Relational Verification of Programs with Integer Data

Filip Konečný*
Department of Intelligent Systems
Faculty of Information Technology
Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech republic
ikonecny@fit.vutbr.cz

Abstract

This work presents novel methods for verification of reachability and termination properties of programs that manipulate unbounded integer data. Most of these methods are based on acceleration techniques which compute transitive closures of program loops.

Firtly, we present an algorithm that accelerates several classes of integer relations and show that the new method performs up to four orders of magnitude better than the previous ones. On the theoretical side, our framework provides a common solution to the acceleration problem by proving that the considered classes of relations are periodic.

Subsequently, we introduce a semi-algorithmic reachability analysis technique that tracks relations between variables of integer programs and applies the proposed acceleration algorithm to compute summaries of procedures in a modular way. Next, we present an alternative approach to reachability analysis that integrates predicate abstraction with our acceleration techniques to increase the likelihood of convergence of the algorithm. We evaluate these algorithms and show that they can handle a number of complex integer programs where previous approaches failed.

Finally, we study the termination problem for several classes of program loops and show that it is decidable. Moreover, for some of these classes, we design a polynomial time algorithm that computes the exact set of program configurations from which non-terminating runs exist. We further integrate this algorithm into a semi-algorithmic method that analyzes termination of integer programs, and show that the resulting technique can verify termination properties of several non-trivial integer programs.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Software/Program Verification—*formal methods, model checking*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*invariants, assertions*

Keywords

programs with integers, counter automata, reachability analysis, termination analysis, acceleration, transitive closures, procedure summaries, recurrent sets, termination preconditions

1. Introduction

Formal verification is the task of analyzing *correctness* of a computer system with respect to a given *specification*. A majority of formal verification methods is *sound*, meaning that when a method finds no counterexample to the specification, then the inspected system is guaranteed to be correct. This is in contrast with traditional testing since the fact that no error manifests in a given set of test cases does not imply correctness of the system under inspection. Therefore, formal verification can be seen as a technique that is complementary to traditional testing. The experience of the computer systems industry in the last two decades shows that both techniques are needed to ensure reliability of computer systems. This need arises from increasing complexity and pervasiveness of computer systems. Indeed, more and more sophisticated software systems of growing sizes run on an increasingly complex hardware. Moreover, many computer systems are safety-critical since they operate aircraft, medical devices, nuclear systems, etc., and their malfunction may have serious consequences. Both hardware manufacturers and software developers aim to deliver reliable systems. Hence the growing need for more sophisticated formal verification methods that can cope with the increasing complexity of computer systems.

Model checking [29, 64] is one of the main branches of formal verification. It is an approach of automated checking whether a *model* of a computer system (where the model can sometimes be identical to the system) satisfies a certain correctness specification by systematically exploring the state space of the system being verified. Model checking can be usually fully automated and moreover, in cases when a model violates a given specification,

*Recommended by thesis supervisor: Prof. Tomáš Vojnar. Defended at Faculty of Information Technology, Brno University of Technology on October 29, 2012.

Konečný, F. Relational Verification of Programs with Integer Data. Information Sciences and Technologies Bulletin of the ACM Slovakia, Vol. 5, No. 1 (2013) 13-24

a model checking method can provide a counter-example which demonstrates the cause of the violation. Traditionally, model checking has been applied to systems with a *finite state space*, especially in hardware. A major success has been achieved when explicit representation of states has been replaced with a *symbolic* one [27]. This is because a symbolic state represents a potentially large set of concrete states which can then be manipulated simultaneously. More recently, model checking has been extended to deal with *infinite* state spaces which arise, e.g., in software that manipulates dynamic data structures or integers. In certain cases, infinite state verification can be reduced to finite state verification by applying e.g. cut-off techniques [15]. When such reduction is not possible, a model checking method needs to use a symbolic representation for possibly infinite sets of states that need to be explored. Many such representations have been proposed including those based on various kinds of *automata* and *logics* [16, 14]. Since an automaton recognizes a potentially infinite set of traces and a logical formula may have a potentially infinite set of models, they can be used to encode potentially infinite sets of states.

In our work, we address model checking of sequential non-recursive programs with *unbounded integer data* (also known as counter automata, counter systems, or counter machines). The interest for integer programs comes from the fact that they can encode various classes of systems with unbounded (or very large) data domains, such as hardware circuits, cache memories, or software systems with variables of non-primitive types, such as integer arrays, pointers and/or recursive data structures. This comes with no surprise since, in theory, any Turing-complete class of systems can be simulated by integer programs, as shown by Minsky [58]. For practical purposes, however, a number of recent works have revealed cost-effective reductions of verification problems for several classes of complex systems to decision problems phrased in terms of integer programs. Examples of such systems that can be effectively verified by means of integer programs, include: specifications of hardware components [68], programs with singly-linked lists [12, 39, 19, 23], trees [45], and integer arrays [46, 44, 18]. Hence the growing interest for analysis tools working on integer programs.

Our work contributes to the field of model checking of programs with integer data by developing methods for reachability and termination analysis. Given an integer program, a set of initial states, and a set of error states, the *reachability problem* asks whether the program has a computation starting in a state from the initial set which leads to a state from the error set. The *termination problem* asks whether each computation of a program that starts in a state from the initial set eventually halts. A slightly more general problem is the *conditional termination problem* which asks to compute the set of initial program states from which every computation halts.

2. State of the Art

Since the result of Minsky [58], proving Turing-completeness of 2-counter machines with increment, decrement and zero test, research on analysis and verification of integer programs has been pursued in two orthogonal directions. The first one is defining subclasses of systems for which various decision problems, such as reachability or termination, are found to be decidable. Based on these results, model checking algorithms that perform precise analysis have been proposed. The second direction is concerned with finding sound (but not necessarily complete) answers to decision problems in a cost-effective way, by abstracting the system under inspection.

2.1 Decidable Classes of Integer Programs

Examples of classes of integer programs where the reachability problem is decidable include *reversal-bounded counter machines* [52], *Petri nets* and *vector addition systems* [65], or *flat counter automata* [31, 55]. These classes pose certain restrictions on the syntax of transition labels (vector addition systems, flat counter automata), control structure (flat counter automata), or semantics (reversal-bounded counter machines) of integer programs. Often, decidability of various problems is proved by defining the set of reachable states in a decidable logic, such as Presburger arithmetic [63].

The termination problem was studied for Petri nets where it was shown undecidable when strong fairness is considered [28] and decidable for weak fairness assumptions [53]. A direct consequence of a result on the liveness problem from [36] is that the termination problem is decidable for reversal-bounded counter automata. Concerning flat counter automata, the decidability status of the termination problem remains open.

2.2 Precise Model Checking Methods

A closely related line of work consists in attempts to apply model checking techniques to the verification of integer programs. To solve the reachability problem, one typically proceeds by computing the set of states that are reachable from the initial set I via the transition relation R of the inspected system, and checking the emptiness of the intersection with the set of error states. The set of reachable states can be defined as the post-image of the initial set I via the transitive closure R^* of the transition relation R, formally as $R^*(I)$. The computation of $R^*(I)$ is usually done by computing successive under-approximations of the set of reachable states. Consider a naive technique that computes the set of reachable states S by first initializing S_0 with I ($S_0 \leftarrow I$) and then iteratively computing $S_{i+1} \leftarrow S_i \cup R(S_i)$, where $R(S_i)$ denotes the post-image of S_i via the transition relation R. The computation terminates if $S_k = S_{k+1}$ for some $k \geq 0$. Termination is guaranteed only if the set of reachable states is finite. Therefore, early methods for verification of infinite state systems like FIFO-channel systems [13, 11] attempted to increase the likelihood of convergence of the computation by applying *acceleration* techniques. Essentially, given a loop L of a system and a reachability set S_i computed so far, an acceleration technique aims to compute effects of executing the loop L arbitrarily many times, formally, to compute $S_{i+1} \leftarrow S_i \cup L^+(S_i)$ where L^+ denotes the transitive closure of L. As a result, S_{i+1} may potentially contain infinitely more states than S_i.

To this end, it is important to know for which classes of arithmetic relations it is possible to compute the transitive closure precisely. To the best of our knowledge, the three main classes of integer relations for which transitive closures can be computed effectively and precisely are: (1) difference bounds constraints [31, 22], (2) octagons [17], and (3) finite monoid affine transformations [10, 38]. For these three classes, the transitive closures can be moreover defined in Presburger arithmetic.

The model checking methods from [10, 38] are based on computation of transitive closures of finite monoid affine relations. For certain restricted classes of systems, one can prove termination of the method from [38] as reported in [6]. These classes are typically equivalent, from a semantical point of view, to *flat* systems. A system is *flat* if it has no nested loops and, moreover, each loop in the system can be accelerated. Another model checking method from [2] is based on an extrapolation heuristics that guesses an effect of iterating a loop infinitely many times and then checks whether this guess is correct.

The above methods lack modularity, which is one of the keys to scalability. Since larger programs are usually organized in many small functions, a modular verification approach aims at running an analysis per function (in isolation), computing a *function summary* which is a relation between the input and output valuations of its parameters, and combining the results in a final verification condition.

On what concerns the existing acceleration algorithms for difference bounds and octagonal relations, they suffer from poor scalability in the number of variables. As reported in [17], these methods can only handle difference bounds relations with no more than 10 variables and octagonal relations with no more than 5 variables.

2.2.1 Methods Based on Abstractions

Another, orthogonal, direction of work is concerned with finding sound (but not necessarily complete) answers to decision problems in a cost-effective way. Such approaches, based on the theory of *abstract interpretation* [33], use abstract domains (such as, e.g., polyhedra [34], octagons [57], etc.) and compute fixed points of the transfer functions, which are over-approximations of the sets of reachable states. A drawback of the methods based solely on abstract interpretation is the inability to deal with spurious counterexamples (false positives), i.e., errors caused by the use of a too coarse abstract domain. To ensure termination of state space exploration, abstract interpretation applies *widening*. In contrast to acceleration, the widening can be seen as an operator that computes the set of reachable states S_{i+1} as an extrapolation of S_i and $R(S_i)$ such that $S_{i+1} \supseteq S_i \cup R(S_i)$.

The method of *predicate abstraction* [42] combines ideas from abstract interpretation and model checking in order to compute program invariants in a goal-driven fashion. The underlying idea is to verify a program by reasoning about its *abstraction* that is easier to analyze, and is defined with respect to a set of predicates [42]. The set of predicates is refined, by adding new predicates, to achieve the precision needed to prove the absence or the presence of errors [4]. This technique is also known as Counter Example-based Abstraction Refinement (CEGAR) [30]. Typically, predicate abstraction computes an over-approximation of the transition system generated by a program and verifies whether an error state is reachable in the abstract system. If no error occurs in the abstract system, the algorithm reports that the original system is safe. Otherwise, if a path to an error state (counterexample) has been found in the abstract system, the corresponding concrete path is checked. If this latter path corresponds to a real execution of the system, then a real error has been found. Otherwise, the counterexample is spurious, the abstraction is refined in order to exclude the counterexample, and the procedure continues.

A key difficulty in this approach is to automatically find, during the refinement phase, predicates that make the abstraction sufficiently precise [5]. A breakthrough technique is to generate predicates based on *Craig interpolants* [35] derived from the proof of infeasibility of a spurious trace [48]. Adding the interpolant to the set of predicates prevents the spurious trace from reappearing in the refined abstract model. Cutting edge CEGAR tools, such as, e.g., ARMC [62], BLAST [49] or CPAchecker [9], are empirically successful on a variety of domains and are quite effective in finding bugs and certifying correctness of real-life systems (device drivers, web servers, subsystems of operating system kernels). However, abstraction refinement using interpolants suffers from unpredictability of interpolants computed by provers, which can cause the verification process to diverge and never discover a sufficient set of predicates (even in cases such predicates exist). The failure of such a refinement approach manifests in a sequence of predicates that rule out longer and longer counterexamples, but still fails to discover more general inductive invariants that would rule out multiple spurious counterexamples during the refinement step.

3. Background

This sections presents notions used later in this paper. We give basic notions on integers relations in Section 3.2. Next, Section 3.3 presents programs with integer data. Sections 3.4, 3.5, and 3.6 introduce the three classes of relations we study in this thesis: difference bounds, octagonal, and finite monoid affine relations, respectively.

3.1 Presburger Arithmetic

A *linear term* t over a set of variables in $\mathbf{x}$ is a linear combination of the form $a_0 + \sum_{i=1}^{n} a_i x_i$, where $a_0, a_1, \ldots, a_n \in \mathbb{Z}$. An *atomic proposition* is a predicate of the form either $t \leq 0$, or $t \equiv_c 0$, where t is a linear term, $c \in \mathbb{N}_+$ is a strictly positive integer, and $\equiv_c$ is the equivalence relation modulo c. Given an integer $c \in \mathbb{N}_+$ and a linear term t, we write $c \mid t$ to denote the predicate $t \equiv_c 0$. *Presburger arithmetic* is the first-order logic over atomic propositions of the form either $t \leq 0$ or $t \equiv_c 0$. Presburger arithmetic has quantifier elimination and is decidable [63]. For simplicity we consider only formulas in Presburger arithmetic in this thesis.

3.2 Integer Relations

In the rest of this paper we will fix the set of variables $\mathbf{x} = \{x_1, x_2, \ldots, x_N\}$, for some $N > 0$. The set of *primed* variables is $\mathbf{x}' = \{x'_1, x'_2, \ldots, x'_N\}$. These variables are assumed to be ranging over $\mathbb{Z}$.

Let $R_1, R_2 \subseteq \mathbb{Z}^N \times \mathbb{Z}^N$ be integer relations. Relational composition is defined as $R_1 \circ R_2 = \{(s, s') \in \mathbb{Z}^N \times \mathbb{Z}^N \mid \exists s'' \in \mathbb{Z}^N \,.\, (s, s'') \in R_1 \wedge (s'', s') \in R_2\}$. For any relation $R \subseteq \mathbb{Z}^N \times \mathbb{Z}^N$, we consider R^0 to be the identity relation $\mathcal{I} = \{(s, s) \mid s \in \mathbb{Z}^N\}$ and we define $R^{i+1} = R^i \circ R$, for all $i \geq 0$. We say that R^i is the *i-th power* of R. With these notations, $R^+ = \bigcup_{i=1}^{\infty} R^i$ denotes the *transitive closure* of R, and $R^* = R^+ \cup \mathcal{I}$ denotes the *reflexive and transitive closure* of R. The inverse of R is defined as $R^{-1} = \{(s', s) \mid (s, s') \in R\}$. Further, we define $R^{-m} = (R^m)^{-1}$ for each $m \geq 1$. For any relation $R \in \mathbb{Z}^N \times \mathbb{Z}^N$ and set $S \subseteq \mathbb{Z}^N$, we define the post-image of S via R as $R(S) = \{s' \mid \exists s \in S \wedge (s, s') \in R\}$. Then, the pre-image of S via R is defined as $R^{-1}(S)$.

An integer set $S \subseteq \mathbb{Z}^N$ is typically defined by an arithmetic formula $S(\mathbf{x})$. Similarly, an integer relation $R \subseteq \mathbb{Z}^N \times \mathbb{Z}^N$ is typically defined by an arithmetic formula $R(\mathbf{x}, \mathbf{x}')$. Intuitively, $\mathbf{x}$ represent input variables and $\mathbf{x}'$ represents output variables. Note that for each $m \geq 0$, $R^{-m}(\mathbb{Z}^N)$ can be defined by $\exists \mathbf{x}'.R^m(\mathbf{x}, \mathbf{x}')$. For a relation $R(\mathbf{x}, \mathbf{x}')$ and valuations $\nu, \nu' : \mathbf{x} \to \mathbb{Z}$, we denote by $\nu, \nu' \models R(\mathbf{x}, \mathbf{x}')$ the fact that the formula obtained by replacing each occurrence of $x \in \mathbf{x}$ with $\nu(x)$ and each occurrence of $x' \in \mathbf{x}'$ with $\nu'(x)$ is logically valid.

3.3 Integer Programs

In our work, we address model checking of sequential non-recursive programs with unbounded integer data. We view functions of such programs as *counter automata* (CA), which are essentially control flow graphs where edges are labeled with arithmetic formulae. Control states in CA can be marked as initial (denoting function entry), final (denoting function exits), or error control state (denoting an unwanted program state). Error control states in control flow graphs correspond to calls to the assert function in program. An example of such program and its CA representation is given in Figure 1. Despite its relative simplicity, many tools can fail to prove that the assertion holds. Each edge label defines a relation between values of integers (counters) in the source and the destination control state. The assigned variables appear primed in CAs, e.g. the statement `x:=x+1` is represented as $x' = x + 1$.

```
    var i,j : Int
l0: havoc(i)
    assume(i >= 0)
l1: havoc(j)
    assume(j >= 0)
l2: var x: Int = i
    var y: Int = j
l3: while (x != 0) {
l4:   x = x − 1
l5:   y = y − 1 }
l6: if (i == j) assert(x == y)
```

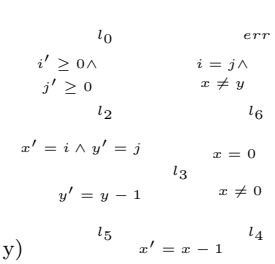

Figure 1: Example program and its corresponding CA. $havoc(x)$ assigns a random value to x.

3.4 Difference Bounds Relations

Difference bounds constraints are known as *zones* in the context of timed automata verification [1] and abstract interpretation [57]. They are defined syntactically as conjunctions of atomic propositions of the form $x - y \leq c$, where x and y are variables and c is an integer constant. Difference bounds constraints can be represented as matrices and graphs. Moreover, their canonical form, useful for efficient inclusion checks, can be computed by the classical Floyd-Warshall algorithm.

Difference bounds relations are difference bounds constraints where variables can be also primed (e.g. $x - x' \leq 0$). Intuitively, primed variables denote future values of variables and unprimed variables denote current values of variables. Difference bounds relations have been studied by Comon and Jurski who showed, in [31], that their transitive closure is Presburger definable. Their proof was subsequently simplified in [22], using the notion of *zigzag* automata. Intuitively, zigzag automaton corresponding to a difference bounds relation R is a finite weighted automaton that encodes m-th power of R by minimal runs of length $m+2$. As we show in Section 4, zigzag automata can be also used in proving *periodicity* of difference bounds relations, which allows to compute R^+ efficiently, using the algorithm presented in Section 4. Furthermore, they also play a crucial role in Section 5 in designing a PTIME algorithm computing the weakest termination sets and in proving existence of linear ranking functions for difference bounds and octagonal relations.

3.5 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 [57]. They are defined syntactically as a conjunctions of atomic propositions of the form $\pm x \pm y \leq c$, where x and y are variables and c is an integer constant. Thus, they can be seen as a generalization of difference bounds constraints. We adopt the classical representation of octagonal constraints (or octagons, for short) $\phi(x_1, \ldots, x_N)$ as difference bounds constraints $\phi(y_1, \ldots, y_{2N})$, where y_{2i-1} stands for $+x_i$ and y_{2i} stands for $-x_i$ with an implicit condition $y_{2i-1} = -y_{2i}$, for each $1 \leq i \leq N$. With this convention, [3] provides an algorithm for computing the canonical form of an octagon, by first computing the canonical form of the corresponding difference bounds constraint and subsequently *tightening* the difference bounds constraints $y_i - y_j \leq c$.

Octagonal relations are defined as octagonal constraints where variables can be also primed. Octagonal relations were studied in [17] where it was shown that the transitive closure is Presburger definable. The core result of [17] is that the canonical form of the m-th power of an octagonal relation R can be computed directly from the m-th power of a difference bounds relation that represents R.

3.6 Finite Monoid Affine Relations

Sections 3.4 and 3.5 presented two classes of non-deterministic relations. In this section, we present linear affine relations which are a general model of deterministic transition relations. Linear affine relations are relations of the form $\mathbf{x}' = A \times \mathbf{x} + \mathbf{b} \wedge \phi(\mathbf{x})$, where $\mathbf{x}' = A \times \mathbf{x} + \mathbf{b}$ is an affine transformation and $\phi(\mathbf{x})$ is a Presburger guard. We present two subclasses of linear affine relations, called *finite monoid affine relations* and *polynomially bounded affine relations*.

The class of finite monoid affine relations was the first class of integer relations for which the transitive closure has been shown to be Presburger definable by Boigelot [10]. Informally, an affine relation is a finite monoid relation if the set of powers of its transformation matrix $\{A^m \mid m \geq 0\}$ is finite. Originally, Boigelot characterized this class by two decidable conditions in [10]. Later, Finkel and Leroux noticed in [38] that Boigelot's conditions correspond to the finite monoid property, which is also known to be decidable [56].

The second subclass of polynomially bounded relations is defined by dropping one of the Boigelot's conditions and by requiring that the guard of a relation is linear. We study this subclass in Section 5 which presents a method for computation of termination preconditions for this class.

4. Computing Transitive Closures of Periodic Relations

This section addresses the acceleration problem for three classes of integer relations presented in Section 3: difference bounds, octagonal, and finite monoid affine relations. We first describe a general framework for computing the transitive closures in Section 4.1. Then, we discuss its instantiations for the three classes in Section 4.2. Next, we discuss the related work in Section 4.3. Finally, Section 4.4 reports on experiments.

4.1 General Theoretical Framework

In this section, we present a general framework for computing *closed forms* and *transitive closures* of certain relations, called *periodic*. The closed form of a relation $R(\mathbf{x}, \mathbf{x}')$ is a relation $\widehat{R}(k, \mathbf{x}, \mathbf{x}')$ where substituting the parameter k with an integer m gives a relation equivalent to R^m for each $m \geq 0$. Once the closed form is computed, the transitive closure of R can be defined as $\exists k \geq 1 \,.\, \widehat{R}(k, \mathbf{x}, \mathbf{x}')$.

We define a notion of periodicity on classes of relations that can be naturally represented as matrices. In general, an infinite sequence of integers is said to be *periodic* if the elements of the sequence situated at equal distance one from another differ by the same quantity (while admitting some non-periodic initial prefix of finite length in the sequence). This definition is generalized to matrices of integers, entry-wise. Assuming that each finite power R^k of a relation R is represented by a matrix M_k, R is said to be periodic if the infinite sequence $\{M_k\}_{k=0}^{\infty}$ of matrix representations of powers of R is periodic. Periodicity guarantees that this sequence has an infinite subsequence which can be captured by an existentially quantified formula. This formula consequently represents infinitely many powers of the relation. Then, the remaining powers can be computed by composing the existentially quantified formula with certain powers of the relation.

For instance, consider the relation R defined as $x' = y + 1 \wedge y' = x$. This relation is periodic, and its odd powers $R^1, R^3, R^5, \ldots$ can be represented by the formula $\exists \ell \geq 0 \,.\, (x' = y + \ell + 1 \wedge y' = x + \ell)$. Even powers R^2, $R^4, R^6, \ldots$ can then be represented by composing this formula with R.

We show that the closed form of a periodic relation can be defined, once the *period* and the *prefix* of the relation are known. We present a generic algorithm that finds a period and a prefix of periodic relations and computes their closed form and transitive closure. The main idea of the algorithm is to discover integers $b \geq 0$ and $c > 0$ such that the sequence $\{M_{b+kc}\}_{k \geq 0}$ is periodic. The algorithm calls MaxConsistent and MaxPeriodic which check periodicity of the $\{M_{b+kc}\}_{k \geq 0}$ sequence with respect to b and c, the current candidates for the prefix and period.

4.2 Instantiations of the Framework

This section describes how the generic transitive closure algorithm described in the previous section can be instantiated to three classes of arithmetic relations for which the transitive closure is known to be definable in Presburger arithmetic: difference bounds relations, octagonal relations, and finite monoid affine transformations. To compute the transitive closure of these relations using the algorithm, one first needs to prove that the three classes are periodic, otherwise termination of the algorithm is not guaranteed. Our proofs rely mostly on a fact that any matrix is periodic when its powers are computed in the tropical semiring $(\mathbb{Z}_\infty, \min, +, \infty, 0)$. The intuition behind periodicity of difference bounds relations is that the k-th power of a relation from this class can be encoded by minimal runs of length k in *zigzag automata* which in turn can be computed as the k-th tropical power of the incidence matrix of the automaton. Thus, periodicity of the sequence of tropical powers of the incidence matrix entails periodicity of a difference bounds relation. Periodicity of the three classes thus provides common grounds to the acceleration problem and also gives shorter proofs for the fact that the transitive closures of these three classes are definable in Presburger arithmetic.

The efficiency of the algorithm depends on two factors. Given a relation with prefix b and period c, the algorithm makes $O((b + c)^2)$ iterations of its main loop. Thus, the prefix b and the period c are important complexity parameters. We prove that the asymptotic bound for them is in $2^{\mathcal{O}(N)}$ where N the number of variables used to define the input relation. For difference bounds and octagonal relations, these bounds are closely related to bounds on the prefix and the period of the incidence matrix of zigzag automata. We therefore also give an alternative proof to the fact that each matrix is periodic in the tropical semiring which moreover gives asymptotic bounds.

Another important efficiency factor is the complexity of the procedures MaxConsistent and MaxPeriodic, that are called by the transitive closure algorithm in order to detect the maximal interval that is periodic with respect to the current prefix and period candidates. In general, for all three classes of relations we consider, these procedures can be implemented using Presburger arithmetic queries. However, in practice, one would like to avoid as much as possible using Presburger solvers, due to reasons of high complexity of decision procedures for Presburger arithmetic. We give direct decision methods which avoid calls to external Presburger or SMT solvers completely and which are of polynomial time complexity in the size of the prefix, period, $\|R\|$, and N, where $\|R\|$ denotes the sum of absolute values of the coefficients of a relation R and N denotes the number of variables used to define a given relation R.

Finally, we prove that for all three classes of integer relations, the transitive closure algorithm runs in EXPTIME in the number of variables, and PTIME in the sum of absolute values of the coefficients of R or, equivalently, in EXPTIME in the size of the binary representation of R.

We have implemented the transitive closure algorithm and observed that in some cases, we can achieve a speed-up of four orders of magnitude when compared with older algorithms for difference bounds and octagonal relations. Moreover, the algorithm scales much better in the number of variables. We report on experiments in Section 4.4.

4.3 Related Work

Acceleration of affine relations has been considered primarily in the works of Annichini et al. [2], Boigelot [10], and Finkel and Leroux [38]. Finite monoid affine relations have been first studied by Boigelot [10] who shows that the finite monoid property is decidable and that the transitive closure is Presburger definable in this case. On

Relation		new	compact		canonical	
			old	speedup	old	speedup
d_0	$(x - x' = -1) \wedge (x = y')$	0.18	0.70	3.90	38.77	215.39
d_1	$(x - x' = -1) \wedge (x' = y')$	0.18	18.18	101.00	38.77	215.39
d_2	$(x - x' = -1) \wedge (x = y') \wedge (x - z' \leq 5) \wedge (z = z')$	1.20	26.50	22.10	33431.20	27859.30
d_3	$(x - x' = -1) \wedge (x = y') \wedge (x - z \leq 5) \wedge (z = z')$	0.60	32.70	54.50	33505.50	55841.70
d_4	$(x - x' = -1) \wedge (x = y) \wedge (x - z \leq 5) \wedge (z = z')$	0.50	702.30	1404.60	48913.80	97827.60
d_5	$(a = c) \wedge (b = a') \wedge (b = b') \wedge (c = c')$	1.80	5556.60	3087.00	$> 10^6$	∞
d_6	$(a - b' \leq -1) \wedge (a - e' \leq -2) \wedge (b - a' \leq -2)$ $\wedge (b - c' \leq -1) \wedge (c - b' \leq -2) \wedge (c - d' \leq -1)$ $\wedge (d - c' \leq -2) \wedge (d - e' \leq -1 \wedge e - a' \leq -1)$ $\wedge (e - d' \leq -2) \wedge (a' - b \leq 4) \wedge (a' - c \leq 3)$ $\wedge (b' - c \leq 4 \wedge b' - d \leq 3) \wedge (c' - d \leq 4) \wedge (c' - e \leq 3)$ $\wedge (d' - a \leq 3 \wedge d' - e \leq 4) \wedge (e' - a \leq 4) \wedge (e' - b \leq 3)$	5.6	$> 10^6$	∞	$> 10^6$	∞
o_1	$(x + x' = 1)$	0.21	0.91	4.33	0.91	4.33
o_2	$(x + y' \leq -1) \wedge (-y - x' \leq -2)$	0.29	0.85	2.93	0.84	2.90
o_3	$(x \leq x') \wedge (x + y' \leq -1) \wedge (-y - x' \leq -2)$	0.32	0.93	2.91	0.94	2.94
o_4	$(x + y \leq 5) \wedge (-x + x' \leq -2) \wedge (-y + y' \leq -3)$	0.21	3.67	17.48	13.52	64.38
o_5	$(x + y \leq 1) \wedge (-x \leq 0) \wedge (-y \leq 0)$	1.20	20050.90	16709.10	$> 10^6$	∞
o_6	$(x \geq 0) \wedge (y \geq 0) \wedge (x' \geq 0) \wedge (y' \geq 0)$ $\wedge (x + y \leq 1) \wedge (x' + y' \leq 1) \wedge (x - 1 \leq x')$ $\wedge (x' \leq x + 1) \wedge (y - 1 \leq y') \wedge (y' \leq y + 1)$	2.5	$> 10^6$	∞	$> 10^6$	∞

Table 1: A comparison with older algorithms on difference bounds and octagons. Times are in milliseconds.

vars	FLATA			FAST				
	done	av.	E_T	done	av.	E_T	E_M	E_B
10	50	1.5	0	49	0.6	0	0	1
15	50	1.6	0	31	10.5	17	0	2
20	50	1.6	0	4	3.4	9	8	29
25	50	1.6	0	2	4.2	2	10	36
50	50	1.6	0	0	-	0	0	50
100	49	7.7	1	0	-	0	0	50

(a) – matrix density 3%

vars	FLATA			FAST				
	done	av.	E_T	done	av.	E_T	E_M	E_B
10	50	1.5	0	22	6.9	23	1	4
15	50	1.5	0	1	20.6	4	3	42
20	50	1.6	0	0	-	1	0	49
25	43	1.7	7	0	-	0	0	50
50	50	2.3	0	0	-	0	0	50
100	42	5.5	8	0	-	0	0	50

(b) – matrix density 10%

Table 2: Comparison with FAST (MONA plugin) on deterministic difference bounds. Times are in seconds. E_T: timeout 30 s, E_B: BDD too large, E_M: out of memory.

what concerns non-deterministic transition relations, the transitive closure of a difference bounds constraint was first shown to be Presburger definable by Comon and Jurski [31]. Their proof was subsequently simplified and extended to parametric difference bounds constraints in [22]. It was shown in [17] that also octagonal relations can be accelerated precisely, and that the transitive closure is also Presburger definable. Our proofs of periodicity are based on some of the previous results from [22, 17]. For difference bounds constraints, we simplify the proof from [22] by applying a result from tropical semiring theory [67] on periodicity of matrices in the tropical semiring. We also give an alternative proof of matrix periodicity that moreover establishes bounds on the size of the prefix and the period of a matrix.

4.4 Experiments

We have implemented the transitive closure algorithm for difference bounds and octagonal relations within the FLATA toolset [50]. We compared the performance of this algorithm with existing transitive closure computation methods for difference bounds [22] and octagonal relations [17].

Table 1 shows the results of the comparison between the older algorithms described in [22, 17] (denoted as *old*) and our new algorithm for difference bounds relations $d_{1,\ldots,6}$ and octagonal relations $o_{1,\ldots,6}$. The tests have been performed on both *compact* (minimum number of constraints) and *canonical* (i.e. closed, for difference bounds and tightly closed, for octagons) relations. The *speedup* column gives the ratio between the *old* and *new* execution times. The experiments were performed on a 2.53GHz machine with 4GB of memory.

As shown in Table 1, the maximum observed speedup is almost 10^5 for difference bounds (d_4 in canonical form) and of the order of four for octagons. For the relations d_5 (canonical form), d_6 and o_6 the computation using older methods took longer than 10^6 msec. It is also worth noticing that the highest execution time with the new method was of 2.5 msec.

Table 2 compares FLATA with the FAST tool [7] on counter systems with one self loop labeled with a randomly generated deterministic difference bounds relation. We generated 50 such relations for each size $N = 10, 15, 20, 25, 50, 100$. Notice that FAST usually runs out of memory for more than 25 variables, whereas FLATA can handle 100 variables in reasonable time (less than 8 seconds on average).

5. Computing Termination Preconditions of Integer Relations

In this section, we address the problem of conditional termination, which is that of defining the *weakest termination set*—the set of initial configurations from which a given program terminates.

5.1 Decidability of the Termination Problem

We define the *weakest non-termination set*—the set of initial configurations from which a non-terminating execu-

tion exists (the dual of the weakest non-termination set)—as the greatest fixpoint of the pre-image of the transition relation. This definition enables the representation of this set whenever (1) the closed form of the relation of the loop is definable in a logic that has quantifier elimination and (2) the greatest fixpoint of the pre-image of the relation can be computed as the infimum of the Kleene sequence. We show that these conditions are met for three classes of relations: difference bounds, octagonal, and finite monoid affine relations, and that the weakest non-termination set of relations from these classes is an effectively computable Presburger formula. This entails the decidability of the termination problem for these classes. The Presburger formula defining the weakest non-termination set is defined using the closed form of a relation. This induces a method for computation of the weakest non-termination set which inherits the asymptotic complexity of the algorithm computing the closed form, which we proved to be EXPTIME in Section 4.

Next, we study the class of linear affine relations and give a method of under-approximating the termination precondition for a non-trivial subclass of polynomially bounded affine relations.

5.2 Computing Termination Preconditions in Polynomial Time

We study the classes of difference bounds and octagonal relations further and observe that the representation of powers of a relation by zigzag automata provides a better argument to decide termination of these classes. More precisely, we notice that well-foundedness of a difference bounds or an octagonal relation manifests in the existence of a negative cycle in the corresponding zigzag automaton, which makes the limit of the Kleene sequence empty. These observations lead to a PTIME algorithm computing the weakest non-termination sets for difference bounds and octagonal relations that avoids the closed form computations and complex quantifier eliminations. Further, the structure of the negative cycle in a zigzag automaton can be used to prove a result on existence of linear ranking functions. We show that we can construct a witness relation R' for each difference bounds or octagonal relation R such that R and R' have equal sets of infinite runs and, moreover, that R' has a linear ranking function if it is well-founded.

5.3 Related Work

The literature on program termination is vast. Most work focuses however on universal termination, such as the techniques for synthesizing linear ranking functions of Van Gelder and Sohn [69] or Podelski and Rybalchenko [60], and the more sophisticated method of Bradley, Manna and Sipma [24], which synthesizes lexicographic polynomial ranking functions, suitable when dealing with disjunctive loops. However, not every terminating program (loop) has a linear (polynomial) ranking function. In this chapter, we show that for an entire class of non-deterministic linear relations, defined using octagons, termination is always witnessed by a computable octagonal relation that has a linear ranking function.

Another line of work considers the decidability of termination for simple (conjunctive) linear loops. Initially, Tiwari [70] showed decidability of termination for affine linear loops interpreted over *reals*, while Braverman [25] refined this result by showing decidability over *rationals* and over *integers*, for homogeneous relations of the form $C_1\mathbf{x} > 0 \ \wedge \ C_2\mathbf{x} \geq 0 \ \wedge \ \mathbf{x}' = A\mathbf{x}$. The non-homogeneous integer case seems to be much more difficult as it is closely related to the open *Skolem's Problem* [47]: given a linear recurrence $\{u_i\}_{i\geq 0}$, determine whether $u_i = 0$ for some $i \geq 0$.

To our knowledge, the first work on proving non-termination of simple loops is reported in [43]. The notion of *recurrent sets* occurs in this work, however, without the connection with fixpoint theory, which is introduced in the present work. Finding recurrent sets in [43] is complete with respect to a predefined set of templates, typically linear systems of rational inequalities.

The work which is closest to ours is probably that of Cook et al. [32]. In that paper, the authors develop an algorithm for deriving termination preconditions by first guessing a ranking function candidate (typically the linear term from the loop condition) and then inferring a supporting assertion which guarantees that the candidate function decreases with each iteration. The step of finding a supporting assertion requires a fixpoint iteration in order to find an invariant condition. Unlike our work, the authors of [32] do not address issues related to completeness: the method is not guaranteed to find the weakest precondition for termination, even in cases when this set can be computed. On the other hand, it is applicable to a large range of programs extracted from real-life software. To compare our method with theirs, we tried the examples available in [32]. For those which are polynomially bounded affine relations, we used our under-approximation method and have computed termination preconditions, which turn out to be slightly more general than the ones reported in [32].

6. Verification of Programs with Integer Data

In this chapter, we show how the techniques developed in the preceding sections, which deal with several restricted classes of program loops, can be used in verification of larger programs. We present several algorithms for reachability and termination analysis of non-recursive programs with integer data. Sections 6.1 and 6.2 present techniques for modular reachability and termination analysis, respectively, while Section 6.3 presents an enhancement of a non-modular reachability analysis algorithm based on predicate abstraction.

6.1 Modular Reachability Analysis

We propose a reachability analysis method that computes *procedure summaries* by tracking relations and computing loop accelerations, using acceleration techniques from Section 4. Since the considered programs are non-recursive, we can order the procedures topologically with respect to the call graph of a program. Then, summaries of procedures without call transitions can be computed first. Subsequently, the remaining procedures can be analyzed in the given topological order by plugging the computed summaries at the call sites. Finally, we check whether the error summary of the main procedure is empty.

A key to computing a procedure summary is acceleration of disjunctive loops, for which we design a semi-algorithm that attempts to compute effects of all possible interleavings of the disjuncts. Supposing the relation is of the form $R_1 \vee \ldots \vee R_k$ where each disjunct R_i can be accelerated

(e.g. by the methods in Section 4), the semi-algorithm enumerates, in a breadth-first manner, all interleavings of accelerated disjuncts, i.e., interleavings of $R_1^+, \ldots, R_k^+$. Thus, it computes increasingly larger underapproximations of R^+. If two such successive underapproximations are equivalent, the algorithm terminates and returns the precise transitive closure.

6.2 Modular Termination Analysis

We describe a method that computes over-approximations of non-termination sets of procedures (thus, its negation is a pre-condition for termination). Since we consider non-recursive programs, there cannot be an infinite sequence of procedure calls that never return, and hence, each non-terminating run must loop infinitely in one of the procedures. By first computing a *transition invariant* of each procedure, one can then compute an over-approximation of the set of initial configurations from which a run that loops infinitely in that procedure exists. Next, these sets can be propagated to the main procedure by computing their pre-image via summaries of procedures. We show that such sets can be computed by combining techniques from Chapter 5, which compute the weakest non-termination sets for certain classes of loops, with the reachability analysis techniques from Section 6.1.

The method can be summarized as follows. For simplicity, suppose first that R is the (disjunctive) transition relation of a procedure without call transitions and that R encodes the program counter. Our method first computes (1) a *transition invariant*, i.e. a relation $R_1^\# \vee \ldots \vee R_m^\#$ which over-approximates the transitive closure of R restricted to reachable states, and (2) a *reachability relation*, i.e., a relation which over-approximates $(R^+ \wedge Init)$. Next, we compute the weakest non-termination set $\mathrm{WNT}(R_i^\#)$ of each disjunct $R_i^\#$, by applying methods from Chapter 5. We prove that computing the pre-image of $\mathrm{WNT}(R_1^\#) \vee \ldots \vee \mathrm{WNT}(R_m^\#)$ via the reachability relation gives an over-approximation of the weakest non-termination set of the procedure. We apply the polynomial time algorithm from Section 5 to compute the weakest non-termination sets $\mathrm{WNT}(R_i^\#)$. A transition invariant and a reachability relation can be computed by a slight adaptation of the techniques described in Section 6.1.

Next, we show that the above approach can be generalized to non-recursive programs with multiple procedures. Since each infinite run of non-recursive program loops infinitely in one of its procedures, one can compute over-approximations of non-termination sets for each procedures by applying ideas from the previous paragraph, and then propagate these sets to the main procedure by computing their pre-image via summaries of procedures.

Our method can be seen as a generalization of a result by Podelski and Rybalchenko [61] who proved that disjunctive well-foundedness of a transition invariant of a program entails termination of that program. We extend their result to computation of over-approximations of non-termination sets.

Finally, we study *flat integer programs*, a class of programs which (1) have no nested loops and (2) each loop is either difference bounds, octagonal, or finite monoid affine relation. We show that the weakest termination set of flat integer programs is Presburger definable and can be effectively computed using our methods described above. This immediately entails decidability of the termination problem for flat integer programs, by decidability of Presburger arithmetic.

6.3 Predicate Abstraction with Acceleration

This section describes how transitive closure computation can help predicate abstraction to deal with the divergence problem by computing inductive interpolants that rule potentially infinite number of spurious counterexamples in one refinement step. The underlying idea is that by accelerating dynamically detected looping patterns in spurious traces, we can analyze multiple concrete traces at once. We further prove that inductive interpolants that rule out multiple traces can be computed directly from classical Craig interpolants and transitive closures of loops.

We have devised Counterexample-Guided *Accelerated* Abstraction Refinement (CEGAAR), a new reachability analysis algorithm which combines *interpolation-based predicate discovery* in counterexample-guided predicate abstraction with *acceleration* technique for computing the transitive closure of loops. CEGAAR applies acceleration to dynamically discovered looping patterns in the unfolding of the transition system and combines overapproximation with underapproximation. It constructs inductive invariants that rule out an infinite family of spurious counterexamples, alleviating the problem of divergence in predicate abstraction without loosing its adaptive nature. We present theoretical and experimental justification for the effectiveness of CEGAAR, showing that inductive interpolants can be computed from classical Craig interpolants and transitive closures of loops.

6.4 Experiments

This section describes an experimental evaluation of techniques for reachability and termination analysis described in Sections 6.1, 6.2, and 6.3.

Reachability Analysis. We have implemented the reachability analysis method based on acceleration and procedure summaries, described in Section 6.1, in the Flata verifier [50]. We use algorithms that are specific to subclasses of integer relations (e.g. difference bounds or octagonal relations) for operations such as composition, satisfiability, and transitive closure. We resort to an external SMT solver Yices [37] only for checking satisfiability of polyhedra and modulo relations. The CEGAAR algorithm, described in Section 6.3, was implemented by building on (1) the Flata verifier [50] that computes accelerations and (2) on the predicate abstraction engine Eldarica [50] which uses the Princess interpolating theorem prover [26, 66] to generate interpolants.

We have collected about 50 benchmarks in the Numerical Transition Systems format (NTS) which were extracted automatically from different sources: (a) C programs with arrays provided as examples of divergence in predicate abstraction [54], (b) verification conditions for programs with arrays, expressed in the SIL logic of [18] and translated to NTS, (c) small C programs with challenging loops, (d) NTS extracted from programs with singly-linked lists by the L2CA tool [12], (e) C programs provided as benchmarks in the NECLA static analysis suite, (f) C programs with asynchronous procedure calls translated into NTS

using the approach of [41] (the examples with extension .optim are obtained via an optimized translation method [40]), and (g) models extracted from VHDL models of circuits following the method of [68].

Comparing FLATA with ELDARICA, we observe that the tools behave in a complementary way. In some cases (examples (a)), the predicate abstraction method fails due to an unbounded number of loop unrollings required by refinement. In these cases, acceleration was capable to find the needed invariant rather quickly. On the other hand (examples (f)), the acceleration approach was unsuccessful in reducing loops with linear but non-octagonal relations. In these cases, the predicate abstraction found the needed Presburger invariants for proving correctness and error traces for the erroneous examples.

Last, we report on our reachability analysis techniques that combine predicate abstraction and acceleration: (1) *static acceleration*—a lightweight acceleration technique generalizing large block encoding (LBE) [8] with transitive closures which simplifies the control flow graph prior to predicate abstraction, and (2) dynamic acceleration (CEGAAR) described in Section 6.3. The results on the set of benchmarks suggest that we have arrived at a fully automated verifier that is robust in verifying automatically generated integer programs with a variety of looping control structure patterns. An important question we explored is the importance of dynamic application of acceleration as well as of overapproximation and underapproximation. In some cases, such as mergesort from the (d) benchmarks and split_vc.1 from (b) benchmarks, the acceleration overhead does not pay off. The problem is that static acceleration tries to accelerate every loop in the CFG rather than accelerating the loops occurring on spurious paths leading to error. Acceleration of inessential loops generates large formulas as the result of combining loops and composition of paths during large block encoding. The CEGAAR algorithm is the only approach that could handle all of our benchmarks. There are cases in which the FLATA tool outperforms CEGAAR. We attribute this deficiency to the nature of predicate abstraction, which tries to discover the required predicates by several steps of refinement. In the verification of benchmarks using CEGAAR, acceleration was exact 11 times in total. In 30 cases, the over-approximation of the loops was successful. In 15 cases, over-approximation failed, and so the tool resorted to under-approximation. This suggests that all techniques that we presented are essential to obtain an effective verifier.

Termination Analysis. We have validated the termination analysis methods described in Section 6.2 by automatically verifying termination of all the octagonal running examples, and of several integer programs synthesized from (i) programs with lists [12] and (ii) VHDL models [68]. We have first verified termination of the LISTCOUNTER and LISTREVERSAL programs, which were obtained using the translation scheme from [12], which generates an integer program from a program manipulating dynamically allocated single-selector linked lists. Using the same technique, we also verified the COUNTER and SYNLIFO programs, obtained by translating VHDL designs of hardware counter and synchronous LIFO [68]. These models have infinite runs for any input values, which is to be expected, as they encode the behavior of synchronous reactive circuits.

Model	Size $\|\mathbf{x}\|$	Size $\|Q\|$	Size $\|T\|$	Time [s]	Termination Set
(a) Examples from [54]					
anubhav (C)	29	20	25	3.2	$i \geq 0$
cousot (C)	29	31	34	4.0	$\bot$
(d) Examples from L2CA [12]					
listcounter (C)	4	31	35	1.2	$\top$
listreversal (C)	7	97	107	32.6	$\top$
(f) VHDL models from [68]					
counter (C)	2	6	13	0.8	$\bot$
register (C)	2	10	49	1.4	$\bot$
synlifo (C)	3	43	1006	1016.4	$\bot$

Table 4: Termination Sets for Integer Programs ($\top$ denotes *true* and $\bot$ denotes *false*).

7. Conclusions

7.1 Summary

We have presented several methods that solve various problems related to formal verification of programs that manipulate integer data. Most of the techniques are built upon a novel algorithm computing transitive closures of difference bounds, octagonal, and finite monoid affine relations, which are shown to be periodic. We have proved that this algorithm runs in EXPTIME in the size of the binary representation of the input relation. Moreover, the experimental evidence for difference bounds and octagonal relations shows that the algorithm scales well in the number of variables and is up to four orders of magnitude faster than previous algorithms.

Next, we have studied the conditional termination problem for the classes of periodic relations and showed that the weakest non-termination set is Presburger definable for relations from these classes. As a consequence, the weakest non-termination set is Presburger definable for flat counter automata. Moreover, we have proved that the weakest non-termination set of octagonal (difference bounds) relations is an octagon (difference bounds constraint) itself and, moreover, that it can be computed in polynomial time.

We have presented a semi-algorithmic method for reachability analysis of non-recursive integer programs that is based on computation of procedure summaries and is therefore modular. It uses the transitive closure algorithm as one of its main components. The algorithm computing the summary relation can be adapted to compute transition invariants which are known to be crucial in proving program termination. We show that transition invariants can be used, in combination with algorithms computing non-termination sets, to compute termination preconditions for integer programs.

Further, we have addressed the divergence problem of the predicate abstraction and showed that it can be alleviated by incorporating acceleration into the abstraction-refinement framework in order to generate interpolants that are inductive and rule out potentially infinite sets of spurious counterexamples. Last but not least, we have performed experiments with a number of benchmarks and showed that for many of them, our methods outperform other existing approaches to verification of integer programs.

7.2 Published Results

The transitive closure algorithm studied in Section 4 is an optimized version of an algorithm that we originally

Model	Time [s] F.	E.	S.	D.
(a) Examples from [54]				
anubhav (C)	0.8	3.0	4.0	3.1
copy1 (E)	2.0	7.2	5.8	5.9
cousot (C)	0.6	-	6.2	5.9
loop1 (E)	1.7	7.1	5.2	5.4
loop (E)	1.8	5.9	4.8	5.4
scan (E)	3.3	-	5.1	5.0
string_concat1 (E)	5.3	-	10.1	7.3
string_concat (E)	4.9	-	7.0	7.5
string_copy (E)	4.6	-	6.3	5.7
substring1 (E)	0.6	9.4	18.2	8.3
substring (E)	2.1	3.3	6.3	3.5
(b) Verification conditions for array programs [18]				
rotation_vc.1 (C)	0.6	2.0	9.5	2.0
rotation_vc.2 (C)	1.6	2.2	18.5	2.2
rotation_vc.3 (C)	1.2	0.3	18.3	0.3
rotation_vc.1 (E)	1.1	1.3	10.2	1.3
split_vc.1 (C)	3.9	3.7	91.1	3.6
split_vc.2 (C)	3.0	2.3	74.1	2.2
split_vc.3 (C)	3.3	0.6	75.0	0.6
split_vc.1 (E)	28.5	2.3	185.6	2.4

Model	Time [s] F.	E.	S.	D.
(c) Examples from [59]				
boustrophedon (C)	-	-	-	14.4
gopan (C)	0.4	-	-	6.4
halbwachs (C)	-	-	7.3	7.0
rate_limiter (C)	31.7	6.1	8.1	5.5
(d) Examples from L2CA [12]				
bubblesort (E)	14.9	9.9	9.5	9.3
insdel (E)	0.1	1.3	2.5	1.4
insertsort (E)	2.0	4.2	5.0	4.0
listcounter (C)	0.3	-	1.9	3.7
listcounter (E)	0.3	1.4	1.6	1.4
listreversal (C)	4.5	3.0	6.0	3.3
listreversal (E)	0.8	2.7	8.1	2.8
mergesort (E)	1.2	7.7	21.3	7.4
selectionsort (E)	1.5	8.1	13.7	7.7
(e) NECLA benchmarks				
inf1 (E)	0.2	2.0	2.0	2.0
inf4 (E)	0.9	3.7	3.7	3.7
inf6 (C)	0.1	2.0	2.0	2.0
inf8 (C)	0.3	3.6	3.4	3.9

Model	Time [s] F.	E.	S.	D.
(f) Examples from [41]				
h1 (E)	-	5.1	5.6	5.1
h1.optim (E)	0.8	2.9	5.5	2.9
h1h2 (E)	-	9.4	10.1	12.2
h1h2.optim (E)	1.1	3.3	4.4	3.4
simple (E)	-	6.4	7.0	8.4
simple.optim (E)	0.8	3.0	5.1	2.9
test0 (C)	-	23.0	23.4	29.2
test0.optim (C)	0.3	3.2	5.4	3.2
test0 (E)	-	5.4	5.9	5.7
test0.optim (E)	0.6	3.0	5.8	2.9
test1.optim (C)	0.9	4.7	5.9	7.8
test1.optim (E)	1.5	4.4	5.9	4.7
test2_1.optim (E)	1.6	5.2	5.5	5.6
test2_2.optim (E)	2.9	4.6	5.9	4.6
test2.optim (C)	6.4	27.2	30.1	30.0
wrpc.manual (C)	0.6	1.2	1.4	1.2
wrpc (E)	-	7.9	8.4	8.2
wrpc.optim (E)	-	5.1	8.5	5.2
(g) VHDL models from [68]				
counter (C)	0.1	1.6	1.6	1.6
register (C)	0.2	1.1	1.1	1.1
synlifo (C)	16.6	22.1	21.4	22.0

Table 3: Benchmarks for Flata, Eldarica without acceleration, Eldarica with acceleration of loops at the CFG level (Static) and CEGAAR (Dynamic acceleration). The letter after the model name distinguishes Correct from models with a reachable Error state. Items with "-" led to a timeout for the respective approach.

published in [20]. Section 5 extends the results we published in [21] with a PTIME algorithm for the weakest non-termination preconditions of octagonal relations. Section 6.3 presents a predicate abstraction-based method that we published in [51]. In [50], we presented the Flata tool which implements most of the techniques described in this thesis.

7.3 Future Work

The asymptotic bound on the running time of the acceleration algorithm presented in this thesis has been shown to be in EXPTIME. A natural question that arises is whether the acceleration problem can be proved to be NP or PSPACE complete. Another question is whether the search for the correct prefix and period of a relation, performed in the transitive closure algorithm, cannot be optimized more than how it is currently achieved by the MaxConsistent and MaxPeriodic procedures. For instance, if the size of the prefix and the period could be efficiently computed (or at least approximated) directly from the input relations, the search could be significantly improved for relations which have very high prefix or period. Furthermore, direct computability of the size of the prefix and period would render the procedures MaxConsistent and MaxPeriodic superfluous, which would further reduce the complexity of the algorithm.

On what concerns our study of the termination problem for difference bounds and octagonal relations, a direction that is worth pursuing is to study whether the linear ranking functions, for which we proved existence, can be efficiently computed. We have shown that a linear ranking function can be directly constructed from negative cycles in zigzag automata. The problem we encounter here is that the size of zigzag automata is exponential. The question therefore is whether the construction of the zigzag automaton can be bypassed.

More broadly, a possible future research could explore whether the methods presented in this thesis can be extended to larger classes of programs, for instance by allowing recursion and parallelism. A related line of study is whether the (accelerated) predicate abstraction can be extended to handle recursive programs, for instance by making it reason about trace summaries instead of the set of reachable states.

References

[1] R. Alur and D. L. Dill. The theory of timed automata. In *proc. of REX Workshop*, volume 600 of *LNCS*, pages 45–73, Berlin, Heidelberg, 1991. Springer Verlag.

[2] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *Proc. of CAV*, volume 1855 of *LNCS*, pages 419–434, Berlin, Heidelberg, 2000. Springer Verlag.

[3] 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, Berlin, Heidelberg, 2008. Springer Verlag.

[4] T. Ball, R. Majumdar, T. Millstein, and S. K. Rajamani. Automatic predicate abstraction of C programs. In *Proc. of PLDI*, pages 203–213, New York, NY, USA, 2001. ACM.

[5] T. Ball, A. Podelski, and S. K. Rajamani. Relative completeness of abstraction refinement for software model checking. In *Proc. of TACAS*, volume 2280 of *LNCS*, pages 158–172, Berlin, Heidelberg, 2002. Springer Verlag.

[6] S. Bardin, A. Finkel, J. Leroux, and P. Schnoebelen. Flat acceleration in symbolic model checking. In *Proc. of ATVA*, volume 3707 of *LNCS*, pages 474–488, Berlin, Heidelberg, 2005. Springer Verlag.

[7] S. Bardin, J. Leroux, and G. Point. Fast extended release. In *Proc. of CAV*, volume 4144 of *LNCS*, pages 63–66, Berlin, Heidelberg, 2006. Springer Verlag.

[8] D. Beyer, A. Cimatti, A. Griggio, M. E. Keremoglu, and R. Sebastiani. Software model checking via large-block encoding. In *Proc. of FMCAD*, pages 25–32. IEEE, 2009.

[9] D. Beyer and M. E. Keremoglu. CPAchecker: A tool for configurable software verification. In *Proc. of CAV*, volume 6806 of *LNCS*, pages 184–190, Berlin, Heidelberg, 2011. Springer Verlag.

[10] B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces*. PhD Thesis. Université de Liège, 1999.

[11] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using qdds. *Formal Methods in System Design*, 14(3):237–255, 1999.

[12] A. Bouajjani, M. Bozga, P. Habermehl, R. Iosif, P. Moro, and T. Vojnar. Programs with lists are counter automata. In *Proc. of CAV*, volume 4144 of *LNCS*, pages 517–531, Berlin, Heidelberg, 2006. Springer Verlag.

[13] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of fifo-channel systems with nonregular sets of configurations. *Theoretical Computer Science*, 221(1-2):211–250, 1999.

[14] A. Bouajjani, P. Habermehl, A. Rogalewicz, and T. Vojnar. Abstract regular tree model checking of complex dynamic data structures. In *Proc. of SAS*, volume 4134 of *LNCS*, pages 52–70, Berlin, Heidelberg, 2006. Springer Verlag.

[15] A. Bouajjani, P. Habermehl, and T. Vojnar. Verification of parametric concurrent systems with prioritized fifo resource management. In *Proc. of CONCUR*, volume 2761 of *LNCS*, pages 174–190, Berlin, Heidelberg, 2003. Springer Verlag.

[16] A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract regular model checking. In *Proc. of CAV*, volume 3114 of *LNCS*, pages 372–386, Berlin, Heidelberg, 2004. Springer Verlag.

[17] M. Bozga, C. Gîrlea, and R. Iosif. Iterating octagons. In *Proc. of TACAS*, volume 5505 of *LNCS*, pages 337–351, Berlin, Heidelberg, 2009. Springer Verlag.

[18] M. Bozga, P. Habermehl, R. Iosif, F. Konečný, and T. Vojnar. Automatic verification of integer array programs. In *Proc. of CAV*, volume 5643 of *LNCS*, pages 157–172, Berlin, Heidelberg, 2009. Springer Verlag.

[19] M. Bozga and R. Iosif. On flat programs with lists. In *Proc. of VMCAI*, volume 4349 of *LNCS*, pages 122–136, Berlin, Heidelberg, 2007. Springer Verlag.

[20] M. Bozga, R. Iosif, and F. Konečný. Fast acceleration of ultimately periodic relations. In *Proc. of CAV*, volume 6174 of *LNCS*, pages 227–242, Berlin, Heidelberg, 2010. Springer Verlag.

[21] M. Bozga, R. Iosif, and F. Konečný. Deciding conditional termination. In *Proc. of TACAS*, volume 7214 of *LNCS*, pages 252–266, Berlin, Heidelberg, 2012. Springer Verlag.

[22] M. Bozga, R. Iosif, and Y. Lakhnech. Flat parametric counter automata. *Fundamenta Informaticae*, 91(2):275–303, 2009.

[23] M. Bozga, R. Iosif, and S. Perarnau. Quantitative separation logic and programs with lists. In *Proc. of IJCAR*, volume 5195 of *LNCS*, pages 34–49, Berlin, Heidelberg, 2008. Springer Verlag.

[24] A. R. Bradley, Z. Manna, and H. B. Sipma. Linear ranking with reachability. In *Proc. of CAV*, volume 3576 of *LNCS*, pages 491–504, Berlin, Heidelberg, 2005. Springer Verlag.

[25] M. Braverman. Termination of integer linear programs. In *Proc. of CAV*, volume 4144 of *LNCS*, pages 372–385, Berlin, Heidelberg, 2006. Springer Verlag.

[26] A. Brillout, D. Kroening, P. Rümmer, and T. Wahl. An interpolating sequent calculus for quantifier-free Presburger arithmetic. In *Proc. of IJCAR*, volume 6173 of *LNCS*, pages 384–399, Berlin, Heidelberg, 2010. Springer Verlag.

[27] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[28] H. Carstensen. Decidability questions for fairness in Petri nets. In *Proc. of STACS*, volume 247 of *LNCS*, pages 396–407, Berlin, Heidelberg, 1987. Springer Verlag.

[29] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Proc. of Logic of Programs*, volume 131 of *LNCS*, pages 52–71, Berlin, Heidelberg, 1982. Springer Verlag.

[30] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM*, 50(5):752–794, 2003.

[31] H. Comon and Y. Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In *Proc. of CAV*, volume 1427 of *LNCS*, pages 268–279, Berlin, Heidelberg, 1998. Springer Verlag.

[32] B. Cook, S. Gulwani, T. Lev-Ami, A. Rybalchenko, and M. Sagiv. Proving conditional termination. In *Proc. of CAV*, volume 5123 of *LNCS*, pages 328–340, Berlin, Heidelberg, 2008. Springer Verlag.

[33] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of POPL*, pages 238–252, New York, NY, USA, 1977. ACM.

[34] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84–97, Tucson, Arizona, 1978. ACM Press, New York, NY.

[35] W. Craig. Linear reasoning. A new form of the Herbrand-Gentzen theorem. *The Journal of Symbolic Logic*, 22(3):250–268, September 1957.

[36] Z. Dang, O. H. Ibarra, and P. S. Pietro. Liveness verification of reversal-bounded multicounter machines with a free counter. In *FSTTCS*, pages 132–143, 2001.

[37] B. Dutertre and L. de Moura. The YICES SMT Solver. `http://yices.csl.sri.com/`.

[38] 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, Berlin, Heidelberg, 2002. Springer Verlag.

[39] A. Finkel, E. Lozes, and A. Sangnier. Towards model-checking programs with lists. In *Proc. of ILC*, volume 5489 of *LNCS*, pages 56–86, Berlin, Heidelberg, 2007. Springer Verlag.

[40] P. Ganty. Personal communication, 2012.

[41] P. Ganty and R. Majumdar. Algorithmic verification of asynchronous programs. *ACM Trans. Program. Lang. Syst.*, 34(1):6:1–6:48, 2012.

[42] S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In *Proc. of CAV*, volume 1254 of *LNCS*, pages 72–83, Berlin, Heidelberg, 1997. Springer Verlag.

[43] A. Gupta, T. A. Henzinger, R. Majumdar, A. Rybalchenko, and R. Xu. Proving non-termination. In *Proc. of POPL*, pages 147–158, New York, NY, USA, 2008. ACM.

[44] P. Habermehl, R. I., and T. Vojnar. A logic of singly indexed arrays. In *Proc. of LPAR*, volume 5330 of *LNCS*, pages 558–573, Berlin, Heidelberg, 2008. Springer Verlag.

[45] P. Habermehl, R. Iosif, A. Rogalewicz, and T. Vojnar. Proving termination of tree manipulating programs. In *Proc. of ATVA*, volume 4762 of *LNCS*, pages 145–161, Berlin, Heidelberg, 2007. Springer Verlag.

[46] P. Habermehl, R. Iosif, and T. Vojnar. What else is decidable about integer arrays? In *Proc. of FoSSaCS*, volume 4962 of *LNCS*, pages 474–489, Berlin, Heidelberg, 2008. Springer Verlag.

[47] V. Halava, T. Harju, M. Hirvensalo, and J. Karhumaki. Skolem's problem – on the border between decidability and undecidability, 2005.

[48] T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from proofs. In *Proc. of POPL*, pages 232–244, New York, NY, USA, 2004. ACM.

[49] T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Software verification with BLAST. In *Proc. of SPIN*, volume 2648 of *LNCS*, pages 235–239, Berlin, Heidelberg, 2003. Springer Verlag.

[50] H. Hojjat, R. Iosif, F. Garnier, F. Konečný, V. Kuncak, and P. Rümmer. A verification toolkit for numerical transition systems. In *Proc. of FM*, 2012. To appear.

[51] H. Hojjat, R. Iosif, F. Konečný, V. Kuncak, and P. Rümmer. Accelerating interpolants. In *Proc. of ATVA*, 2012. To appear.

[52] O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, January 1978.

[53] P. Jančar. Decidability of a temporal logic problem for Petri nets. *Theoretical Computer Science*, 74(1):71–93, 1990.

[54] R. Jhala and K. L. McMillan. A practical and complete approach to predicate refinement. In *Proc. of TACAS*, volume 3920 of *LNCS*, pages 459–473, Berlin, Heidelberg, 2006. Springer Verlag.

[55] J. Leroux and G. Sutre. Flat counter automata almost everywhere! In *Proc. of ATVA*, volume 3707 of *LNCS*, pages 489–503, Berlin, Heidelberg, 2005. Springer Verlag.

[56] A. Mandel and I. Simon. On finite semigroups of matrices. *Theoretical Computer Science*, 5(2):101–111, 1977.

[57] A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.

[58] M. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

[59] D. Monniaux. Personal Communication, 2012.

[60] A. Podelski and A. Rybalchenko. A complete method for the synthesis of linear ranking functions. In *Proc. of VMCAI*, volume 2937 of *LNCS*, pages 465–486, Berlin, Heidelberg, 2004. Springer Verlag.

[61] A. Podelski and A. Rybalchenko. Transition invariants. In *LICS'04*, pages 32–41, 2004.

[62] A. Podelski and A. Rybalchenko. ARMC: The logical choice for software model checking with abstraction refinement. In *Proc. of PADL*, volume 4354 of *LNCS*, pages 245–259. Springer Verlag, Berlin, Heidelberg, 2007.

[63] M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. *Comptes rendus du I Congrés des Pays Slaves*, page 92âĂŞ101, 1929.

[64] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in cesar. In *Proc. of the 5th Colloquium on International Symposium on Programming*, volume 137 of *LNCS*, pages 337–351, Berlin, Heidelberg, 1982. Springer Verlag.

[65] C. Reutenauer. *The mathematics of Petri nets*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990.

[66] P. Rümmer. A constraint sequent calculus for first-order logic with linear integer arithmetic. In *Proc. of LPAR*, volume 5330 of *LNCS*, pages 274–289, Berlin, Heidelberg, 2008. Springer Verlag.

[67] B. D. Schutter. On the ultimate behavior of the sequence of consecutive powers of a matrix in the max-plus algebra. *Linear Algebra and its Applications*, 307:103–117, 2000.

[68] A. Smrcka and T. Vojnar. Verifying parametrised hardware designs via counter automata. In *Proc. of HVC*, volume 4899 of *LNCS*, pages 51–68, Berlin, Heidelberg, 2007. Springer Verlag.

[69] K. Sohn and A. Van Gelder. Termination detection in logic programs using argument sizes. In *Proc. of PODS*, pages 216–226, New York, NY, USA, 1991. ACM.

[70] A. Tiwari. Termination of linear programs. In *Proc. of CAV*, volume 3114 of *LNCS*, pages 70–82, Berlin, Heidelberg, 2004. Springer Verlag.

Selected Papers by the Author

M. Bozga, P. Habermehl, R. Iosif, F. Konečný, and T. Vojnar. Automatic verification of integer array programs. In *Proc. of CAV*, volume 5643 of *LNCS*, pages 157–172, Berlin, Heidelberg, 2009. Springer Verlag.

M. Bozga, R. Iosif, and F. Konečný. Fast acceleration of ultimately periodic relations. In *Proc. of CAV*, volume 6174 of *LNCS*, pages 227–242, Berlin, Heidelberg, 2010. Springer Verlag.

M. Bozga, R. Iosif, and F. Konečný. Deciding conditional termination. In *Proc. of TACAS*, volume 7214 of *LNCS*, pages 252–266, Berlin, Heidelberg, 2012. Springer Verlag.

H. Hojjat, R. Iosif, F. Garnier, F. Konečný, V. Kuncak, and P. Rümmer. A verification toolkit for numerical transition systems. In *Proc. of FM*, 2012. To appear.

H. Hojjat, R. Iosif, F. Konečný, V. Kuncak, and P. Rümmer. Accelerating interpolants. In *Proc. of ATVA*, 2012. To appear.