDash_{\mathsf{MSO}} \varphi$, defined by induction on the structure of φ, where $\langle s, h\rangle$ is a state and ι is an interpretation:

$$\begin{array}{rcl} \langle s, h\rangle, \iota \vDash_{\mathsf{MSO}} \mathrm{null}(x) & \Leftrightarrow & \iota(x) = \mathsf{nil} \\ \langle s, h\rangle, \iota \vDash_{\mathsf{MSO}} \mathrm{edge}_i(x, y) & \Leftrightarrow & (h(\iota(x)))(i) = \iota(y) \end{array}$$

The semantics of the first- and second-order quantifiers $\exists x$, $\exists X$ and that of the boolean connectives $\wedge, \neg$ is the classical one. The problem asking for the existence of a state S such that $tw(S) \leq k$ and $S \vDash_{\mathsf{MSO}} \varphi$, where $k > 0$ is a given integer constant and φ is a closed MSO formula, is decidable [Cou90]. It is known that the complexity of this problem is non-elementary, even when MSO is interpreted over words, with successor function [Sto74].

The translation of $\mathsf{SL}_{\mathrm{btw}}$ into MSO maps any SL variable $\mathsf{x} \in \mathsf{Var}$ into a first-order variable $x \in \mathsf{Var}_{\mathrm{mso}}$ and the special variable nil into x_{nil}, with an associated constraint $\mathrm{null}(x_{\mathsf{nil}})$. The SL atomic propositions and separating conjunction are translated into MSO formulae, with a free variable X denoting the set of locations in the heap, by a recursive function $Tr(\phi)$:

$$\begin{array}{rcl} Tr(\mathsf{emp}) & \equiv & \forall x \,.\, \neg X(x) \\ Tr(\mathsf{x} \mapsto (\mathsf{y}_1, \ldots, \mathsf{y}_k)) & \equiv & \mathrm{Sing}(x, X) \wedge \bigwedge_{i=1}^{k} \mathrm{edge}_i(x, y_i) \\ Tr(\phi_1 * \phi_2) & \equiv & \exists Y \exists Z \,.\, Tr(\phi_1)[Y/X] \wedge Tr(\phi_2)[Z/X] \wedge \mathrm{Part}(Y, Z, X) \end{array}$$

where $\mathrm{Sing}(x, X)$ means that X is a singleton whose only element is x and $\mathrm{Part}(Y, Z, X)$ means that Y and Z are a partition of X. Clearly, these are MSO-definable constraints.

The next step is to define the (possibly infinite) set of states that are models of a given predicate P_i defined by an inductive system $\mathcal{P}$. We observe that each such state can be decomposed into a tree-like structure, called the *backbone* and a set of *non-local* edges, defined below:

Definition 16. *A* backbone *of a state $S = \langle s, h\rangle$ is a bijective tree $t : \mathbb{N}^* \to \mathrm{dom}(h)$ such that for all $p \in \mathrm{dom}(t)$ and $d \in \mathbb{N}$, $p.d \in \mathrm{dom}(t)$ only if $t(p) \xrightarrow{i} t(p.d)$ is a selector edge of S, for some $i \in \mathsf{Sel}$. Given a backbone t of S, a selector edge $\ell \xrightarrow{i} \ell'$ of S is* local w.r.t. *t if $\ell = t(p)$ and $\ell' = t(p.d)$ for some positions $p, p.d \in \mathrm{dom}(t)$.*

Intuitively, each state $S \in [\![\mathsf{P}_i]\!]$ is the model of a first-order SL formula obtained by a finite unfolding of P_i. In this respect, the progress condition (1) ensures that each unfolding step (corresponding to the application of a rule) associates exactly one variable to a location in the domain of the heap.

The translation uses a second-order variable X_{ij}^d for the set of locations allocated by the rule r_{ij}, when applied to a backbone position $p = q.d$ that is the d-th child of its parent ($d = -1$ when $p = \varepsilon$). The set of local edges between the positions q and $q.d$ of the unfolding tree, where q is labeled by a rule r_{ij} and $q.d$ is labeled by a rule $\mathsf{r}_{\ell k}$ is defined by a constraint $local_{i,j,\ell,k}^d(x,y)$, inferred by an analysis of r_{ij} and $\mathsf{r}_{\ell k}$. Then the successor relation in the backbone is defined by an MSO formula:

$$succ^d(x,y) \equiv \bigvee_{\substack{1 \leq i,\ell \leq n \\ 1 \leq j \leq m_i \\ 1 \leq k \leq m_\ell}} X_{ij}(x) \wedge X_{\ell k}^d(y) \wedge local_{i,j,\ell,k}^d(x,y)$$

With this definition, the backbone of any state $S \in [\![\mathsf{P}_i]\!]$ is defined by an MSO formula $backbone_i(x, \mathbf{X})$, where x is the root of the backbone and $\mathbf{X}$ is the set of all variables X_{ij}^d. Intuitively, this formula ensures that the sets in $\mathbf{X}$ define a tree structure which, moreover, corresponds to a correct unfolding of the predicate P_i in the inductive system $\mathcal{P}$.

The last step of the translation is the definition of non-local edges of a state w.r.t. the backbone defined by its unfolding tree. For instance, the non-local edge $\ell^{01} \xrightarrow{3} \ell^{10}$ in Figure 7.2 (a) is the consequence of the atomic propositions $\mathsf{r}^0 \mapsto (\mathsf{nil}, \mathsf{nil}, \mathsf{lr}^{01})$, $\mathsf{l}^1 \mapsto (\mathsf{nil}, \mathsf{nil}, \mathsf{ll}^{10})$ and the chain of equalities $\mathsf{lr}^{01} = \mathsf{lr}^0 = \mathsf{z}^\varepsilon = \mathsf{ll}^1 = \mathsf{l}^1$, where $\mathsf{r}^0, \mathsf{z}^\varepsilon, \mathsf{l}^1, \mathsf{lr}^{01}, \mathsf{lr}^0, \mathsf{ll}^1$ and ll^{10} are existentially quantified variables in the SL formula corresponding to the unfolding tree.

The main idea is that these chains of equalities follow regular paths in the backbone tree, which can be encoded using *tree walking automata* (TWA) [Boj08]. For instance, in the case of the tll predicate in Figure 7.1, all non-local edges correspond to regular walks of the form $\nwarrow^* \nearrow \searrow \swarrow^*$ in the backbone tree, where $\nearrow$ ($\nwarrow$) moves up when being a left (right) child and $\swarrow$ ($\searrow$) moves down to the left (right) child. In a nutshell, we define a TWA $A_{\mathcal{P}}$ for the entire inductive system $\mathcal{P}$, where the source and the destination of each non-local edge are marked by the initial and final states of $A_{\mathcal{P}}$, respectively. Then we build an MSO formula $non\_local(\mathbf{X})$ which encodes the set of trees accepted by $A_{\mathcal{P}}$. We refer the interested reader to [IRS13] for the details of these definitions.

Finally, the parameters of the predicate $\mathsf{P}_i(\mathsf{x}_{i,1}, \ldots, \mathsf{x}_{i,\#(\mathsf{P}_i)})$ have to be mapped to locations in the heap. Because parameters can be propagated

though an unfolding tree via equalities, we use another TWA to track their final destinations. By encoding this TWA into MSO, we obtain an MSO formula $param(x_{i,1}, \ldots, x_{i,\#(\mathsf{P}_i)}, \mathbf{X})$ describing these equalities. The formula Ψ_{P_i} describing the models of P_i is then defined as follows:

$$\Psi_{\mathsf{P}_i} \equiv \exists x \exists \mathbf{X} \ . \ backbone_i(x, \mathbf{X}) \wedge non\_local(\mathbf{X}) \wedge param(x_{i,1}, \ldots, x_{i,\#(\mathsf{P}_i)}, \mathbf{X})$$

Then an entailment problem $\mathsf{P}_i(\mathsf{x}_1, \ldots, \mathsf{x}_p) \models_{\mathsf{SL}} \mathsf{P}_j(\mathsf{x}_1, \ldots, \mathsf{x}_p)$ is equivalent to the unsatisfiability of the MSO formula $\Psi_{\mathsf{P}_i}(x_1, \ldots, x_n) \wedge \neg \Psi_{\mathsf{P}_j}(x_1, \ldots, x_n)$, interpreted over states with treewidth bounded by $\mathcal{O}(|\mathcal{P}|_{\text{nvar}})$. By Courcelle's Theorem [Cou90], this latter problem is decidable.

A finer analysis of the translation of SL_{btw} predicates into MSO and of the translation of MSO over graphs of bounded treewidth into MSO over trees [Cou90] reveals that the number of quantifier alternations in the final formulae is always bounded by a constant. This means that the satisfiability problem for these formulae has elementary time complexity (although an upper bound computed in this way would involve several exponentials), leading to the following result:

Theorem 24 ([IRS13])**.** *The satisfiability and entailment problems for* SL_{btw} *are in* ELEMENTARY*.*

7.2 An Exptime Inductive Fragment of SL

The SL_{btw} defined in the previous section is not very appealing for program verification purposes, due to the high complexity of the MSO translation and the blowup in the size of automata needed to check satisfiability of MSO formulae over trees. For this reason, we define a subset of SL_{btw}, called SL_{loc}, for which entailments can be encoded as inclusion of languages recognized by tree automata and, moreover, the reduction takes polynomial time in the size of the input inductive system. Since the language inclusion problem for TA is EXPTIME-complete, we derive a similar result for the entailment problem in SL_{loc}, with a matching lower bound obtained by reduction from the universality problem for TA [CDG+05, Theorem 1.7.7, Corollary 1.7.9].

We recall that a variable $\mathsf{x} \in \mathsf{Var}$ is *allocated* in an SL_{btw} formula if it occurs on the left-hand side of an atomic proposition $\mathsf{x} \mapsto (\mathsf{y}_1, \ldots, \mathsf{y}_k)$. A variable y_i is *referenced* if it occurs on the right-hand side of such a proposition. With these notions, we define the subset SL_{loc} consisting only of *local* inductive systems:

Definition 17. *An inductive system* $\mathcal{P} = \left\{\mathsf{P}_i(\mathsf{x}_{i,1}, \ldots, \mathsf{x}_{i,\#(\mathsf{P}_i)})\right\}_{i=1}^{n}$ *is* local *if and only if:*

- *each parameter* $\mathsf{x}_{i,j}$ *is allocated in each rule of* P_i *and* $(\vec{\mathsf{y}}_j)$ *is referenced at each occurrence* $\mathsf{P}_i(\vec{\mathsf{y}})$ *in* $\mathcal{P}$*, or*
- *each parameter* $\mathsf{x}_{i,j}$ *is referenced in each rule of* P_i *and* $(\vec{\mathsf{y}}_j)$ *is allocated at each occurrence* $\mathsf{P}_i(\vec{\mathsf{y}})$ *in* $\mathcal{P}$*.*

For instance, the $\mathsf{list}, \mathsf{dll}$ and tree systems in Figure 7.1 are local, whereas tll is not. We recall that, due to the progress (1) and connectivity (2) conditions, each state $S \in [\![\mathsf{P}_i]\!]$ is the model of a basic SL formula corresponding to an unfolding tree of the predicate P_i. The locality condition ensures furthermore that each selector edge of a state $S \in [\![\mathsf{P}_i]\!]$ is local with respect to the backbone defined by the unfolding tree (Definition 16).

The first step of the entailment decision procedure is building a TA for a given local inductive system. Roughly speaking, the TA we build recognizes unfolding trees of the inductive system. The alphabet of such a TA consists of small basic SL formulae describing the neighborhood of each allocated variable, together with a specification of the connections between each such formula and its parent and children in the unfolding tree. Each alphabet symbol in the TA is called a *tile*. Once the tile alphabet is defined, the states of the TA correspond naturally to the predicates of the inductive system, and the transition rules correspond to the rules of the system.

Formally, a *tile* is a tuple $T = \langle \varphi, \mathbf{x}_{-1}, \mathbf{x}_0, \ldots, \mathbf{x}_{d-1} \rangle$, for some $d \geq 0$, where $\varphi \equiv (\exists \mathsf{u})\ \mathsf{u} \mapsto (\mathsf{v}_1, \ldots, \mathsf{v}_k) \wedge \Pi$ is a SL formula in which u is possibly existentially quantified, Π is a conjunction of equalities between variables, and each $\mathbf{x}_i$ is a tuple of pairwise distinct variables, called a *port*. We further assume that all ports contain only free variables from φ and that they are pairwise disjoint. The variables $\mathbf{x}_{-1}$ are *incoming*, $\mathbf{x}_0, \ldots, \mathbf{x}_{d-1}$ are *outgoing*, and $\mathrm{par}(T) = FV(\varphi) \setminus (\mathbf{x}_{-1} \cup \ldots \cup \mathbf{x}_{d-1})$ are called *parameters*. The *arity* of a tile $T = \langle \varphi, \mathbf{x}_{-1}, \ldots, \mathbf{x}_{d-1} \rangle$ is the number of outgoing ports, denoted by $\#(T) = d$. We denote $\mathrm{form}(T) \equiv \varphi$ and $\mathrm{port}_i(T) \equiv \mathbf{x}_i$, for all $-1 \leq i < d$. Moreover, we assume that each port $\mathbf{x}_i$ can be factorized as $\mathbf{x}_i^{fw} \cdot \mathbf{x}_i^{bw}$ (distinguishing *forward* links going from the root to the leaves and *backward* links going in the opposite direction, respectively) such that:

- the backward incoming tuple $\mathbf{x}_{-1}^{bw}$ consists only of variables referenced by the unique allocated variable u, ordered by the corresponding selectors.
- each forward outgoing tuple $\mathbf{x}_i^{fw}$ consists of variables referenced by the unique allocated variable z, ordered by the corresponding selectors.
- $(\mathbf{x}_{-1}^{fw} \cup \mathbf{x}_0^{bw} \cup \ldots \cup \mathbf{x}_{d-1}^{bw}) \cap \{\mathsf{v}_1, \ldots, \mathsf{v}_k\} = \varnothing$ and $\Pi \equiv \mathbf{x}_{-1}^{fw} = \mathsf{u} \ \wedge \ \bigwedge_{i=0}^{d-1} \mathbf{x}_i^{bw} = \mathsf{u}$.[1]

We denote by $\mathrm{port}_i^{fw}(T)$ and $\mathrm{port}_i^{bw}(T)$ the tuples $\mathbf{x}_i^{fw}$ and $\mathbf{x}_i^{bw}$, respectively, for all $-1 \leq i < d$. The set of canonical tiles is denoted as $\mathcal{T}$.

[1] For a tuple $\mathbf{x} = \langle \mathsf{x}_1, \ldots, \mathsf{x}_k \rangle$, we write $\mathbf{x} = \mathsf{u}$ for $\bigwedge_{i=1}^{k} \mathsf{x}_i = \mathsf{u}$.

Definition 18. *A tree* $t : \mathbb{N}^* \rightharpoonup_{fin} \mathcal{T}$ *is* canonical *if, for each* $0 \leq i < \#(t(p))$, *we have* $|\text{port}_i^{fw}(t(p))| = |\text{port}_{-1}^{fw}(t(p.i))|$ *and* $|\text{port}_i^{bw}(t(p))| = |\text{port}_{-1}^{bw}(t(p.i))|$.

Given a tree $t : \mathbb{N}^* \rightharpoonup_{fin} \mathcal{T}$ labeled with tiles, we denote by $\Phi(t)$ its *characteristic formula*, obtained by renaming each variable x in $t(p)$ by x^p, for each $p \in \text{dom}(t)$ and equating the overlapping incoming and outgoing variables.

An important property of canonical trees is that each state S that is a model of the characteristic formula $\Phi(t)$ of a canonical tree t ($S \models_{\mathsf{SL}} \Phi(t)$) has a unique backbone u, that is isomorphic to t, i.e. $\text{dom}(u) = \text{dom}(t)$. For an example, consider a state $S \in [\![\mathsf{dll}]\!]$ (Figure 7.1) encoded by two different canonical trees (Figure 7.3).

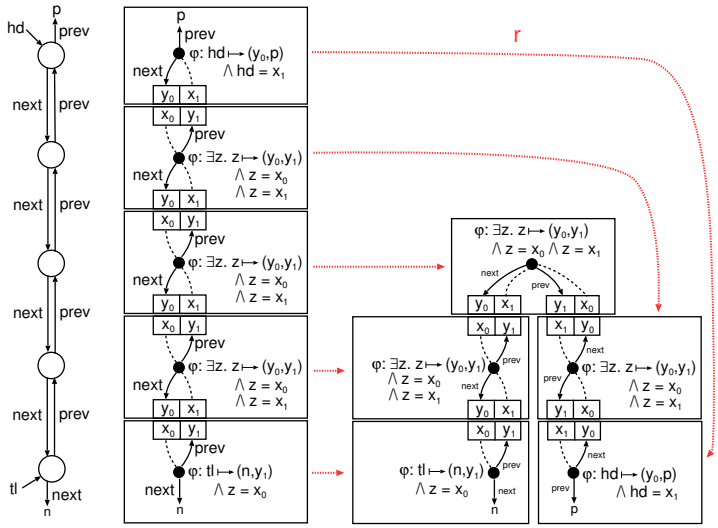


Figure 7.3: A state $S \models_{\mathsf{SL}} \mathsf{dll}(\mathsf{hd}, \mathsf{p}, \mathsf{tl}, \mathsf{n})$ and two canonical tree encodings

Given a predicate $\mathsf{P}_i(\mathsf{u}_1, \ldots, \mathsf{u}_{\#(\mathsf{P}_i)})$ defined by a local inductive system $\mathcal{P}$, we build the tree automaton $A^i_{\mathcal{P}}$ recognizing the canonical trees that encode the set of states $[\![\mathsf{P}_i]\!]$, as follows. For each rule in the system, we create a tile whose incoming and outgoing ports $\mathbf{x}_i$ are factorized as $\mathbf{x}_i^{fw} \cdot \mathbf{x}_i^{bw}$. The backward part of the input port $\mathbf{x}_{-1}^{bw}$ and the forward parts of the output ports $\{\mathbf{x}_i^{fw}\}_{i \geq 0}$ are sorted according to the order of incoming selector edges from the single points-to formula in the rule. The output ports $\{\mathbf{x}_i\}_{i \geq 0}$ are sorted within the tile according to the order of the selector edges pointing to $(\mathbf{x}_i^{fw})_0$ for each $i \geq 0$. Finally, each predicate name P_j is associated with a state q_j, and for each inductive rule, the procedure creates a transition rule in the TA. The final state is q_i, corresponding to the predicate P_i. The

size of $A^i_{\mathcal{P}}$ is linear in the size of the input inductive system $\mathcal{P}$.

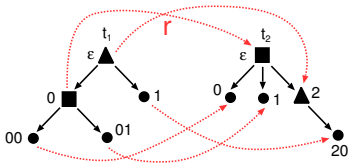
However, there are instances of the entailment problem that cannot be directly solved by language inclusion between tree automata defined above, due to the following *polymorphic representation problem*: the same set of states can be defined by two different inductive predicates, and the tree automata mirroring their definitions will report that the entailment does not hold. This happens because the tile alphabets of the two automata are different. For example, doubly-linked lists can also be defined in reverse:

$$\mathsf{dll}_{\mathrm{rev}}(\mathsf{hd},\mathsf{n},\mathsf{tl},\mathsf{p}) \equiv \mathsf{hd} \mapsto (\mathsf{p},\mathsf{n}) \wedge \mathsf{hd} = \mathsf{tl} \ \vee \ \exists \mathsf{x} \ . \ \mathsf{tl} \mapsto (\mathsf{x},\mathsf{n}) * \mathsf{dll}_{\mathrm{rev}}(\mathsf{hd},\mathsf{tl},\mathsf{x},\mathsf{p})$$

The solution to this problem comes from the following observation. If a state S has local edges w.r.t. two different backbones t_1 and t_2, then we can obtain t_2 from t_1 by picking a position $p \in dom(t_1)$ and making it the root of t_2, while maintaining in t_2 all edges from t_1 (Fig. 7.2).

Definition 19. *Given two trees $t_1, t_2 : \mathbb{N}^* \rightharpoonup_{fin} \Sigma$ and a bijective mapping $r : dom(t_1) \to dom(t_2)$, we say that t_2 is an* r-rotation *of t_1, denoted $t_1 \sim_r t_2$, iff $\forall p \in dom(t_1) \forall d \in \mathcal{D}_+(t_1) : p.d \in dom(t_1) \Rightarrow \exists e \in \mathcal{D}(t_2) \ . \ r(p.d) = r(p).e$.*

We write $t_1 \sim t_2$ if there exists a bijective mapping $r : dom(t_1) \to dom(t_2)$ such that $t_1 \sim_r t_2$. An example of a rotation r of a tree t_1 to a tree t_2 such that $r(\varepsilon) = 2$, $r(0) = \varepsilon$, $r(1) = 20$, $r(00) = 0$, and $r(01) = 1$ is shown in Fig. 7.2.



Next, we define rotation on canonical trees. This definition is a refinement of Definition 19. Namely, the change in the structure of the tree is mirrored by a change in the tile alphabet labeling the tree in order to preserve the state which is represented by the canonical tree.

Definition 20. *Given two canonical trees $t, u : \mathbb{N}^* \rightharpoonup_{fin} \mathcal{T}^c$ and a bijective mapping $r : dom(t) \to dom(u)$, we say that u is a* canonical rotation *of t, denoted $t \sim^c_r u$, if and only if $t \sim_r u$ and there exists a substitution $\sigma_p : Var \rightharpoonup_{fin} Var$ for each $p \in dom(t)$ such that $\mathrm{form}(t(p))[\sigma_p] \equiv \mathrm{form}(u(r(p)))$ and, for all $0 \leq i < \#_t(p)$, there exists $j \in \mathcal{D}(u)$ such that $r(p.i) = r(p).j$ and:*

$$\mathrm{port}^{fw}_i(t(p))[\sigma_p] \equiv \textit{if } j \geq 0 \textit{ then } \mathrm{port}^{fw}_j(u(r(p))) \textit{ else } \mathrm{port}^{bw}_{-1}(u(r(p)))$$
$$\mathrm{port}^{bw}_i(t(p))[\sigma_p] \equiv \textit{if } j \geq 0 \textit{ then } \mathrm{port}^{bw}_j(u(r(p))) \textit{ else } \mathrm{port}^{fw}_{-1}(u(r(p)))$$

We write $t \sim^c u$ if there exists a mapping r such that $t \sim^c_r u$. For instance, the two canonical trees in Figure 7.3 are related by a canonical rotation (depicted in red).

The following lemma is the key for proving completeness of our entailment checking for local inductive systems: if a (local) state is a model of the characteristic formulae of two different canonical trees, then these trees must be related by canonical rotation.

Lemma 9. *Let $t : \mathbb{N}^* \rightharpoonup_{fin} \mathcal{T}$ be a canonical tree and $S = \langle s, h \rangle$ be a state such that $S \models \Phi(t)$. Then, for any canonical tree $u : \mathbb{N}^* \rightharpoonup_{fin} \mathcal{T}$, we have $S \models \Phi(u)$ iff $t \sim^c u$.*

Given a TA $A = \langle Q, \mathcal{T}, \Delta, F \rangle$ recognizing a set of canonical trees, we build a TA A_{rot} such that $\mathcal{L}(A_{\text{rot}}) = \{u \mid \exists t \in \mathcal{L}(A) \ . \ u \sim^c t\}$. We define A_{rot} as a union $\bigcup_{\rho \in \Delta} A_\rho$, where A_ρ is a TA that depends on the choice of a particular transition rule ρ of A. The time needed to compute each A_ρ is, moreover, linear in the size of A. It follows that the time needed to compute A_{rot} is quadratic in the size of A, thus its size is quadratic in the size of A, as well.

The idea behind the construction of $A_\rho = \langle Q_\rho, \Sigma, \Delta_\rho, \{q_\rho^f\} \rangle$ can be understood by considering a tree $t \in \mathcal{L}(A)$, a run $\pi : dom(t) \to Q$, and a position $p \in dom(t)$, which is labeled with the right hand side of the rule $\rho = T(q_1, \ldots, q_k) \to q$ of A. Then $\mathcal{L}(A_\rho)$ will contain the rotated tree u, i.e. $t \sim_r^c u$, where the significant position p is mapped into the root of u by the rotation function r, i.e. $r(p) = \epsilon$. To this end, we introduce a new rule $T_{new}(q_0, \ldots, q^{rev}, \ldots, q_k) \to q_\rho^f$ where the tile T_{new} mirrors the change in the structure of T at position p, and $q^{rev} \in Q_\rho$ is a fresh state corresponding to q. The construction of A_ρ continues recursively, by considering every rule of A that has q on the left hand side: $U(q'_1, \ldots, q, \ldots, q'_\ell) \to s$. This rule is changed by swapping the roles of q and s and producing a rule $U_{new}(q'_1, \ldots, s^{rev}, \ldots q'_\ell) \to q^{rev}$ where U_{new} mirrors the change in the structure of U. Intuitively, the states $\{q^{rev} \mid q \in Q\}$ mark the unique path from the root of u to $r(\epsilon) \in dom(u)$. The recursion stops when either (i) s is a final state of A, (ii) the tile U does not specify a forward edge in the direction marked by q, or (iii) all states of A have been visited.

We have thus reduced an entailment problem $\mathsf{P}_i \models_{\mathsf{SL}} \mathsf{P}_j$ in a local inductive system $\mathcal{P}$ to the tree language inclusion problem $\mathcal{L}(A^i_{\mathcal{P}}) \subseteq \mathcal{L}(A^j_{\mathcal{P}})_{\text{rot}}$. Because both TA can be built in time polynomial in the size of $\mathcal{P}$ and, moreover, the language inclusion problem for TA is EXPTIME-complete, the entailment problem is in EXPTIME. We obtain a matching lower bound by reduction from the universality problem for TA, which is, again, EXPTIME-complete.

Theorem 25. *The entailment problem is* EXPTIME*-complete for* SL_{loc}.

Proof. See [IRV14b, Theorem 3]. □

7.3 Discussion and Open Problems

The decidability of entailments for the $\mathsf{SL}_{\mathrm{btw}}$ fragment is currently the most general result for SL with inductive definitions. Moreover, the restriction of $\mathsf{SL}_{\mathrm{btw}}$ to local inductive systems provides a practical and complete algorithm for deciding entailments, which has been implemented in a prototype tool (Slide [IRV]). This tool has been compared to other inductive solvers for SL during SL-COMP'14 [SC14], an event collocated with the competition of Satisfiability Modulo Theories (SMT'14).

The comparison with other inductive provers (Sleek [NC08], Cyclist [Gor], Spen [ELS+]) outlined several strengths and weaknesses of automata-based methods. On the positive side, the tree automata encoding using rotations provides a complete method that can decide entailments such as $\mathsf{dll}(\mathsf{hd},\mathsf{n},\mathsf{tl},\mathsf{p}) \models_{\mathsf{SL}} \mathsf{dll}_{\mathrm{rev}}(\mathsf{hd},\mathsf{n},\mathsf{tl},\mathsf{p})$, which are beyond the capabilities of other solvers. On the negative side, the automata encoding is very poor in dealing with disequalities between variables (needed, for instance, to define acyclic lists) or with extensions of SL that allow reasoning about the data in the cells. Such extensions require the development of new inclusion techniques for automata over infinite alphabets, or, more generally, automata extended with variables ranging over infinite domains. To this end, an initial approach was to develop a counterexample-driven abstraction refinement semi-algorithm for the inclusion of counter machines [IRV14a]. Extending this procedure to tree automata with integer variables is considered as future work.

Another direction for future work is analyzing the relation between cyclic induction proofs and antichain-based language inclusion of tree automata. On one hand, proof search is promising in providing sound and efficient decision procedures. On the other hand, automata-based methods provide complete decision procedures. A closer analysis of the relation between inductive proof search and language inclusion between automata can be the key to obtaining both fast and complete proof systems.

Chapter 8

Conclusions and Perspectives

This thesis presents several contributions to program verification, spanning the period between 2006 and 2016. The main ingredients of program verification are logics and automata theory, in a broad sense. The contributions presented in this thesis belong to both the areas of logic and (extended) automata theory, highlighting several connections between them.

Regarding logic, we found novel decidable fragments of first-order arithmetic, interpreted over integers [BI05, BIL06, BIL09] (Chapter 3) and developed specific logics for reasoning about data structures, commonly found in programs, such as arrays with integer values [HIV08b, HIV08a] (Chapter 6, Section 6.3) or dynamically allocated mutable data structures [IRS13, IRV14b] (Chapter 7).

Concerning automata models, we studied decision problems (reachability and termination) for several classes of counter machines, i.e. automata extended with integer variables [BIL06, BIL09, BGI09, BIK10, BIK14b, BIK13] (Chapter 4). We found that, despite the expressive power of the model, the decision problems of restricted classes, such as the one defined by *flatness* (absence of nested cycles in the control structure) have rather good complexity bounds (polynomial or NP-complete) in many non-trivial cases. This result motivates the use of flat counter machines to defining underapproximations of programs, in order to devise efficient bug-finding semi-algorithms [KIB09].

Furthermore, we studied recursive counter machines [GIK13, GIK12, GI15a] (Chapter 5). We nailed down a parameter of the analysis, called the *index*, such that, when fixing this parameter to a constant, the recursive counter machine can be verified by looking at an equivalent, non-recursive, counter machine. Moreover, when the set of interprocedurally valid paths

belongs to a bounded pattern, the reachability problems are in NEXPTIME, and become NP-complete, when the index is fixed. Defining bounded underapproximations of the set of interprocedurally valid paths of a recursive counter machine is a generalisation of the method of defining flat underapproximations for standard counter machines, leading to efficient bug-finding semi-algorithms [KIB09].

Focusing on more realistic program models, we considered programs with dynamic memory and linked recursive data structures, as lists, trees and beyond [BBH+06, HIRV07, IR09, IR13] (Chapter 6). A first approach to program verification consists in modeling programs with singly-linked lists as counter machines. This translation leverages existing analyses for counter machines, such as abstract interpretation with polyhedra [GM12], interpolant-based predicate abstraction [McM06] or inference of ranking functions for termination analysis [BMS05, CPR06]. The method is sound and complete for programs with lists [BIP] and sound for programs with trees, i.e. it can prove termination but cannot always detect non-termination.

Another interesting application of counter machines in program verification is deciding the validity of entailments in universally quantified array logics, by reduction to reachability problems of flat counter machines [HIV08b, HIV08a] (Chapter 6, Section 6.3). Under several restrictions, the translation of logical formulae into counter machines can be made inductively on the structure of the formulae, as in the case of the classical translation of MSO formulae into finite automata [HIV08a]. In general, the nonlocal constraints introduced by the interaction between several universally quantified index variables make the translation more difficult [HIV08b]. Program analysis using these logics involves defining non-trivial post-condition calculi and invariant generation techniques [BHI+09].

The key for the scalability of program analyses is compositionality. For programs with dynamic data structures and low-level pointer manipulations, Separation Logic (SL) has become a mainstream framework for designing modular analyses, based on the principle of local reasoning [IO01, Rey02]. However, the decidability and computation complexity of SL pose non-trivial problems [CYO01]. In this respect, we identified a general decidable fragment of SL [IRS13] and developed a practical SL solver [IRV], based on reduction of entailment problems to language inclusion problems for tree automata [IRV14b] (Chapter 7).

Future Work We plan to address most of the open problems mentioned in this thesis, such as the complexity of reachability and termination problems

for flat counter machines with finite monoid affine relations [Boi99, FL02b], or extensions of the counter machine model to concurrent and branching executions [DJLL13].

Furthermore, developing efficient language inclusion procedures for (tree) automata extended with integer variables and transitions harnessed by arithmetic constraints seems to be important in extending the capabilities of inductive solvers for SL. In this respect, we plan of developing efficient semi-algorithms based on simulations, antichains and counterexample-driven abstraction refinement with interpolants [IRV14a].

In a different vein, defining language inclusion between tree automata (possibly extended with integer variables) as proof search in a deductive system could provide entailment provers with the ability of generating proof certificates for valid entailments. This requires defining a formal relation between antichain-based language inclusion algorithms [ACH+10] and deductive systems based on cyclic proofs [Acz77].

Finally, we plan to address the verification of concurrent and distributed programs with abstract data structures (containers) handled via high-level logical specification contracts. These container libraries, encountered in most modern programming languages, give software developers a way of abstracting from low-level implementation details related to memory management, such as dynamic allocation and pointer handling. However, the implementations of these libraries use optimized low-level data structures and algorithms for heap memory management, e.g. skip lists, red-back trees, or overlaid linked lists. To ensure correctness of software systems manipulating complex data structures, two essential issues must be considered: (1) the correctness of programs implementing applications, assuming high level specifications of the methods provided by the external container libraries they use, and (2) the correctness of the implementations of such libraries with respect to their abstract specifications, the fact that a specific implementation of containers ensures the expected abstract behavior.

Bibliography

[Abr96] J-R. Abrial. *The B-Book: Assigning Programs to Meanings,*. Cambridge University Press, 1996.

[ACH+10] P.A. Abdulla, Y.-F. Chen, L. Holik, R. Mayr, and T. Vojnar. When simulation meets antichains. In *Proc. of TACAS'10*, volume 6015 of *LNCS*, pages 158–174. Springer, 2010.

[Acz77] P. Aczel. An introduction to inductive definitions*. In *HANDBOOK OF MATHEMATICAL LOGIC*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, pages 739 – 782. Elsevier, 1977.

[AM09] R. Alur and P. Madhusudan. Adding nesting structure to words. *JACM*, 56(3):16, 2009.

[APR+01] T. Arons, A. Pnueli, S. Ruah, Y. Xu, and L. Zuck. *Parameterized Verification with Automatically Computed Inductive Assertions*, pages 221–234. Springer Berlin Heidelberg, 2001.

[B$\ddot{\text{o}}$2] J. R. Büchi. On a Decision Method in Restricted Second-Order Arithmetic. In *International Congress on Logic, Methodology, and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.

[B$\acute{\text{o}}$2] A. Bés. A survey of arithmetical definability. *A Tribute to Maurice Boffa. Bulletin de la Société Mathématique de Belgique*, 1 - 54, 2002.

[Bar03] J. Barnes. *High Integrity Software: The SPARK Approach to Safety and Security.* Addison-Wesley, 2003.

[BBH+06] A. Bouajjani, M. Bozga, P. Habermehl, R. Iosif, P. Moro, and T. Vojnar. Programs with lists are counter automata. In *CAV*, volume 4144 of *LNCS*, pages 517–531, 2006.

[BBH+11] A. Bouajjani, M. Bozga, P. Habermehl, R. Iosif, P. Moro, and T. Vojnar. Programs with lists are counter automata. *Formal Methods in System Design*, 38(2):158–192, 2011.

[BCC+03] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *PLDI*, pages 196–207. ACM, 2003.

[BCC+07] J. Berdine, C. Calcagno, B. Cook, D. Distefano, P.W. O'Hearn, T. Wies, and H. Yang. Shape analysis for composite data structures. In *Proc. CAV'07*, volume 4590 of *LNCS*. Springer, 2007.

[BDM+11] M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Logic*, 12(4):27:1–27:26, 2011.

[Bel76] A. P. Beltyukov. Decidability of the universal theory of natural numbers with addition and divisibility. *Zapiski Nauch. Sem. Leningrad Otdeleniya Mathematical Institute*, 60:15 – 28, 1976.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR '97: Concurrency Theory, 8th International Conference, Warsaw, Poland, July 1-4, 1997, Proceedings*, pages 135–150, 1997.

[BFG+14] M. Blondin, A. Finkel, S. Göller, C. Haase, and P. McKenzie. Reachability in two-dimensional vector addition systems with states is pspace-complete. *CoRR*, abs/1412.4259, 2014.

[BFL+11] M. Barnett, M. Fähndrich, K. R. M. Leino, P. Müller, W. Schulte, and H. Venter. Specification and verification: the spec# experience. *Commun. ACM*, 54(6):81–91, 2011.

[BFP] S. Bardin, A. Finkel, and J. Leroux L. Petrucci. Fast: Fast acceleration of symbolic transition systems. http://tapas.labri.fr/trac/wiki/FASTer.

[BGI09] M. Bozga, C. Gîrlea, and R. Iosif. Iterating octagons. In *Proc. of TACAS*, volume 5505 of *LNCS*, pages 337–351. Springer Verlag, 2009.

[BHI+09] M. Bozga, P. Habermehl, R. Iosif, F. Konečný, and T. Vojnar. Automatic verification of integer array programs. In *CAV*, volume 5643 of *LNCS*, pages 157–172, 2009.

[BHMV94] V. Bruyère, G. Hansel, C. Michaux, and R. Villemaire. Logic and p-recognizable sets of integers. *Bull. Belg. Math. Soc*, 1:191–238, 1994.

[BHRV06] A. Bouajjani, P. Habermehl, A. Rogalewicz, and T. Vojnar. *Abstract Regular Tree Model Checking of Complex Dynamic Data Structures*, pages 52–70. Springer Berlin Heidelberg, 2006.

[BHZ08] R. Bagnara, P. M. Hill, and E. Zaffanella. An improved tight closure algorithm for integer octagonal constraints. In *Proc. of VMCAI*, volume 4905 of *LNCS*, pages 8–21. Springer Verlag, 2008.

[BI05] M. Bozga and R. Iosif. On decidability within the arithmetic of addition and divisibility. In *Foundations of Software Science and Computational Structures, 8th International Conference, FOSSACS 2005*, pages 425–439, 2005.

[BI07] M. Bozga and R. Iosif. On flat programs with lists. In *Verification, Model Checking, and Abstract Interpretation, 8th International Conference, VMCAI 2007, Nice, France, January 14-16, 2007, Proceedings*, pages 122–136, 2007.

[BIK10] M. Bozga, R. Iosif, and F. Konečný. Fast acceleration of ultimately periodic relations. In *CAV*, volume 6174 of *LNCS*, pages 227–242, 2010.

[BIK12] M. Bozga, R. Iosif, and F. Konečný. Deciding conditional termination. In *Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'12, pages 252–266. Springer-Verlag, 2012.

[BIK13] M. Bozga, R. Iosif, and F. Konecný. The complexity of reachability problems for flat counter machines with periodic loops. *CoRR*, abs/1307.5321, 2013.

[BIK14a] M. Bozga, R. Iosif, and F. Konecný. Deciding conditional termination. *Logical Methods in Computer Science*, 10(3), 2014.

[BIK14b] M. Bozga, R. Iosif, and F. Konecný. Safety problems are np-complete for flat integer programs with octagonal loops. In *Verification, Model Checking, and Abstract Interpretation - 15th International Conference, VMCAI 2014, San Diego, CA, USA, January 19-21, 2014, Proceedings*, pages 242–261, 2014.

[BIL06] M. Bozga, R. Iosif, and Y. Lakhnech. *Flat Parametric Counter Automata*, pages 577–588. Springer Berlin Heidelberg, 2006.

[BIL09] M. Bozga, R. Iosif, and Y. Lakhnech. Flat parametric counter automata. *Fundamenta Informaticae*, 91(2):275–303, 2009.

[BIP] M. Bozga, R. Iosif, and S. Perarnau. L2CA: Lists to Counter Automata. http://www-verimag.imag.fr/L2CA-homepage.html.

[BJNT00] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *CAV*, volume 1855 of *LNCS*, pages 403–418. Springer, 2000.

[BJS07] A. Bouajjani, Y. Jurski, and M. Sighireanu. *A Generic Framework for Reasoning About Dynamic Networks of Infinite-State Processes*, pages 690–705. Springer Berlin Heidelberg, 2007.

[BMS05] A. R. Bradley, Z. Manna, and H. B. Sipma. *The Polyranking Principle*, pages 1349–1361. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[BMS06] A. R. Bradley, Z. Manna, and H. B. Sipma. *What's Decidable About Arrays?*, pages 427–442. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[Boi99] B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces*. PhD, Univ. de Liège, 1999.

[Boj08] M. Bojańczyk. *Tree-Walking Automata*, pages 1–2. Springer Berlin Heidelberg, 2008.

[CC79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *POPL*, pages 269–282. ACM, 1979.

[CC05] H. Comon and V. Cortier. Tree automata with one memory set constraints and cryptographic protocols. *Theor. Comput. Sci.*, 331(1):143–214, February 2005.

[CCF+07] P. Cousot, R. Cousot, J. Feret, A. Miné, L. Mauborgne, D. Monniaux, and X. Rival. Varieties of static analyzers: A comparison with ASTRÉE. In *TASE*. IEEE, 2007.

[CD11] C. Calcagno and D. Distefano. Infer: An automatic program verifier for memory safety of c programs. In *NASA FM*, volume 6617 of *LNCS*, pages 459–465. Springer, 2011.

[CDG+05] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications. http://www.grappa.univ-lille3.fr/tata, 2005.

[CE69] R. B. Crittenden and C. L. Vanden Eynden. A proof of a conjecture of Erdös. *Bulletin of American Mathematical Society*, 75(6):1326 – 1329, 1969.

[CGJ+00] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. *Counterexample-Guided Abstraction Refinement*, pages 154–169. Springer Berlin Heidelberg, 2000.

[Chu32] A. Church. A set of postulates for the foundation of logic. *Annals of Mathematics*, 33(2):346–366, 1932.

[Chu36] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345 – 363, 1936.

[CJ98] H. Comon and Y. Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In *CAV*, volume 1427 of *LNCS*, pages 268–279, 1998.

[CKK+12] P. Cuoq, F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles, and B. Yakobowski. Frama-C - a software analysis perspective. In *SEFM*, volume 7504 of *LNCS*, pages 233–247, 2012.

[Cou90] B. Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12 – 75, 1990.

[Cow00] J. Cowles. Knuth's generalization of mccarthy's 91 function. In *Computer-Aided reasoning: ACL2 case studies*, pages 283–299. Kluwer Academic Publishers, 2000.

[CPR06] B. Cook, A. Podelski, and A. Rybalchenko. Terminator: Beyond Safety. In *Proc. of CAV 2006*, volume 4144 of *LNCS*. Springer, 2006.

[CYO01] C. Calcagno, H. Yang, and P. W. OHearn. Computability and complexity results for a spatial assertion language for data structures. In *FST TCS 2001: Foundations of Software Technology and Theoretical Computer Science*, pages 108–119. Springer Berlin Heidelberg, 2001.

[DA14] L. D'Antoni and R. Alur. Symbolic visibly pushdown automata. In *Proc. of CAV'14*, volume 8559 of *LNCS*. Springer, 2014.

[DDS12] S. Demri, A.K. Dhar, and A. Sangnier. Taming past LTL and flat counter systems. In *IJCAR*, volume 7364, pages 179–193, 2012.

[DJLL13] S. Demri, M. Jurdzinski, O. Lachish, and R. Lazic. The covering and boundedness problems for branching vector addition systems. *J. Comput. Syst. Sci.*, 79(1):23–38, 2013.

[EG11] J. Esparza and P. Ganty. Complexity of pattern-based verification for multithreaded programs. In *POPL '11*, pages 499–510. ACM Press, 2011.

[EKL10] J. Esparza, S. Kiefer, and M. Luttenberger. Newtonian program analysis. *JACM*, 57(6):33:1–33:47, 2010.

[ELS+] C. Enea, O. Lengal, M. Sighireanu, T. Vojnar, and Z. Wu. Spen: Solver for separation logic entailments. http://www.liafa.univ-paris-diderot.fr/spen/.

[FL02a] A. Finkel and J. Leroux. How to compose presburger-accelerations: Applications to broadcast protocols. In *FST TCS '02*, pages 145–156. Springer, 2002.

[FL02b] A. Finkel and J. Leroux. How to compose presburger-accelerations: Applications to broadcast protocols. In *Proc. of FST TCS*, volume 2556 of *LNCS*, pages 145–156. Springer Verlag, 2002.

[Flo67] R. W. Floyd. Assigning meanings to programs. In *Proceedings of Symposia in Applied Mathematics*, volume 19, pages 19–32, 1967.

[FM07] J.-C. Filliâtre and C. Marché. The Why/Krakatoa/Caduceus platform for deductive program verification. In *CAV*, pages 173–177, 2007.

[FS08] A. Finkel and A. Sangnier. Reversal-bounded counter machines revisited. In *Proceedings of the 33rd International Symposium on Mathematical Foundations of Computer Science (MFCS'08)*, volume 5162 of *Lecture Notes in Computer Science*, pages 323–334. Springer, 2008.

[G$\ddot{3}$1] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38:173 – 198, 1931.

[GH96] S. Graf and H.Saïdi. Verifying invariants using theorem proving. In *CAV*, volume 1102 of *LNCS*, pages 196–207. Springer, 1996.

[GI81] E. M. Gurari and O. H. Ibarra. The complexity of decision problems for finite-turn multicounter machines. *J. Computer and System Sciences*, 22:220–229, 1981.

[GI15a] P. Ganty and R. Iosif. Interprocedural reachability for flat integer programs. In *Fundamentals of Computation Theory - 20th International Symposium, FCT 2015, Gdańsk, Poland, August 17-19, 2015, Proceedings*, pages 133–145, 2015.

[GI15b] P. Ganty and R. Iosif. Interprocedural reachability for flat integer programs. *CoRR*, abs/1405.3069v3, 2015.

[GIK12] P. Ganty, R. Iosif, and F. Konecný. Underapproximation of procedure summaries for integer programs. *CoRR*, abs/1210.4289, 2012.

[GIK13] P. Ganty, R. Iosif, and F. Konecný. Underapproximation of procedure summaries for integer programs. In *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013*, pages 245–259, 2013.

[Gin66] S. Ginsburg. *The Mathematical Theory of Context-Free Languages*. McGraw-Hill, Inc., New York, NY, USA, 1966.

[GM12] T. M. Gawlitza and D. Monniaux. Invariant generation through strategy iteration in succinctly represented control flow graphs. *Logical Methods in Computer Science*, 8(3), 2012.

[Gon] L. Gonnord. Aspic: Accelerated symbolic polyhedral invariant computation. http://laure.gonnord.org/pro/aspic/aspic.html.

[Gor] N. Gorogiannis. Cyclist: A cyclic theorem prover framework. https://github.com/ngorogiannis/cyclist.

[GS64] S. Ginsburg and E. H. Spanier. Bounded algol-like languages. *Transactions of the American Mathematical Society*, 113(2):333–368, 1964.

[GS66] S. Ginsburg and E. H. Spanier. Semigroups, presburger formulas and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966.

[HA28] D. Hilbert and W. Ackermann. *Grundzüge der theoretischen Logik (Principles of Mathematical Logic)*. Springer-Verlag, 1928.

[Haa14] C. Haase. Subclasses of presburger arithmetic and the weak exp hierarchy. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, CSL-LICS '14, pages 47:1–47:10. ACM, 2014.

[HHR+] P. Habermehl, L. Holik, A. Rogalewicz, J. Simacek, T. Vojnar, and O. Lengal. Forester: a tool for the verification of programs with pointers. http://www.fit.vutbr.cz/research/groups/verifit/tools/forester/.

[HIK+12] H. Hojjat, R. Iosif, F. Konečný, V. Kuncak, and P. Rümmer. *Accelerating Interpolants*, pages 187–202. Springer Berlin Heidelberg, 2012.

[HIRV07] P. Habermehl, R. Iosif, A. Rogalewicz, and T. Vojnar. Proving termination of tree manipulating programs. In *Automated Technology for Verification and Analysis, 5th International Symposium, ATVA 2007, Tokyo, Japan, October 22-25, 2007, Proceedings*, pages 145–161, 2007.

[HIV06] P. Habermehl, R. Iosif, and T. Vojnar. Automata-based verification of programs with tree updates. In *Tools and Algorithms for the Construction and Analysis of Systems, 12th International Conference, TACAS 2006*, pages 350–364, 2006.

[HIV08a] P. Habermehl, R. Iosif, and T. Vojnar. A logic of singly indexed arrays. In *Logic for Programming, Artificial Intelligence, and*

Reasoning, 15th International Conference, LPAR 2008, Doha, Qatar, November 22-27, 2008. Proceedings, pages 558–573, 2008.

[HIV08b] P. Habermehl, R. Iosif, and T. Vojnar. What else is decidable about integer arrays? In *Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS 2008*, pages 474–489, 2008.

[HIV10] P. Habermehl, R. Iosif, and T. Vojnar. Automata-based verification of programs with tree updates. *Acta Inf.*, 47(1):1–31, 2010.

[HKG+12] H. Hojjat, F. Konečný, F. Garnier, R. Iosif, V. Kuncak, and P. Rümmer. *A Verification Toolkit for Numerical Transition Systems*, pages 247–251. Springer Berlin Heidelberg, 2012.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, October 1969.

[HP79] J. Hopcroft and J-J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8(2):135 – 159, 1979.

[HR] H. Hojjat and P. Rümmer. Eldarica: a predicate abstraction engine. http://lara.epfl.ch/w/eldarica.

[Iba78] O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM*, 25(1):116–133, 1978.

[IKB] R. Iosif, F. Konecny, and M. Bozga. Numerical transition systems. http://nts.imag.fr/index.php/Main_Page.

[IO01] S. S. Ishtiaq and P. W. O'Hearn. Bi as an assertion language for mutable data structures. *SIGPLAN Not.*, 36(3):14–26, January 2001.

[IR09] R. Iosif and A. Rogalewicz. Automata-based termination proofs. In *Implementation and Application of Automata, 14th International Conference, CIAA 2009, Sydney, Australia, July 14-17, 2009. Proceedings*, pages 165–177, 2009.

[IR13] R. Iosif and A. Rogalewicz. Automata-based termination proofs. *Computing and Informatics*, 32(4):739–775, 2013.

[IRS13] R. Iosif, A. Rogalewicz, and J. Simácek. The tree width of separation logic with recursive definitions. In *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, pages 21–38, 2013.

[IRV] R. Iosif, A. Rogalewicz, and T. Vojnar. Slide: Separation logic with inductive definitions. http://www.fit.vutbr.cz/research/groups/verifit/tools/slide/.

[IRV14a] R. Iosif, A. Rogalewicz, and T. Vojnar. Abstraction refinement for trace inclusion of data automata. *CoRR*, abs/1410.5056, 2014.

[IRV14b] R. Iosif, A. Rogalewicz, and T. Vojnar. Deciding entailments in inductive separation logic with tree automata. In *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings*, pages 201–218, 2014.

[KIB09] F. Konecny, R. Iosif, and M. Bozga. Flata: a verification toolset for counter machines. http://nts.imag.fr/index.php/Flata, 2009.

[KLW13] D. Kroening, M. Lewis, and G. Weissenbacher. Under-approximating loops in C programs for fast counterexample detection. In *CAV '13*, LNCS, pages 381–396. Springer, 2013.

[Kon14] F. Konecný. PTIME computation of transitive closures of octagonal relations. *CoRR*, abs/1402.2102, 2014.

[Kos82] S. R. Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 267–281. ACM, 1982.

[LAJ] G. Lalire, M. Argoud, and B. Jeannet. Interproc. http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/interproc/index.html.

[Ler12] J. Leroux. Vector Addition Systems Reachability Problem (A Simpler Solution). In Andrei Voronkov, editor, *The Alan Turing Centenary Conference*, volume 10 of *EPiC Series*, pages 214–228. Andrei Voronkov, 2012.

[Lip76a] L. Lipshitz. The diophantine problem for addition and divisibility. *Transaction of the American Mathematical Society*, 235:271 – 283, January 1976.

[Lip76b] R. J. Lipton. The reachability problem is exponential-space-hard. Technical Report 62, Yale University, Department of Computer Science, 1976.

[LOW15] Antonia Lechner, Jol Ouaknine, and James Worrell. On the Complexity of Linear Arithmetic with Divisibility. In *Proceedings of the 30th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 667–676. IEEE, 2015.

[LS04] J. Leroux and G. Sutre. *On Flatness for 2-Dimensional Vector Addition Systems with States*, pages 402–416. Springer Berlin Heidelberg, 2004.

[Luk78] M. Luker. A family of languages having only finite-index grammars. *Information and Control*, 39(1):14–18, 1978.

[Luk80] M. Luker. Control sets on grammars using depth-first derivations. *Math. Systems Theory*, 13:349–359, 1980.

[Mat70] Y. Matiyasevich. Enumerable sets are diophantine. *Journal of Sovietic Mathematics*, 11:354 – 358, 1970.

[May81] E. W. Mayr. An algorithm for the general petri net reachability problem. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, pages 238–246. ACM, 1981.

[McM06] K. L. McMillan. Lazy abstraction with interpolants. In *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, pages 123–136, 2006.

[Min67] M. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

[Min06] A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.

[Mos52] A. Mostowski. On direct products of theories. *The Journal of Symbolic Logic*, 17(1):1–31, 1952.

[Nai82] M. Nair. A new method in elementary prime number theory. *Journal of the London Mathematical Society*, s2-25(3):385–391, 1982.

[NC08] H. H. Nguyen and W.-N. Chin. Enhancing program verification with lemmas. In *Proc of CAV'08*, volume 5123 of *LNCS*. Springer, 2008.

[OP99] P. W. O'Hearn and D.J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5:215–244, 6 1999.

[OW12] J. Ouaknine and J. Worrell. Decision problems for linear recurrence sequences. In *Reachability Problems - 6th International Workshop, RP 2012, Bordeaux, France, September 17-19, 2012. Proceedings*, pages 21–28, 2012.

[Pot90] L. Pottier. Solutions minimales des systemes diophantiens lineaires : bornes et algorithmes. Research Report RR-1292, INRIA, 1990.

[Pre29] M. Presburger. Über die Vollstandigkeit eines gewissen Systems der Arithmetik. *Comptes rendus du I Congrés des Pays Slaves*, Warsaw 1929.

[Rab68] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Bull. Amer. Math. Soc.*, 74(5):1025–1029, 09 1968.

[Rac78] C. Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223 – 231, 1978.

[Rey02] J.C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *Proc. of LICS'02*. IEEE CS Press, 2002.

[RHS95] T. Reps, S. Horwitz, and M. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *POPL '95*, pages 49–61. ACM, 1995.

[Rob49] J. Robinson. Definability and decision problems in arithmetic. *The Journal of Symbolic Logic*, 14(2):98 – 114, June 1949.

[RV] A. Rogalewicz and T. Vojnar. Artmc: Abstract regular tree model checking. http://www.fit.vutbr.cz/research/groups/verifit/tools/artmc/.

[SC14] M. Sighireanu and D. Cok. Report on sl-comp'14. *JSAT*, 9:173–186, 2014.

[Sch61] M.P. Schutzenberger. On the definition of a family of automata. *Information and Control*, 4(23):245–270, 1961.

[Sch00] B. De Schutter. On the ultimate behavior of the sequence of consecutive pow- ers of a matrix in the max-plus algebra. *Linear Algebra and Its Applications*, 307(1-3):103–117, 2000.

[See91] D. Seese. The structure of models of decidable monadic theories of graphs. *Annals of Pure and Applied Logic*, 53(2):169–195, 1991.

[Sem79] A. L. Semenov. On certain extensions of the arithmetic of addition of natural numbers. *Izv. Akad. Nauk SSSR Ser. Mat.*, 43:1175–1195, 1979.

[SP81] M. Sharir and A. Pnueli. Two approaches to interprocedural data flow analysis. In *Program Flow Analysis: Theory and Applications*, chapter 7, pages 189–233. Prentice-Hall, Inc., 1981.

[SRW02] M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. *ACM Trans. on Prog. Languages and Systems*, 24(3), 2002.

[Sto74] L. J. Stockmeyer. *The complexity of decision problems in automata theory and logic*. PhD thesis, Massachusetts Institute of Technology. Dept. of Electrical Engineering, 1974. https://dspace.mit.edu/handle/1721.1/15540.

[Tur37] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937.

[VSS05] K. N. Verma, H. Seidl, and T. Schwentick. On the complexity of equational horn clauses. In *CADE '05*, volume 1831 of *LNCS*, pages 337–352, 2005.