Scalable Linear Invariant Generation with Farkas' Lemma

Hongming Liu, Hongfei Fu, Zhiyong Yu, Jiaxin Song, Guoqiang Li

Location-by-Location Linear Invariant Generation with Farkas' Lemma

HONGMING LIU, Shanghai Jiao Tong University, China
HONGFEI FU, Shanghai Jiao Tong University, China
ZHIYONG YU, Shanghai Jiao Tong University, China
JIAXIN SONG, Shanghai Jiao Tong University, China
GUOQIANG LI, Shanghai Jiao Tong University, China

Invariant generation is a classical problem that has been studied over decades, with the objective to automatically generate invariants at critical program locations that can be used for the formal analysis of the underlying program. In this work, we propose a novel approach for linear invariant generation via constraint solving and Farkas' Lemma. Our main contribution is an implemented algorithmic insight that generates the linear invariants only at a single program location every time to run the invariant generation process. The insight endows the following advantages over the previous approaches that generate the linear invariants once for all the program locations. First, the insight leads to more subsumptions between polyhedra and hence reduces the number of generator-computation problems involved in the invariant generation. Second, in each generator-computation problem, the insight has the potential to conduct the computation under a projected polyhedral cone with reduced dimension, thus can reduce the time needed for a single generator computation. Third, the insight allows a parallel computation that assigns the whole task of invariant generation over all program locations to multiple processors, for which each processor solves only a small fraction of program locations. Theoretically, we prove that the invariants generated from our approach is at least as tight as from the previous approach [Sankaranarayanan *et al.*, SAS 2004]. Practically, we conduct experiments on realistic scenarios involving complex linear invariants, and show that our approach can be much faster than [Sankaranarayanan *et al.*, SAS 2004] in these examples (even if one considers the sum of the execution time for all the program locations).

1 INTRODUCTION

Invariants. An assertion at a program location is called an *invariant* if it is always satisfied by the values taken by the program variables whenever the location is reached in the execution of the program. Invariants play a fundamental role in program analysis and verification as they provide over-approximation for reachable program states. Therefore, they are widely used in proving basic properties such as safety, reachability (including termination) and time complexity, etc. A detailed demonstration of these applications is as follows.

Safety Analysis. Safety is one of the most classical model checking problems that given a program and a set of safety assertions that must hold at specific program locations of the program, the task is to prove that the assertions indeed hold or report that they might be violated by some execution of the program. As invariants serve as an over-approximation of the set of reachable program states, an effective way for safety analysis is to first generate invariants and then check whether the invariants imply the safety assertions. Many existing approaches for safety analysis rely on invariants to prove the desired safety properties (see e.g. [3, 54, 58]).

Reachability Analysis. Reachability is the most basic liveness property that investigates whether a program location is reachable from a given set of initial program states. The simplest reachability objective is to prove the termination of programs (i.e., reachability to the termination program

Authors' addresses: Hongming Liu, Shanghai Jiao Tong University, Shanghai, China, hm-liu@sjtu.edu.cn; Hongfei Fu, Shanghai Jiao Tong University, Shanghai, China, fuhf@cs.sjtu.edu.cn; Zhiyong Yu, Shanghai Jiao Tong University, Shanghai, China, yuzhiyong18@sjtu.edu.cn; Jiaxin Song, Shanghai Jiao Tong University, Shanghai, China, sjtu_xiaosong@sjtu.edu.cn; Guoqiang Li, Shanghai Jiao Tong University, Shanghai, China, li.g@sjtu.edu.cn.

location). For termination analysis, a principal approach is to synthesize a ranking function [34] whose value always decreases in a well-founded relation along the execution of the underlying program. To synthesize a ranking function, virtually all synthesis algorithms require adequate invariants as input (see e.g. [4, 10, 19, 21, 59]). Beyond termination, reachability analysis also considers the reachability to erroneous program locations, resulting in static detection of program errors. In this scenario, danger invariants (a.k.a inductive reachability witnesses) [6, 27], intuitively as invariants extended with ranking functions, has been proposed as a formal witness to the reachability. Recently, invariants have also been shown necessary in the reachability analysis of probabilistic programs [12, 16, 72, 73].

Time-Complexity Analysis. Another basic problem is to automatically infer asymptotic complexity bounds on the runtime of a program. Current algorithms for tackling this problem, such as [14], rely heavily on adequate invariants.

Accuracy Matters. The quality of the generated invariants is measured by their accuracy, i.e., the amount of over-approximation against the actual set of reachable program states. In both the applications above, the accuracy of the invariants is an important factor. Inaccurate invariants can lead to loose results or even failure to get meaningful results for program analysis and verification.

Invariant Generation. Invariant generation is the classical problem that asks to automatically generate invariants for an input program, and has been studied for decades. Various approaches have been proposed to solve the problem, such as abstract interpretation [23, 25], constraint solving [15, 20, 47], recurrence analysis [32, 45, 49, 50], logical inference [30, 35, 36, 39, 55, 67], machine learning [37, 42, 76], dynamic analysis [26, 57, 68], etc.

Inductive Invariants. To infer invariants at program locations directly is often infeasible, and most existing approaches generate invariants by considering a strengthened notion called *inductive invariants*. An inductive invariant at a program location is an assertion that holds for the first visit to the location and is preserved under every cyclic execution path to and from the location. Inductive invariants are guaranteed to be invariants, and the well-established method to prove that an assertion is an invariant is to find an inductive invariant that strengthens it [20, 54].

Numerical Inductive Invariants. An important category of inductive invariants is that of *numerical inductive invariants* that captures the relationship between the numerical values taken by the program variables. Numerical values are a basic aspect of programs, and many common failures of programs (such as array out-of-bound, division by zero, etc.) are closely related to numerical values. Thus, numerical inductive invariants are essential in proving numeric-critical properties. In this work, we consider automated generation of numerical inductive invariants. To be more precise, we consider algorithmic approaches for generating *linear* inductive invariants, as follows.

Linear Inductive Invariants. A notable subclass of numerical inductive invariants is the class of linear inductive invariants. Informally, a numerical inductive invariant is linear if the invariant takes the form of a system of linear inequalities over the program variables. Linear inductive invariants are the most basic form of numerical invariant invariants, hence is important both for the academic and practical purpose. To resolve the automated generation of linear inductive invariants, we consider the method of constraint solving below.

The Method of Constraint Solving. To solve the invariant-generation problem, constraint-solving based approaches usually consider the following paradigm: first establish a template with unknown parameters for the target invariant, then collect constraints from the inductive condition for invariants, and finally solve the unknown parameters in the template to get the desired invariants. Constraint-solving based approaches for numerical invariant generation can roughly be classified by linear and polynomial invariant generation. For linear invariant generation, Farkas' Lemma provides

a complete characterization of the inductive condition and has been studied in [20, 65], which was further solved by quantifier elimination [20] and several heuristics [65]. The STING invariant generator [70] implements the approach in [65], and the INVGEN invariant generator [40] integrates abstraction interpretation and the approach in [65]. Besides, an approach based on eigenvectors for a restricted class of invariants is proposed in [29]. Recently, probabilistic linear invariants have also been considered in probabilistic programs through Farkas' Lemma and Motzkin's Transposition theorem [17, 48]. For polynomial invariant generation, a variety of approaches were proposed in the literature. Complete approaches (that typically have high runtime complexity) were proposed through quantifier elimination [47] and other computer-algebra based techniques [75]. Semi-complete approaches (that have lower runtime complexity but retain completeness in restricted situations as compared with complete approaches) through Positivstellensätze have been proposed in [15]. The special case of polynomial-equality invariants was solved completely by Zariski-closure [43] and Gröbner basis [62]. Heuristics for polynomial invariant generation have also been extensively studied, such as semidefinite programming with relaxation [1, 22, 53], Lagrange interpolation [18], reduction to linear algebra [28], Hypergeometric sequences [44] and Gröbner basis [64]. Recently, an approach based on Stengle's Postivstellensatz for generating probabilistic polynomial invariants in probabilistic programs is proposed in [33]. Compared with other methods (such as abstract interpretation, machine learning, etc.), constraint solving has the advantage of having a theoretical guarantee on the accuracy of the generated invariants based on the considered numerical form of the invariants, but typically require higher runtime complexity.

Our Contribution. We follow the constraint-solving method and consider automated generation of linear inductive invariants over affine programs. An affine program is an imperative program where every assignment is an affine expression over program variables and every guard condition (in e.g. conditional branches, while loops, etc.) is a propositional combination of comparison between affine expressions over program variables. Our work is based on the previous works [20, 65] that establish the first framework for solving linear inductive invariants through Farkas' Lemma. The contribution of this work lies at the key insight that instead of considering all the program locations simultaneously when applying the approaches in [20, 65], one could focus on a single program location and only solves the invariant at the concerned location. The insight enables a more effective subsumption testing, generator computation of polyhedral cones with fewer dimension, and the possibility to speed up the whole invariant generation through parallel computation where muliple processors compute the invariants together and each processor handles only a small fraction of program locations. We show that the new approach with our novel insight outputs at least as tight invariants as compared with the original approach [65]. Moreover, experimental results show that our insight leads to dramatic improvement on the amount of runtime needed for invariant generation over large realistic examples, as compared with [65]. Note that although our insight is simple, to our best knowledge, our result is the first major improvement after almost two decades (since [65]) in constraint solving approaches.

1.1 Related Works

Compared with the vast amount of existing works for invariant generation, our result has the following novelties.

- Compared with the most relevant previous results [65] in constraint solving, our approach has the potential of substantially improving the runtime by adopting our novel insight of focusing on one program location at a time. Experimental results show that our approach is orders of magnitude faster than the previous approach [65].

- Compared with other methods, we follow the constraint-solving method that in general has a theoretical guarantee on the accuracy of the generated invariants in a given specific form. Note that our approach is based on [65] that uses several heuristics to reduce the amount of runtime, which means that these heuristics decrease the original full theoretical guarantee from constraint solving. However, as shown in [65], adopting these heuristics still ensures substantial improvement on the accuracy of the generated invariants against the mainstream approach of abstract interpretation. Our approach guarantees to output at least as tight invariants as from [65], therefore inherits the merit of high accuracy from constraint solving.

Below we compare our approach with existing approaches in more detail.

Constraint Solving. Our result falls in the category of linear invariant generation, thus is incomparable with the approaches for polynomial invariant generation [1, 15, 18, 22, 28, 43, 44, 47, 53, 62, 64, 75]. Below we compare results on linear invariant generation. Compared with results [20, 65] that are based on Farkas' Lemma, our approach further incoporates the novel sight of handling a single program location at a time that can substantially speed up the invariant generation process while preserving the precision of the generated invariants. Compared with the result [29] that uses eigenvectors to tackle restricted classes of linear invariants, our result tackles the general class of affine programs, thus focusing on a completely different aspect.

Abstract Interpretation. A mainstream method to find inductive invariants is *abstract interpretation* [2, 8, 13, 24, 56, 61, 63], which is the oldest and most classical approach to invariant generation. Roughly speaking, the method of abstract interpretation first establishes an abstract domain for the specific form of invariants to be generated, and then perform forward/backward propagation to reach a fixed point. Compared with constraint solving, abstract interpretation does not provide any guarantee on the accuracy of the generated invaraints, except for some rare cases [38]. This point is experimental observed in [65], where the method of constraint solving can produce much tighter invariants even with several heuristics that loses some precision. In our results, we prove that our approach generates at least as tight invariants as the approach in [65], hence inherits the advantage of higher accuracy from constraint solving.

Recurrence Analysis. Another closely-related method is *recurrence analysis* [11, 32, 45, 49, 50]. The method of recurrence analysis usually first transforms the invariant-generation problem into a recurrence relation, and then solve the invariants through the analysis of the recurrence relation. Compared with constraint solving, the disadvantage of recurrence-based approaches is that they are only applicable to the situations where the underlying recurrence relation has a closed form solution, while with appropriate templates constraint solving can be applied to any program.

Logical Inference. Invariants could also be obtained through logical inference, such as abductive inference [30], Craig interpolation [35, 55], ICE learning [36, 74], predicate abstraction [39], random search [67], etc. Compared with constraint solving, logical-inference based approaches usually could not give any theoretical guarantee on the accuracy of the generated invariants.

Machine Learning. The method of machine learning [37, 42, 76] solves the invariant-generation problem as follows: first, one needs to establish a (typically large) training set of programs whose invariants are given; then, trainning approaches (such as neural networks) are applied to the trainning set to train an invariant generator; finally, the trained invariant generator is used to generator invariants for programs outside the trainning set. Compared with constraint solving, machine-learning based approaches require a large training set to cover as most typical situtations as possible, and still cannot guarantee the correctness and accuracy of the generated invariants.

Dynamic Analysis. Dynamic analysis [26, 57, 68] first runs the program of concern for multiple times to collect its execution data, and then guess the invariants based on the collected data. Compared with constraint solving that statically generates the invariants with correctness and precision guarantee, dynamic analysis needs to run the program multiple times to collect a potentially large amount of execution data, and similar to machine learning cannot give any guarantee on the correctness and accuracy of the generated invariants.

Summary. We consider linear-invariant generation over affine programs. Our approach is based on constraint solving and hence can achieve better accuracy of the generated invariants than methods such as abstract interpretation or machine learning, and a wider application range than recurrence analysis. Compared with existing constraint-solving based approaches, our approach is capable of substantially speeding up the process of invariant generation through the insight of generating the invariants location by location instead of handling all program locations in one shot.

2 LINEAR TRANSITION SYSTEMS AND INVARIANTS

We consider linear transition systems (LinTS's) [65] as the underlying model for invariant generation. A LinTS is composed of locations and linear transitions between locations, thus is suitable for modelling the executions of an affine program, for which a location in a LinTS corresponds to a program location (a.k.a program counter) of an affine program, and the linear transitions corresponds to the affine updates (arising from assignment statements) and guards (arising from if-branches and while-loops) in the program. Throughout this work, we denote by $\mathbb{N}, \mathbb{Z}$ and $\mathbb{R}$ the sets of natural numbers (including zero), integers and real numbers respectively.

To present the definitions for LinTS's, we first define the basic notions of linear (in)equalities and assertions. For linear inequalities, we only consider the non-strict comparison operator $\geq$. Note that although an equality $\alpha = \beta$ can be equivalently expressed by two inequalities $\alpha \leq \beta$ and $\alpha \geq \beta$, the equalities in a LinTS are tackled directly as various optimizations could be applied to equalities. Also note that inequalities of the form $\alpha \leq \beta$ could be equivalently tranformed into $-\alpha \geq -\beta$.

Linear (In)equalities and Assertions. A *linear equality* over a set $V = \{x_1, \ldots, x_n\}$ of real-valued variables is of the form $a_1x_1 + \cdots + a_nx_n + b = 0$, where a_i's and b are real coefficients. Analogously, a *linear inequality* over V is of the form $a_1x_1 + \cdots + a_nx_n + b \geq 0$. A *linear assertion* over V is a conjunction of linear equalities and inequalities over V.

Then we present the model of linear transition systems.

Linear Transition Systems. A *linear transition system* (LinTS) is a tuple $\langle X, X', L, \mathcal{T}, \ell^*, \theta\rangle$ where we have:

- X is a finite set of real-valued variables such that each variable $x \in X$ represents the current value of the variable, while $X' = \{x' \mid x \in X\}$ is the corresponding set of *primed variables* such that each primed variable $x' \in X'$ represents the value of the unprimed counterpart $x \in X$ in the next step of the system;
- L is a finite set of *locations* and $\ell^* \in L$ is the initial location;
- $\mathcal{T}$ is a finite set of *transitions* where each transition τ is a triple $\langle \ell, \ell', \rho\rangle$ that specifies the transit from the current location ℓ to the next location ℓ' with the guard condition ρ as a linear assertion over $X \cup X'$;
- θ is a linear assertion over the variables X that specifies the initial condition at the initial location ℓ^*.

To describe the behaviour of a LinTS, we further define the notions of valuations, configurations and their associated satisfaction relation as follows.

Valuations and Configurations. A *valuation* over a variable set V is a function $\sigma : V \to \mathbb{R}$ that assigns to each variable $x \in V$ a real value $\sigma(x)$ that corresponds to the current value held by x. In this work, we mainly consider valuations over the variable set X of a LinTS and simply abbreviate "valuation over X" as "valuation" (i.e., omitting X). Given a LinTS, a *configuration* is a pair (ℓ, σ) such that $\ell \in L$ is a location and σ is a valuation (over X), with the intuition that ℓ is the current location and σ specifies the current values for the variables in the LinTS.

The Satisfaction Relation. Given a linear assertion φ over X and a valuation σ (over X), we write $\sigma \models \varphi$ to mean that σ satisfies φ, i.e., φ is true when one substitutes the corresponding values $\sigma(x)$ in σ to all the variables x in φ. Analogously, given two valuations σ, σ' (over X) and a linear assertion φ over $X \cup X'$, we write $\sigma, \sigma' \models \varphi$ to mean that φ is true when one substitutes every variable $x \in X$ by $\sigma(x)$ and every variable $x' \in X'$ by $\sigma'(x)$ in φ. Moreover, given two linear assertions φ, ψ over X, we write $\varphi \models \psi$ to mean that it is always the case that φ implies ψ, i.e., for every valuation σ we have that $\sigma \models \varphi$ implies $\sigma \models \psi$.

Now we describe the semantics of a LinTS.

The Semantics of LinTS's. Informally, a LinTS starts at its initial location ℓ^* with an arbitrary initial valuation σ^* such that $\sigma^* \models \theta$, constituting an initial configuration $(\ell_0, \sigma_0) = (\ell^*, \sigma^*)$; then at each step n ($n \geq 0$), given the current configuration (ℓ_n, σ_n), the LinTS determines the next configuration $(\ell_{n+1}, \sigma_{n+1})$ by first selecting a transition $\tau = \langle \ell, \ell', \rho \rangle$ such that $\ell = \ell_n$ and then choosing $(\ell_{n+1}, \sigma_{n+1})$ to be any configuration that satisfies $\ell_{n+1} = \ell'$ and $\sigma_n, \sigma_{n+1} \models \rho$. Formally, the semantics of an LinTS is specified the notion of paths. A *path* π is a finite sequence of configurations $(\ell_0, \sigma_0) \dots (\ell_n, \sigma_n)$ such that

- (**Initialization**) $\ell_0 = \ell^*$ and $\sigma_0 \models \theta$, and
- (**Consecution**) for every $0 \leq k \leq n-1$, there exists a transition $\tau = \langle \ell, \ell', \rho \rangle$ satisfying $\ell = \ell_k$, $\ell' = \ell_{k+1}$ and $\sigma_k, \sigma_{k+1} \models \rho$.

Intuitively, a path starts with some legitimate initial configuration (as specified by **Initialization**) and evolves by repeatedly applying the transitions to the current configuration (as described in **Consecution**). Thus, any path $\pi = (\ell_0, \sigma_0) \dots (\ell_n, \sigma_n)$ corresponds to a possible evolution of the underlying LinTS.

Below we present an example LinTS from [20].

Example 1. *Consider a scenario of a vagrant robot from [20]. The control of the robot works in two alternating modes modelled as two locations ℓ_0, ℓ_1. Each mode takes a time between 1 and 2 seconds to complete its task. In mode ℓ_0, the robot moves in the positive direction of x and y, and in mode ℓ_1, it moves in the positive direction of x and negative direction of y. Figure 1 shows the LinTS of a vagrant robot that consists of three variables x, y, t and the two location ℓ_0, ℓ_1. The variable pair (x, y) corresponds to the current position of the robot on the two-dimensional plane, while the variable t records the amount of the elapsed time. From the LinTS, we have the following:*

- *At the start, the robot is at its initial position $(0, 0)$ (i.e., $x = 0$ and $y = 0$) at time $t = 0$ in the initial mode ℓ_0.*
- *At the location ℓ_0, the robot takes the transition τ_1. During the transition, the robot first moves for a time period $\Delta t = t' - t$ between 1 to 2 seconds (as indicated by the linear assertion $1 \leq t' - t \leq 2$ in ρ_1), for which the movement is in the positive direction of both the x- and y-axis, each with a nondeterministic distance that falls in the interval $[\Delta t, 2\Delta t]$ (as specified by the linear assertions $\Delta t \leq x' - x \leq 2\Delta t$ and $\Delta t \leq y' - y \leq 2\Delta t$ in ρ_1); then the robot changes its mode to ℓ_1.*
- *At the location ℓ_1, the robot takes the transition τ_2 and changes its mode to ℓ_0. The movement of the robot is specified by ρ_2 and similar to the situation at ℓ_0. The only difference is that the*

$$X = \{x, y, t\}, L = \{\ell_0, \ell_1\}, \mathcal{T} = \{\tau_1, \tau_2\}$$
$$\tau_1 : \langle \ell_0, \ell_1, \rho_1 \rangle \ \tau_2 : \langle \ell_1, \ell_0, \rho_2 \rangle$$
$$\theta : x = 0 \wedge y = 0 \wedge t = 0,$$
$$\rho_1 : \begin{bmatrix} t' - t \leq x' - x \leq 2(t' - t) \wedge \\ t' - t \leq y' - y \leq 2(t' - t) \wedge \\ 1 \leq t' - t \leq 2 \end{bmatrix}$$
$$\rho_2 : \begin{bmatrix} t' - t \leq x' - x \leq 2(t' - t) \wedge \\ -(t' - t) \leq y' - y \leq -2(t' - t) \wedge \\ 1 \leq t' - t \leq 2 \end{bmatrix}$$

Fig. 1. The LinTS for a Vagrant Robot

robot now moves in the positive direction along the x-axis and in the negative direction along the y-axis.

- *The robot switches between ℓ_0 and ℓ_1 back and forth by alternately taking the transitions τ_1 and τ_2.*

A possible path under this LinTS would be $(\ell_0, (x, y, t) = (0, 0, 0)), (\ell_1, (x, y, t) = (2, 2, 1)), (\ell_0, (x, y, t) = (3, 1, 2))$. □

In this work, we consider algorithms for linear invariant generation that work on LinTS's as an intermediate model. Informally, an invariant at a location is a logical formula that is always satisfied by the values of the variables whenever the location is entered by some path. An invariant is linear if it is a linear assertion. The formal definition is as follows.

(Linear) Invariants. An *invariant* at a location ℓ of a LinTS is a logical formula φ such that for every path under the LinTS $\pi = (\ell_0, \sigma_0) \dots (\ell_n, \sigma_n)$ and $0 \leq k \leq n$, it holds that $\ell_k = \ell$ implies $\sigma_k \models \varphi$. Furthermore, we say that an invariant φ is *linear* if φ is a linear assertion over the variable set X.

To automatically generate invariants, one often investigates a strengthened notion called *inductive invariants*. Since we only consider linear invariants, we directly present the definition of inductive linear invariants in the form of inductive linear assertion maps.

(Inductive) Linear Assertion Maps. A *linear assertion map* over an LinTS is a function η that maps every location ℓ to a linear assertion $\eta(\ell)$ over the variables X. A linear assertion map η is inductive if the following conditions hold:

- (**Initialization**) $\theta \models \eta(\ell^*)$;
- (**Consecution**) For every transition $\tau = \langle \ell, \ell', \rho \rangle$, we have that $\eta(\ell) \wedge \rho \models \eta(\ell')'$, where $\eta(\ell')'$ is the linear assertion obtained by replacing every variable $x \in X$ in $\eta(\ell')$ with its next-value counterpart $x' \in X'$.

Informally, a linear assertion map is inductive if it is (i) implied by the initial condition given by θ at the initial location ℓ^* (i.e., **Initialization**) and (ii) preserved under the application of every transition (i.e., **Consecution**). By a straightfoward induction on the length of a path under a LinTS, one could verify that the linear assertion at every location in an inductive linear assertion map is guaranteed to be a linear invariant. In the rest of the work, we focus on the automated synthesis of inductive linear assertion maps.

3 AN OVERVIEW OF OUR APPROACH

In this section, we present an overview of our approach. We first review the original approaches in [20, 65] that generate linear invariants through constraint solving and Farkas' Lemma, and then sketch our key insight.

3.1 The Original Approaches [20, 65]

In [20, 65], a specialized constraint-solving approach for linear invariant generation is proposed via Farkas' Lemma [31]. The use of Farkas' Lemma transforms the inductive condition for linear invariants equivalently into a system of quadratic constraints, by solving which one could obtain a concrete linear invariant. The key point is that the use of Farkas' Lemma simplifies the constraints from the inductive condition. In [20], the constraints obtained after applying Farkas' Lemma were solved exactly through quantifier elimination. While in [65], these constraints were solved by several heuristics to mitigate the high runtime complexity caused by quantifier elimination.

To review these two approaches, we first recall Farkas' Lemma. Farkas' Lemma is a fundamental result that characterizes basic relationships between linear inequalities. Below we present the version of Farkas' Lemma that charactertizes the implication from a linear assertion to a linear inequality. We follow the presentation of Farkas' Lemma in [20].

Theorem 3.1 (Farkas' Lemma). *Consider a linear assertion φ over a set $V = \{x_1, \ldots, x_n\}$ of real-valued variables in the form of a conjunction of the following linear inequalities:*

$$\varphi : \begin{array}{c} a_{11} \cdot x_1 + \cdots + a_{1n} \cdot x_n + b_1 \geq 0 \\ \vdots \qquad\qquad \vdots \quad \vdots \\ a_{m1} \cdot x_1 + \cdots + a_{mn} \cdot x_n + b_m \geq 0 \end{array}$$

When φ is satisfiable (i.e., there exists a valuation over V that satisfies φ), we have that it implies a linear inequality ψ

$$\psi : c_1 \cdot x_1 + \cdots + c_n \cdot x_n + d \geq 0$$

(i.e., $\varphi \models \psi$) if and only if there exist non-negative real numbers $\lambda_0, \lambda_1, \ldots, \lambda_m$ such that (i) $c_j = \sum_{i=1}^{m} \lambda_i \cdot a_{ij}$ for all $1 \leq j \leq n$, and (ii) $d = \lambda_0 + \sum_{i=1}^{m} \lambda_i \cdot b_i$. Moreover, φ is unsatisfiable if and only if the inequality $-1 \geq 0$ (as ψ) can be derived from above.

One direction of Farkas' Lemma is straightforward, as one easily sees that if we have a non-negative linear combination of the inequalities in φ that can derive ψ, then it is guaranteed that ψ holds whenever φ is true. Farkas' Lemma further establishes that the other direction is also valid.

Remark 1. *In the statement of Farkas' Lemma above, if we change a single linear inequality $a_{j1}x_1 + \cdots + a_{jn}x_n + b_j \geq 0$ in φ to equality (i.e., $a_{j1}x_1 + \cdots + a_{jn}x_n + b_j = 0$), then the theorem statement still holds with the relaxation that we do not require $\lambda_j \geq 0$. This could be easily observed by first replacing the equality equivalent with both $a_{j1}x_1 + \cdots + a_{jn}x_n + b_j \geq 0$ and $a_{j1}x_1 + \cdots + a_{jn}x_n + b_j \leq 0$, and then applying Farkas' Lemma. By similar arguments, the theorem statement holds upon changing multiple linear inequalities into equalities with the relaxation of non-negativity for the corresponding λ_j's.*

The application of Farkas' Lemma can be conveniently visualized by the tabular form in Table 1, where $\bowtie_1, \ldots, \bowtie_m \in \{=, \geq\}$ and we mulitply $\lambda_0, \lambda_1, \ldots, \lambda_m$ with their inequalities in φ and sum up them together to get ψ. For $1 \leq j \leq m$, if $\bowtie_j$ is $\geq$, we require $\lambda_j \geq 0$, otherwise (i.e., $\bowtie_j$ is $=$) we do not impose constraints on λ_j.

We then recall several concepts from polyhedra theory.

Table 1. The Tabular Form for Farkas' Lemma

$$\begin{array}{c|rcl} \lambda_0 & 1 & \geq & 0 \\ \lambda_1 & a_{11}\cdot x_1 + \cdots + a_{1n}\cdot x_n + b_1 & \bowtie_1 & 0 \\ \vdots & \vdots \quad\quad\quad\quad \vdots & \vdots & \\ \lambda_m & a_{m1}\cdot x_1 + \cdots + a_{mn}\cdot x_n + b_m & \bowtie_m & 0 \\ \hline & c_1\cdot x_1 + \cdots + c_n\cdot x_n + d & \geq & 0 \leftarrow \psi \\ & -1 & \geq & 0 \leftarrow \mathit{false} \end{array} \quad \left.\begin{array}{l}\\\\\\\\\end{array}\right\} \varphi$$

Polyhedra and polyhedral cones. A subset P of $\mathbb{R}^n$ is a *polyhedron* if $P = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{A}\cdot\mathbf{x} \leq \mathbf{b}\}$ for some real matrix $A \in \mathbb{R}^{m\times n}$ and real vector $\mathbf{b} \in \mathbb{R}^m$, where $\mathbf{x}$ is treated as a column vector and the comparison $\mathbf{A}\cdot\mathbf{x} \leq \mathbf{b}$ is defined in the coordinate-wise fashion. A polyhedron P is a *polyhedral cone* if $P = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{A}\cdot\mathbf{x} \leq \mathbf{0}\}$ for some real matrix $A \in \mathbb{R}^{m\times n}$, where $\mathbf{0}$ is the m-dimensional zero vector. It is well-known from Farkas-Minkowski-Weyl Theorem [66, Corollary 7.1a] that any polyhedral cone P can be represented as $P = \{\sum_{i=1}^{k} \lambda_i \cdot \mathbf{g}_i \mid \lambda_i \geq 0 \text{ for } 1 \leq i \leq k\}$ for some real vectors $\mathbf{g}_1, \ldots, \mathbf{g}_k$, where the set of the real vectors $\mathbf{g}_i$'s is called a collection of *generators* for the polyhedral cone P.

Polyhedron projection. For a polyhedron $P = \{(\mathbf{x}^{\mathrm{T}}, \mathbf{u}^{\mathrm{T}})^{\mathrm{T}} \in \mathbb{R}^{p+q} \mid \mathbf{A}\cdot\mathbf{x} + \mathbf{B}\cdot\mathbf{u} \leq \mathbf{c}\}$ where $\mathbf{A} \in \mathbb{R}^{m\times p}, \mathbf{B} \in \mathbb{R}^{m\times q}$ are real matrices and $c \in \mathbb{R}^m$ is a real vector, the projection of P onto the dimensions $\mathbf{x}$ (i.e., the first p dimensions) is defined as the polyhedron $P[\mathbf{x}] := \{\mathbf{x} \in \mathbb{R}^p \mid \exists \mathbf{u} \in \mathbb{R}^q.(\mathbf{x}^{\mathrm{T}}, \mathbf{u}^{\mathrm{T}})^{\mathrm{T}} \in P\}$.

Now we review the previous approaches in [20, 65].

The Invariant-Generation Workflow. Based on Farkas' Lemma, the approaches in [20, 65] generate linear invariants over a LinTS by the following steps. Below we fix an input LinTS with $X = \{x_1, \ldots, x_n\}$.

Step 1 In the first step, both the approaches establish a template for an inductive linear assertion maps. A template η involves a linear inequality $\eta(\ell) = c_{\ell,1}\cdot x_1 + \cdots + c_{\ell,n}\cdot x_n + d \geq 0$ at each location ℓ of the LinTS, such that the coefficients $c_{\ell,1}, \ldots, c_{\ell,n}, d$ are unknown and to be solved.

Step 2 In the second step, both the approaches establish constraints from the initialization and the consecution conditions for invariants. The initialization condition specifies that the linear inequality $\eta(\ell^*)$ at the initial location ℓ^* should be implied by the initial condition θ, i.e., $\theta \models \eta(\ell^*)$. The consecution condition specifies that every transition preserves the linear assertion map η, i.e., for every transition $\langle \ell, \ell', \rho\rangle$ we have that $\eta(\ell) \wedge \rho \models \eta(\ell')'$.

Step 3 In the third step, both the approaches apply Farkas' Lemma to the constraints collected from the initialization condition $\theta \models \eta(\ell^*)$ and the consecution conditions $\eta(\ell) \wedge \rho \models \eta(\ell')'$ for each transition $\langle \ell, \ell', \rho\rangle$. For intialization, we apply the tabular form (Table 1) to obtain

$$\begin{array}{c|rcl} \lambda_0 & 1 & \geq & 0 \\ \lambda_1 & a_{11}x_1 + \cdots + a_{1n}x_n + b_1 & \bowtie_1 & 0 \\ \vdots & \vdots \quad\quad\quad\quad \vdots & \vdots & \\ \lambda_m & a_{m1}x_1 + \cdots + a_{mn}x_n + b_m & \bowtie_m & 0 \\ \hline & c_{\ell^*,1}x_1 + \cdots + c_{\ell^*,n}x_n + d_{\ell^*} & \geq & 0 \leftarrow \eta(\ell^*) \\ & -1 & \geq & 0 \leftarrow \mathit{false} \end{array} \quad \left.\begin{array}{l}\\\\\\\\\end{array}\right\} \theta$$

which results in a linear assertion over the unknown coefficients $c_{\ell^*,1}, \ldots, c_{\ell^*,n}, d$ and the fresh variables $\lambda_0, \lambda_1, \ldots, \lambda_m$. Similarly, the tabular form for the consecution condition of a transition

$\langle \ell, \ell', \rho \rangle$ gives

$$
\begin{array}{c|l}
\mu & c_{\ell,1}x_1 + \cdots + c_{\ell,n}x_n \qquad\qquad + d_\ell \geq 0 \leftarrow \eta(\ell) \\
\lambda_0 & \qquad\qquad\qquad\qquad\qquad\qquad 1 \geq 0 \\
\lambda_1 & a_{11}x_1 + \cdots + a_{1n}x_n + a'_{11}x'_1 + \cdots + a'_{1n}x'_n + b_1 \bowtie_1 0 \\
\vdots & \quad \vdots \qquad\quad \vdots \qquad\quad \vdots \qquad\quad \vdots \\
\lambda_m & a_{m1}x_1 + \cdots + a_{mn}x_n + a'_{m1}x'_1 + \cdots + a'_{mn}x'_n + b_m \bowtie_m 0 \\
\hline
 & c_{\ell',1}x'_1 + \cdots + c_{\ell',n}x'_n + d_{\ell'} \geq 0 \leftarrow \eta(\ell')' \\
 & -1 \geq 0 \leftarrow \mathit{false}
\end{array}
\qquad \rho
$$

where in addition to $\lambda_j, c_{\ell,j}, d_\ell, c_{\ell',j}, d_{\ell'}$ we have a fresh variable μ as the non-negative muliplier for $\eta(\ell)$. Note that for the consecution condition, the constraint obtained is no longer linear since the fresh variable μ is multiplied to $\eta(\ell)$ in the tabular above.

Step 4 In the last step, the (non-linear) constraints collected in the previous step are solved to obtain the concrete values for the unknown coefficients in η, so that a concrete inductive linear assertion map would be obtained. In [20], these constraints were solved through quantifier elimination, while in [65] the constraints were solved through (i) several heuristics to remove the non-linearity and (ii) the transformation of a polyhedral cone originally represented by its linear inequality to its generator representation. A major heuristics adopted in [65] to remove the non-linearity is to guess possible values for the fresh variables μ in the consecution conditions, based on some practical rules such as factorization. Below we focus on the workflow of [65], since it is the most related work to our result. Since the approach of [65] resolves the mulipliers μ for consecution by guessing their concrete values, the guessed values for the consecution condition of each transition of a LinTS result in a disjunction of linear assertions over the unknown coefficients in the template η, where one guessed value for the μ of the transition under concern corresponds to one linear assertion in the disjunction. By combining conjunctively the constraints obtained from the consecution condition (for every tranisition) and those from the initial condition, we obtain a propositional formula in conjunctive normal form (CNF) where each atomic proposition is a linear assertion over the unknown coefficients of η. More specifically, each atomic proposition is in the form $\mathbf{A} \cdot \mathbf{c} \leq \mathbf{0}$ where $\mathbf{c}$ is the vector of unknown coefficients in the template, hence is a polyhedral cone. Having obtained the CNF formula, the approach expands the formula (through the distributive law between conjunction and disjunction) equivalently into a DNF (disjunctive normal form) formula where each disjunctive clause is a conjunction of certain atomic propositions in the original CNF formula (therefore being still a polyhedral cone), and then computes through the double description method [7] and Parma Polyhedra library (PPL) [7, 60] the generators of each disjunctive clause (in the form of a polyhedral cone) once a disjunctive clause is expanded. A key ingredient to reduce the runtime arising from the expansion from CNF to DNF is *subsumption testing* that checks whether the current conjunction of atomic propositions during the expansion is already subsumed by (i.e., implies) the invariants obtained from the previously-expanded disjunctive clauses; once the current conjunction implies the invariants already obtained, one knows that it will not produce more meaningful invariants, hence the computation for the current conjunction halts and the expansion then switches to other branches. By instantiating the computed generators back to the unknown coefficients in the template, the approach obtains the solved concrete linear invariants, for which one generator corresponds to one inductive linear asertion map.

Below we present an example for the workflow of [65].

Example 2. *Consider the LinTS in Example 1. The approach in [65] first establishes a template η by setting $\eta(\ell_i) := c_{\ell_i,1}x + c_{\ell_i,2}y + c_{\ell_i,3}t + d_{\ell_i} \geq 0$ for $i \in \{0, 1\}$. Then the approach encodes initialiation*

and consecution by the tabular form in Table 1. For the initialization, we have

$$\begin{array}{c|cccc} \lambda_0 & & & 1 & \geq 0 \\ \lambda_1 & x & & & = 0 \\ \lambda_2 & & y & & = 0 \\ \lambda_3 & & & t & = 0 \\ \hline & c_{\ell_0,1}x + c_{\ell_0,2}y + c_{\ell_0,3}t + d_{\ell_0} & & & \geq 0 \leftarrow \eta(\ell_0) \end{array} \quad \theta$$

resulting in the constraints $\left[c_{\ell_0,1} = \lambda_1, c_{\ell_0,2} = \lambda_2, c_{\ell_0,3} = \lambda_3, d_{\ell_0} \geq 0\right]$. *After projecting away the fresh variables* λ_j*'s, we obtain* $[d_{\ell_0} \geq 0]$ *for the initialization. For the consecution conditions, we present the tabular form for the transition* τ_1*:*

$$\begin{array}{c|cccccccc} \mu & c_{\ell_0,1}x+c_{\ell_0,2}y+c_{\ell_0,3}t & & & & & & + d_{\ell_0} \geq 0 \leftarrow \eta(\ell_0) \\ \lambda_0 & & & & & & & 1 \geq 0 \\ \lambda_1 & -x & + & t+ & x' & - & t' & \geq 0 \\ \lambda_2 & x & - & 2t- & x' & + & 2t' & \geq 0 \\ \lambda_3 & & -y- & 2t & + & y'+ & 2t' & \geq 0 \\ \lambda_4 & & y+ & t & - & y'- & t' & \geq 0 \\ \lambda_5 & & - & t & & & t'- 1 & \geq 0 \\ \lambda_6 & & & t & & - & t'+ 2 & \geq 0 \\ \hline & & & & & & c_{\ell_1,1}x'+c_{\ell_1,2}y'+c_{\ell_1,3}t'+d_{\ell_1} & \geq 0 \leftarrow \eta(\ell_1)' \end{array} \quad \rho_1$$

where the fresh variables λ_j*'s are local to the tabular above and does not overlap with the tabulars for other transitions. The* λ_j*'s are again eliminated by polyhedron projection, while the fresh variable* μ *is eliminated by heuristically guessing its values. The transition* τ_2 *is treated in the similar way.*

After applying Farkas' Lemma to the initialization and consecution conditions, the resultant constraint is a CNF formula where each atomic proposition is a polyhedral cone over the unknown coefficients. The approach further expands the CNF equivalently into a DNF formula. For this example, one disjunctive clause from the DNF formula is as follows (where we abbreviate $c_{\ell_i,j}$ *as* c_{ij}*,* d_{ℓ_i} *as* d_i*):*

$$\begin{bmatrix} c_{01}=c_{11}, & c_{02}=c_{12}, & c_{03}=c_{13}, & d_0 \geq 0, \\ 2c_{01}-d_0+c_{12}+c_{13}+d_1 \geq 0, & 2c_{01}-d_0+2c_{12}+c_{13}+d_1 \geq 0, \\ 4c_{01}-d_0+4c_{12}+2c_{13}+d_1 \geq 0, & c_{01}-d_0+c_{12}+c_{13}+d_1 \geq 0, \\ c_{01}-d_0+2c_{12}+c_{13}+d_1 \geq 0, & 2c_{01}-d_0+4c_{12}+2c_{13}+d_1 \geq 0, \\ 2c_{01}+d_0-4c_{12}+2c_{13}-d_1 \geq 0, & 4c_{01}+d_0-4c_{12}+2c_{13}-d_1 \geq 0, \\ c_{01}+d_0-2c_{12}+c_{13}-d_1 \geq 0, & 2c_{01}+d_0-2c_{12}+c_{13}-d_1 \geq 0, \\ c_{01}+d_0-c_{12}+c_{13}-d_1 \geq 0, & 2c_{01}+d_0-c_{12}+c_{13}-d_1 \geq 0 \end{bmatrix} \tag{1}$$

By computing the generators of the polyhedral cone above, we obtain the following generators and their corresponding invariants. Table 2 gives the result where (i) in the left part each row specifies a generator (over the unknown coefficients c_{ij}, d_i*'s) and (ii) in the right part we instantiate the generator to the unknown coefficients in the template* η *to obtain the invariants at locations* ℓ_0 *and* ℓ_1*.*

Table 2. Generators (left) and Their Invariants (right) for (1)

c_{01}	c_{02}	c_{03}	d_0	c_{11}	c_{12}	c_{13}	d_1	$\eta(\ell_0)$	$\eta(\ell_1)$
0	0	0	1	0	0	0	1	$1 \geq 0$	$1 \geq 0$
0	0	0	0	0	0	0	0	$0 \geq 0$	$0 \geq 0$
−1	0	2	0	−1	0	2	0	$-x+2t \geq 0$	$-x+2t \geq 0$
0	−1	1	0	0	−1	1	2	$-y+t \geq 0$	$-y+t+2 \geq 0$
0	−1	2	0	0	−1	2	0	$-y+2t \geq 0$	$-y+2t \geq 0$
0	0	1	0	0	0	1	1	$t \geq 0$	$t+1 \geq 0$
0	1	2	0	0	1	2	0	$y+2t \geq 0$	$y+2t \geq 0$
0	1	1	0	0	1	1	−2	$y+t \geq 0$	$y+t-2 \geq 0$
1	0	−1	0	1	0	−1	0	$x-t \geq 0$	$x-t \geq 0$
0	0	1	0	0	0	1	−1	$t \geq 0$	$t-1 \geq 0$

The obtained invariants are further minimized, so that the final invariants obtained at $\eta(\ell_0)$ are

$$\begin{bmatrix} -x+2t \geq 0 & -y+t \geq 0 & x-t \geq 0 & y+t \geq 0 \end{bmatrix}$$

and for $\eta(\ell_1)$ the invariants are

$$\begin{bmatrix} -x+2t \geq 0 & -y+t+2 \geq 0 & -y+2t \geq 0 \\ t-1 \geq 0 & y+t-2 \geq 0 & x-t \geq 0 \end{bmatrix}$$

It happens that the final invariants from the whole DNF formula coincide with the invariants computed from (1). □

3.2 The Main Idea of Our Approach

A drawback of the approach in [65] is that in the last step of the approach (cf. **Step 4** in the previous subsection), the invariants are obtained as the generators of a whole polyhedral cone that involves all the unknown coefficients at all locations. This leads to two obstacles that may largely increase the runtime of the invariant generation process. The first obstacle is that in the subsumption testing, there is only a small chance that a whole polyhedral cone with all the unknown coefficients may be subsumed by the invariants already generated. The second obstacle is that the generator computation of a polyhedral cone is an expensive operation since it may cause exponential blowup in the number of its variables, thus to compute the generators of a whole polyhedral cone that involves all the unknown coefficients may induce a huge amount of runtime.

To address the two obstacles, we propose the novel insight of generating the invariants one location at a time (which we call the *location-by-location* insight). With this insight, the chance of subsumption is increased since now the subsumption testing involves only one location. Moreover, the amount of runtime to compute the generators may also be reduced since we only need to handle a much smaller number of unknown coefficients (to be more precise, only the unknown coefficients at the location of concern). Note that although the insight seems simple, it can indeed lead to dramatic improvement on the runtime over realistic examples.

Below we present an example to illustrate the location-by-location insight.

Example 3. *Continue with Example 2. Recall that in the last step of the approach [65], we obtain a DNF where each disjunctive clause corresponds to a polyhedral cone over all the unknown coefficients, and compute the generators over each such polyhedral cone. To implement the location-by-location insight, we focus on the location ℓ_0 and run **Steps 1–3** in the previous subsection. Then in **Step 4**, our insight works as follows. Recall that in **Step 4**, the approach [65] expands the CNF into its equivalent DNF, and computes the generators for each atomic proposition (polyhedral cone) of the DNF. Suppose*

that we need to compute the generators of the polyhedral cone specified by (1) only at the location ℓ_0. One way to implement the insight is to project the polyhedral cone onto the dimensions specified by the unknown coefficients c_{0j}'s ($1 \le j \le 3$) and d_0, to obtain the following polyhedral cone

$$\begin{bmatrix} d_0 \ge 0 & c_{01} - c_{02} + c_{03} \ge 0 & c_{01} + c_{02} + c_{03} \ge 0 \\ & 2c_{01} - c_{02} + c_{03} \ge 0 & 2c_{01} + c_{02} + c_{03} \ge 0 \end{bmatrix}.$$

The generators of this projected polyhedral cone gives the following linear invariants at the location ℓ_0:

c_{01}	c_{02}	c_{03}	d_0	$\eta(\ell_0)$
0	0	0	1	$1 \ge 0$
0	0	0	0	$0 \ge 0$
1	0	−1	0	$x - t \ge 0$
0	1	1	0	$y + t \ge 0$
−1	0	2	0	$-x + 2t \ge 0$
0	−1	1	0	$-y + t \ge 0$

After minimization, the invariants added for the location ℓ_0 from the polyhedral cone (1) include

$$\begin{bmatrix} -x + 2t \ge 0 & -y + t \ge 0 & x - t \ge 0 & y + t \ge 0 \end{bmatrix}.$$

*which happen to include all the linear inductive invariants at ℓ_0. Note that we only generate the invariants at ℓ_0 in one invariant-generation process, and keep the invariants at other locations (i.e., ℓ_1) to be **true**. By another invariant-generation process for the location ℓ_1, we get the invariants at the location ℓ_1 as follows:*

$$\begin{bmatrix} -x + 2t \ge 0 & -y + t + 2 \ge 0 & -y + 2t \ge 0 \\ t - 1 \ge 0 & y + t - 2 \ge 0 & x - t \ge 0 \end{bmatrix}.$$

The invariants obtained coincide with the approach in [65]. □

4 THE LOCATION-BY-LOCATION INSIGHT

In this section, we formally demonstrate our location-by-location insight, i.e., generating the invariant only at one location in a single invariant-generation process. The insight is composed of two ingredients. The first one is the reordering of the expansion from the CNF into its equivalent DNF at **Step 4** (Section 3.1) so that the transitions associated with the location of concern is expanded first to maximize the chance of subsumption. The second one is the computation of the generators of each atomic proposition (as a polyhedral cone) in the DNF obtained at **Step 4** that only involve the unknown coefficients at the location of concern. Below we demonstrate the two ingredients in detail. We fix a LinTS and a location $\ell_\star$ on which invariant generation is considered.

4.1 Expanding $\ell_\star$ First from CNF to DNF

Recall that in **Step 4**, we obtain a CNF from which we extract invariants. To facilitate the location-by-location insight, we maximize the chance of subsumption by reordering the expansion from the CNF into its DNF as follows, which is the first ingredient in our location-by-location insight.

Reordering. One motivation of the location-by-location insight aims at increasing the chance of successfull subsumption in **Step 4** by performing the subsumption test at $\ell_\star$ only. To maximize such chance, we reorder the expansion from the CNF to its DNF so that the transitions of the LinTS associated with $\ell_\star$ are expanded first, and the other transitions are expanded last. The reason to expand the transitions with the target location $\ell_\star$ first is as follows: since we focus on the invariant generation at $\ell_\star$, successful subsumptions would be detected earlier if we check transitions that involve the target location $\ell_\star$ first. In contrast, if one chooses an arbitary order for

the expansion, then the expansion at locations other than $\ell_\star$ cannot lead to successful subsumption, as the subsumption testing is operated by checking the polyhedral inclusion that only involves the unknown coefficients at $\ell_\star$ (i.e., not involving the unknown coefficients at other locations). We refine the epansion order by having that the *intra*-transitions in the LinTS that involves only the location $\ell_\star$ (i.e., the transitions from $\ell_\star$ to itself) are expanded first, those that involves $\ell_\star$ and other locations (i.e., the *inter*-transitions from $\ell_\star$ to other locations or from other locations to $\ell_\star$) are expanded second, those that does not involve location $\ell_\star$ are expanded last.

The pseudo-code for our invariant generation algorithm that integrates the location-by-location insight is given in Algorithm 1. In the pseudo-code, the algorithm first reorders the CNF formula $\bigwedge_i C_i$ obtained from **Step 4** in Section 3.1 as stated in the previous paragraph (line 1). The reordered CNF is $\bigwedge_i C'_i$. Then the algorithm initializes the values for $inv(\ell_\star), i$ and all n_i's (lines 2–4): the variable $inv(\ell_\star)$ is a set variable used to collect linear invariants generated in the algorithm and initialized to be $\emptyset$; since we implement expansion from the reordered CNF to its DNF through backtracking, we use the variable i to represent that the algorithm is currently traversing the atomic propositions in C'_i, and the variable n_i to represent that the current atomic proposition being traversed in C'_i is the n_i-th atomic proposition. Next, the algorithm follows the backtracking process that iteratively selects one atomic proposition d_i from each C'_i (line 7) following the increasing order of i, aggressively checks whether the current exploration has already been subsumed by the current invariant $inv(\ell_\star)$ (line 9), and updates the current invariant set by adding the invariants *only* at $\ell_\star$ derived from the conjunction $\bigwedge_{s=1}^{m} d_s$ (as a polyhedral cone) through the **Gen** procedure if all the preceding subsumption testings have been unsuccessful. Note that in the subsumption testing, the condition $\bigwedge_{s=1}^{i} d_s \not\subseteq inv(\ell_\star)$ means that the polyhedral cone defined by the linear assertion $\bigwedge_{s=1}^{i} d_s$ is not a subset of the counterpart generated by the linear invariants (as generators extended with full dimension at other locations) in $inv(\ell_\star)$. The return value is the final set value of $inv(\ell_\star)$ after the whole backtracking process, which collects all the linear invariants that are generators of the polyhedral cones that pass the subsumption testing in the algorithm. A missing part in the pseudo-code is the details for the procedure **Gen**, which will be illustrated in the next subsection as the second ingredient.

4.2 Detailed Implementation of the Procedure Gen

To complete the pseudo-code in Algorithm 1, we give the details for the procedure **Gen**, which is the second ingredient in the location-by-location insight. This procedure generates the linear invariants at the target location $\ell_\star$ from an input polyhedron *poly* over all the unknown coefficients.

There are various solutions to implement the procedure **Gen**. The first solution is still to compute the generators of *poly*, but to project the computed generators onto the unknown coefficients at the location $\ell_\star$, which we refer as **No-Proj** (i.e., no polyhedron projection). The second solution is to apply the classical method of Fourier-Motzkin Elimination (FME) [66, Chapter 12.2] that first projects away the extra dimensions other than the unknown coefficients at the target location $\ell_\star$ and then computes the generators. In this work, we consider Kohler's approach [51] as a third solution.

Kohler's Approach. A problem of the FME method is that it may produce many redundant linear inequalities that significantly increase the amount of time required to compute projection. To avoid introducing redundant inequalities, Kohler [51] proposed an alternative approach through the computation of the generators of the *projection cone* of a polyhedron. To illustrate this approach, we first present a variant form of Farkas' Lemma [66, Corollary 7.1e].

Algorithm 1 Linear Invariant Generation at Location $\ell_\star$

Input: $\ell_\star$: target location;
$\bigwedge_{i=1}^{m} C_i$: the CNF formula obtained from **Step 4** (Section 3.1) in the approach [20] where each C_i is the i-th conjunctive clause and is a disjunction of polyhedra over the unknown coefficients in the template;
$\textsc{Reordering}(\bigwedge_i C_i)$: a procedure that reorders all C_i's into $\bigwedge_i C'_i$ w.r.t whether each C_i corresponds to an intra- or an inter-transition as stated previously; we write $C'_i = \bigvee_{j=1}^{N_i} c'_{i,j}$ where each $c'_{i,j}$ is an atomic proposition in the form of a polyhedron over the unknown coefficients;
$\textsc{Gen}(\ell, poly)$: a procedure that generates the invariants at a location ℓ given a polyhedron $poly$ over the unknown coefficients
Output: a collection $inv(\ell_\star)$ of linear invariants at $\ell_\star$;

1: $\bigwedge_{i=1}^{m} C'_i \leftarrow \textsc{Reordering}(\bigwedge_{i=1}^{m} C_i)$; //$C'_i = \bigvee_{j=1}^{N_i} c'_{i,j}$
2: $inv(\ell_\star) \leftarrow \emptyset$;
3: $i \leftarrow 1$;
4: $n_i \leftarrow 1$ (for $1 \le i \le m$);
5: **while** $i > 0$ **do**
6: **if** $n_i \le N_i$ **then**
7: $d_i \leftarrow c'_{i,n_i}$; //select one atomic proposition
8: $n_i \leftarrow n_i + 1$;
9: **if** $\bigwedge_{s=1}^{i} d_s \not\subseteq inv(\ell_\star)$ **then** //subsumption
10: **if** $i = m$ **then**
11: $inv(\ell_\star) \leftarrow inv(\ell_\star) \cup \textsc{Gen}(\ell_\star, \bigwedge_{s=1}^{m} d_s)$;
12: **else**
13: $i \leftarrow i + 1$;
14: **end if**
15: **end if**
16: **else**
17: $n_i \leftarrow 1$; //backtracking
18: $i \leftarrow i - 1$;
19: **end if**
20: **end while**;
21: **return** $inv(\ell_\star)$;

Theorem 4.1 (Farkas' Lemma Variant). *Let* $\mathbf{A}$ *be a real matrix and* $\mathbf{b}$ *be a real vector. The polyhedron* $P = \{\mathbf{x} \mid \mathbf{A}\mathbf{x} \le \mathbf{b}\}$ *is non-empty if and only if there is no vector* $\mathbf{y}$ *satisfying* $\mathbf{y} \ge \mathbf{0}$, $\mathbf{y}^{\mathrm{T}}\mathbf{A} = \mathbf{0}$ *and* $\mathbf{y}^{\mathrm{T}}\mathbf{b} < 0$.

The polyhedral cone $\{\mathbf{y} \mid \mathbf{y} \ge \mathbf{0} \wedge \mathbf{y}^{\mathrm{T}}\mathbf{A} = \mathbf{0}\}$ derived from the polyhedron $P = \{\mathbf{x} \mid \mathbf{A}\mathbf{x} \le \mathbf{b}\}$ in the statement of Theorem 4.1 is called the *projection cone* of P, which we denoted by $proj(\mathbf{A})$ since it is only related to the matrix $\mathbf{A}$.

Kohler's approach utilizes projection cones to reduce the amount of redundant inequalities. Consider the polyhedron $P = \{(\mathbf{x}^{\mathrm{T}}, \mathbf{u}^{\mathrm{T}})^{\mathrm{T}} \in \mathbb{R}^{p+q} \mid \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \le \mathbf{c}\}$ and the projection of P onto $\mathbf{x}$. The approach treats P as a polyhedron $Q(\mathbf{x})$ with the parameter $\mathbf{x}$ such that $Q(\mathbf{x}) := \{\mathbf{u} \in \mathbb{R}^q \mid \mathbf{B}\mathbf{u} \le \mathbf{c} - \mathbf{A}\mathbf{x}\}$ and computes the generators of the projection cone $proj(\mathbf{B})$. By Theorem 4.1, to ensure that $Q(\mathbf{x})$ is non-empty, it suffices to guarantee that $\mathbf{g}^{\mathrm{T}}(\mathbf{c} - \mathbf{A}\mathbf{x}) \ge 0$ for all the generators $\mathbf{g}$ of $proj(Q(\mathbf{x}))$. Hence, after the generators $\mathbf{g}_1, \ldots, \mathbf{g}_k$ of the projection cone $proj(Q(\mathbf{x}))$ are computed, the approach

outputs the projected polyhedron $P[\mathbf{x}]$ as defined by the linear inequalities $\mathbf{g}_j^{\mathrm{T}}(\mathbf{c} - \mathbf{A}\mathbf{x}) \geq 0$ for $1 \leq j \leq k$.

Our improvement on Kohler's approach. In our situation, the task is to compute the projected generators of a polyhedral cone $\mathbf{A}\mathbf{c} \leq \mathbf{0}$ onto the unknown coefficients at $\ell_\star$ (here $\mathbf{c}$ is the vector of unknown coefficients in the template). Following Kohler's approach, we project the polyhedral cone $\mathbf{A}\mathbf{c} \leq \mathbf{0}$ onto the unknown coefficients at $\ell_\star$, and improve the approach by exploring the detailed structure of the polyhedral cone. Let $\mathbf{c}_\star$ be the vector of unknown coefficients at $\ell_\star$ and $\mathbf{c}_*$ be the vector of unknown coefficients at other locations. Then $\mathbf{A}\mathbf{c} \leq \mathbf{0}$ can be decomposed as

$$\mathbf{A}\mathbf{c} = \begin{pmatrix} \mathbf{F} & \mathbf{0} \\ \mathbf{G} & \mathbf{G}' \\ \mathbf{0} & \mathbf{H} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{c}_* \\ \mathbf{c}_\star \end{pmatrix} \leq \mathbf{0}$$

where $\mathbf{F}\mathbf{c}_* \leq \mathbf{0}$ include the inequalities that involve only variables in $\mathbf{c}_*$, $\mathbf{G}\mathbf{c}_* + \mathbf{G}'\mathbf{c}_\star \leq \mathbf{0}$ include the inequalities that involve both $\mathbf{c}_*$ and $\mathbf{c}_\star$, and $\mathbf{H}\mathbf{c}_\star \leq \mathbf{0}$ include the inequalities that involve only variables in $\mathbf{c}_\star$. Since $\mathbf{H}\mathbf{c}_\star \leq \mathbf{0}$ is not related to $\mathbf{c}_*$, our first (slight) improvement is to consider only

$$\mathbf{B}\mathbf{c}_* \leq \mathbf{b} \text{ with } \mathbf{B} := \begin{pmatrix} \mathbf{F} \\ \mathbf{G} \end{pmatrix} \text{ and } \mathbf{b} := \begin{pmatrix} \mathbf{0} \\ -\mathbf{G}' \cdot \mathbf{c}_\star \end{pmatrix} \tag{2}$$

in the projection onto $\mathbf{c}_\star$. Furthermore, in the projection cone $\mathit{proj}(\mathbf{B})$:

$$\begin{pmatrix} \mathbf{y}_* \\ \mathbf{y}_\star \end{pmatrix} \geq \mathbf{0},\ (\mathbf{y}_*^{\mathrm{T}},\ \mathbf{y}_\star^{\mathrm{T}}) \cdot \begin{pmatrix} \mathbf{F} \\ \mathbf{G} \end{pmatrix} = \mathbf{0}, \tag{3}$$

we only need to consider the projection onto the dimensions $\mathbf{y}_\star$ that correspond to the rows of $\mathbf{G}$ since the vector $\mathbf{b}$ in (2) has all zero at the dimensions $\mathbf{y}_*$. Thus, our second (major) improvement is to project the projection cone $\mathit{proj}(\mathbf{B})$ onto $\mathbf{y}_\star$, and then compute the generators of the projected cone.

4.3 The Accuracy of Our Approach

The accuracy of the generated invariants from our location-by-location insight can be shown by proving that our approach generates at least the invariants from the most related previous approach [65] that our approach is based on. This can be observed from the two points as follows. First, the invariant generation at the target location $\ell_\star$ outputs correctly the projected generators at $\ell_\star$. Second, any disjunctive clause Q (in the form of a polyhedral cone) in the final DNF that passes all the preceding subsumption testing in the previous approach [65] is subsumed by the polyhedral cone $P_\star$ generated by the linear invariants (as generators) in the final result $\mathit{inv}(\ell_\star)$ produced by our approach. This can be easily seen as follows: for any such disjunctive clause Q, if it does not pass the subsumption testing in our new approach, then naturally we have that $Q' \subseteq P_\star$ where Q' is the projection of Q onto the unknown coefficients at $\ell_\star$ (i.e., projection preserves set inclusion); otherwise, the algorithm adds the generators of Q' into $\mathit{inv}(\ell_\star)$ so that the final value of $\mathit{inv}(\ell_\star)$ still include the projected generators of Q onto $\ell_\star$.

5 EXPERIMENTAL RESULTS

Implementation. We implemented our algorithms as an extension of StInG [70] in C++, and used PPL 1.2 [7] for polyhedra computation. To have a prototype implementation that synthesizes linear invariants directly over programs, we wrote the benchmark examples in C programming language and implemented the transformation from these examples into their format in StInG [70] via Sparse 0.6.4 [69] (a C language semantic parser). All results were obtained on an Intel Core i7-7700 (3.6 GHz) machine with 15.5 GiB of memory, running Ubuntu 20.04.2 LTS.

Table 3. Experimental Results for Reordering Expansion

Benchmarks					StInG	Our Approach			
Name		Loc	Dim	Line		Expan-Order A: No-Proj		Expan-Order B: No-Proj	
					Time (Banged)	Time (Banged)	Speedup	Time (Banged)	Speedup
Origin Scheduler	2p	2	16	-	0.01 (26)	0.02 (34)	0.50X	0.01 (34)	1.00X
	3p	3	33	-	0.17 (380)	0.12 (289)	1.41X	0.10 (245)	1.70X
	4p	4	56	-	60.81 (26,629)	2.92 (4,137)	20.82X	1.38 (1,508)	44.06X
	5p	5	85	-	7,436.34 (2,548,704)	228.68 (219,735)	32.51X	26.52 (25,996)	280.40X
	6p	6	120	-	>36,000.00 (>1,685,618)	>36,000.00 (>1,427,726)	-	892.75 (66,905)	>40.32X
Fixed Scheduler	3p	3	33	336	0.17 (279)	0.11 (305)	1.54X	0.10 (261)	1.70X
	4p	4	56	609	4.16 (2,895)	2.46 (4,184)	1.69X	0.98 (1,474)	4.24X
	5p	5	85	1017	135.80 (39,150)	227.16 (141,952)	0.59X	13.97 (8,570)	9.72X
	6p	6	120	1587	7,541.53 (906,454)	28,890.44 (6,850,492)	0.26X	277.96 (63,070)	27.13X
Fischer	6p	7	63	710	9.18 (8,423)	4.35 (4,167)	2.11X	3.07 (4,217)	2.99X
	7p	8	80	987	59.16 (32,668)	28.50 (12,494)	2.07X	13.60 (12,566)	4.35X
	8p	9	99	1327	373.62 (127,918)	139.83 (43,093)	2.67X	70.42 (43,191)	5.30X
	9p	10	120	1736	2,345.96 (503,369)	1,194.47 (163,495)	1.96X	398.26 (163,623)	5.89X
	10p	11	143	2218	14,664.68 (1,985,857)	Out of Memory (-)	-	2,361.87 (649,409)	6.20X
	11p	12	168	2780	>36,000.00 (>3,347,744)	Out of Memory (-)	-	14,307.40 (2,620,864)	>2.51X
Fixed Cars	2p	3	27	216	0.01 (28)	0.02 (45)	0.50X	0.01 (63)	1.00X
	3p	5	60	616	560.70 (788,508)	0.39 (659)	1,437.69X	1.21 (2,299)	463.38X
	4p	7	105	1283	>36,000.00 (>11,727,189)	45.39 (6,841)	>793.12X	86.83 (37,567)	>414.60X

Benchmarks. We consider the following benchmarks from a variety of application domains [5, 52, 65, 70, 71]:

- *Scheduler.* The invariant analysis for cooperative multitasking (task scheduling) activated by interrupts and pre-emptive programming can be used to ensure the liveness of scheduling. In these benchmarks, taken from Arduino [5] and Sankaranarayanan [65], the goal is to generate invariants on cooperative multitasking with multiple threads. The benchmarks here are divided into two categories, namely "Origin Scheduler" and "Fixed Scheduler". The category "Origin Scheduler" considers a non-standard setting to assign an initial condition to every location in the LinTS in order to model the scenario that the initial location is nondeterministic, and the benchmarks are directly in the format of StInG. The category "Fixed Scheduler" considers the orignal benchmarks from StInG but under the standard setting of a prescribed initial location, fixes several places in the original benchmarks that deviate from the original description in [41], and the benchmarks are written in C.
- *Fischer.* Fischer mutual exclusion protocol [52] is a timing-based algorithm that implements mutual exclusion for a distributed system with skewed clocks, and the invariant analysis of this can be used to adjust some parameters to ensure reachability properties. The benchmarks here are taken from PAT [71] and rewritten in C.
- *Cars.* The cars detection is a dynamic decision problem [65] whose invariant analysis can be used to ensure the safety property of autopilot. The cars system has n cars on a straight road and the acceleration and velocity are determining by their controllers. The lead car non-deterministically starts at an arbitary acceleration with initial velocity 0. The controllers of other cars detect whether the distance between the front car and itself is too close or too far, and adjust their acceleration accordingly. We take the benchmarks from [65], rewrite them equivalently into multi-location versions, and implement them in C. We name the category of these benchmarks as "Fixed Cars".

For each benchmark above, we consider a variety on the number r of processes (denoted by "r-p", e.g. "2p") involved. For all the benchmarks, we compare the running time between our algorithms and the original algorithm in StInG.

Results. Our experimental results are sumarized in Table 3– 5, where we set a bound of 10 hours for time-out. In the tables, we have that "Loc" means the number of locations in each benchmark,

Table 4. Experimental Results for Polyhedron Projection

Benchmarks		Our Approach			
Name		Time (Expan-Order B)			
		No-Proj	FME	Kohler	Kohler*
Origin Scheduler	2p	0.01	0.01	0.02	0.02
	3p	0.06	0.08	0.27	0.08
	4p	0.24	0.52	1.42	0.46
	5p	0.38	1.32	4.03	1.07
	6p	38.59	48.88	28.01	11.20
	7p	62,852.98	67,350.57	182.71	114.64
Fixed Scheduler	3p	0.05	0.09	0.16	0.08
	4p	0.16	0.41	1.20	0.34
	5p	1.40	2.12	5.71	1.63
	6p	36.54	46.59	23.98	6.60
	7p	67,671.98	72,406.55	156.64	80.51

Table 5. Experimental Results for Parallel Computation

Benchmarks					Time (Expan-Order B: Kohler*)		
Name		Loc	Dim	Line	Max	Speedup(original)	Speedup
Origin Scheduler	2p	2	16	-	0.01	0.50X	1.00X
	3p	3	33	-	0.05	1.21X	3.40X
	4p	4	56	-	0.47	40.54X	129.38X
	5p	5	85	-	7.52	272.69X	988.87X
	6p	6	120	-	308.48	>41.44X	>116.70X
	7p	7	161	-	5,424.19	>1.60X	>6.63X
Fixed Scheduler	3p	3	33	336	0.04	1.41X	4.25X
	4p	4	56	609	0.39	3.25X	10.66X
	5p	5	85	1017	4.90	8.58X	27.71X
	6p	6	120	1587	72.07	31.35X	104.64X
	7p	7	161	2352	1,350.66	>7.57X	>26.65X
Fischer	6p	7	63	710	1.22	2.91X	7.52X
	7p	8	80	987	7.75	3.96X	7.63X
	8p	9	99	1327	45.41	5.25X	8.22X
	9p	10	120	1736	283.95	5.66X	8.26X
	10p	11	143	2218	1,700.51	6.11X	8.62X
	11p	12	168	2780	10,230.51	>2.51X	>3.51X
Fixed Cars	2p	3	27	216	0.01	1.00X	1.00X
	3p	5	60	616	0.83	463.38X	675.54X
	4p	7	105	1283	75.83	>426.08X	>474.74X

"Dim" means the number of total dimensions of the benchmark, "Line" means the number of lines of the implemented program in C, "Time" is the amount of runtime measured in seconds, "Banged" means the number of successful subsumptions, and "Speedup" shows the ratio of the amount of runtime consumed by StInG against ours, "Our Approach" means the experimental results by our approach, "StInG" means the experimental results by StInG, and the symbol "-" means that the cell in the table is not applicable. Furthermore, under "Our Approach" with the set of locations L and the target location $\ell_\star$, the tag "Expan-Order A" means to make the intra-transitions on $L\backslash\{\ell_\star\}$ in front of the inter-transitions between $L\backslash\{\ell_\star\}$ when reordering expansion, the tag "Expan-Order B" means that making the inter-transitions between $L\backslash\{\ell_\star\}$ in front of the intra-transition on $L\backslash\{\ell_\star\}$ when reordering expansion, "No-Proj" means that computing the generators without any projection method in procedure **Gen**, "FME" means that computing the generators by applying projection of Fourier-Motzkin Elimination, "Kohler" means that computing the generators by applying Kohler's approach, and "Kohler* " means that computing the generators by applying Kohler's approach with our improvement. In all the benchmarks, the generated invariants by our approach are exactly the same as from StInG, thus we focus on the comparison in the runtime. Due to the limit of space, we only present the most representative experimental results and relegate the other results in Appendix A.

Table 3 presents the runtime for StInG, our approach with "Expan-Order A" and "No-Proj" and our approach with "Expan-Order B" and "No-Proj", for which the amount of runtime for our approach is taken as the summation of all locations in the LinTS. From the table, we see that our approach implemented with both expansion order A and B and without polyhedron projection for the generator computation can improve the runtime performance of StInG up to a magnitude of hundreds. Moreover, Expansion order B stably improves the performance as it improves the runtime in all cases. This demonstrates that our improved expansion order from the CNF to its DNF is indeed effective. The effectiveness is also supported by the number of successful subsumptions, as one can observe that a smaller number of successful subsumptions means that these subsumptions are detected earlier by our improved expansion order.

Table 4 records the total amount of time consumed by the generator computation under our improved expansion order. From the table, one can observe that "No-Proj" is the best when the dimension is moderate, while our improvement on Kohler's approach out-performs all other methods when the dimension is high. This shows that our improved Kohler's approach is effective when the dimension in generation computation is high.

Finally, Table 5 checks the advantage of parallel computation through our approach. Recall that by the location-by-location insight, our approach allows to compute the invariants at each location separately. In Table 5, we present the maximum running time under our improved Kohler with parallelism. The "Max" column records the maximum running time over all locations. Compared with the "Speedup(original)" column which records the summation of the running time under our approach at all locations (i.e., without parallelism), our approach with parallel computation achieves much better speed-up on the runtime performance.

6 CONCLUSION AND FUTURE WORK

In this work, we proposed a new approach based on the previous approach [65] through constraint solving and Farkas' Lemma, with the improvement from the location-by-location insight that handles only one (target) program location in a single invariant-generation process. To facilitate the insight, we considered a reordered expansion from a key CNF formula in the invariant generation process into its equivalent DNF form to maximize the success chance of subsumption; we also considered variants of generator computation that generate the invariants at the target location of concern; in addition, the location-by-location insight allows a speed-up through parallel

computation that computes the invariants at the locations separately over multiple processors. Experimental results show that our novel insight can dramatically improve the efficiency of linear invariant generation as compared with the previous approach [65]. A future direction would be to incorporate dynamic programming that reuses intermediate results in the expansion from the CNF into its DNF. Another possible future direction is to integrate other methods (e.g. [9, 46]) on polyhedron projection.

REFERENCES

[1] Assalé Adjé, Pierre-Loïc Garoche, and Victor Magron. 2015. Property-based Polynomial Invariant Generation Using Sums-of-Squares Optimization. In *SAS (LNCS, Vol. 9291)*. Springer, 235–251.

[2] Assalé Adjé, Stéphane Gaubert, and Eric Goubault. 2012. Coupling policy iteration with semi-definite relaxation to compute accurate numerical invariants in static analysis. *Log. Methods Comput. Sci.* 8, 1 (2012). https://doi.org/10.2168/LMCS-8(1:1)2012

[3] Aws Albarghouthi, Yi Li, Arie Gurfinkel, and Marsha Chechik. 2012. Ufo: A Framework for Abstraction- and Interpolation-Based Software Verification. In *CAV (LNCS, Vol. 7358)*. Springer, 672–678. https://doi.org/10.1007/978-3-642-31424-7_48

[4] Christophe Alias, Alain Darte, Paul Feautrier, and Laure Gonnord. 2010. Multi-dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs. In *SAS (LNCS, Vol. 6337)*. Springer, 117–133. https://doi.org/10.1007/978-3-642-15769-1_8

[5] Arduino [n.d.]. Arduino: An open-source electronics platform based on easy-to-use hardware and software. https://github.com/arkhipenko/TaskScheduler.

[6] Ali Asadi, Krishnendu Chatterjee, Hongfei Fu, Amir Kafshdar Goharshady, and Mohammad Mahdavi. 2021. Polynomial reachability witnesses via Stellensätze. In *PLDI*. ACM, 772–787. https://doi.org/10.1145/3453483.3454076

[7] Roberto Bagnara, Elisa Ricci, Enea Zaffanella, and Patricia M. Hill. 2002. Possibly Not Closed Convex Polyhedra and the Parma Polyhedra Library. In *SAS (Lecture Notes in Computer Science, Vol. 2477)*. Springer, 213–229. https://doi.org/10.1007/3-540-45789-5_17

[8] Roberto Bagnara, Enric Rodríguez-Carbonell, and Enea Zaffanella. 2005. Generation of Basic Semi-algebraic Invariants Using Convex Polyhedra. In *SAS (LNCS, Vol. 3672)*. Springer, 19–34. https://doi.org/10.1007/11547662_4

[9] Egon Balas. 1998. Projection with a Minimal System of Inequalities. *Comput. Optim. Appl.* 10, 2 (1998), 189–193. https://doi.org/10.1023/A:1018368920203

[10] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. 2005. Linear Ranking with Reachability. In *CAV (LNCS, Vol. 3576)*. Springer, 491–504. https://doi.org/10.1007/11513988_48

[11] Jason Breck, John Cyphert, Zachary Kincaid, and Thomas W. Reps. 2020. Templates and recurrences: better together. In *PLDI*. ACM, 688–702. https://doi.org/10.1145/3385412.3386035

[12] Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *CAV (LNCS, Vol. 8044)*. Springer, 511–526. https://doi.org/10.1007/978-3-642-39799-8_34

[13] Aleksandar Chakarov and Sriram Sankaranarayanan. 2014. Expectation Invariants for Probabilistic Program Loops as Fixed Points. In *SAS (LNCS, Vol. 8723)*, Markus Müller-Olm and Helmut Seidl (Eds.). Springer, 85–100. https://doi.org/10.1007/978-3-319-10936-7_6

[14] Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. 2019. Non-polynomial Worst-Case Analysis of Recursive Programs. *ACM Trans. Program. Lang. Syst.* 41, 4 (2019), 20:1–20:52. https://doi.org/10.1145/3339984

[15] Krishnendu Chatterjee, Hongfei Fu, Amir Kafshdar Goharshady, and Ehsan Kafshdar Goharshady. 2020. Polynomial invariant generation for non-deterministic recursive programs. In *PLDI*. ACM, 672–687. https://doi.org/10.1145/3385412.3385969

[16] Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. 2018. Algorithmic Analysis of Qualitative and Quantitative Termination Problems for Affine Probabilistic Programs. *ACM Trans. Program. Lang. Syst.* 40, 2 (2018), 7:1–7:45. https://doi.org/10.1145/3174800

[17] Krishnendu Chatterjee, Petr Novotný, and Dorde Zikelic. 2017. Stochastic invariants for probabilistic termination. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*. ACM, 145–160. https://doi.org/10.1145/3009837.3009873

[18] Yu-Fang Chen, Chih-Duo Hong, Bow-Yaw Wang, and Lijun Zhang. 2015. Counterexample-Guided Polynomial Loop Invariant Generation by Lagrange Interpolation. In *CAV (LNCS, Vol. 9206)*. Springer, 658–674. https://doi.org/10.1007/978-3-319-21690-4_44

[19] Yinghua Chen, Bican Xia, Lu Yang, Naijun Zhan, and Chaochen Zhou. 2007. Discovering Non-linear Ranking Functions by Solving Semi-algebraic Systems. In *ICTAC (LNCS, Vol. 4711)*. Springer, 34–49. https://doi.org/10.1007/978-3-540-75292-9_3

[20] Michael Colón, Sriram Sankaranarayanan, and Henny Sipma. 2003. Linear Invariant Generation Using Non-linear Constraint Solving. In *CAV (LNCS, Vol. 2725)*. Springer, 420–432. https://doi.org/10.1007/978-3-540-45069-6_39

[21] Michael Colón and Henny Sipma. 2001. Synthesis of Linear Ranking Functions. In *TACAS (LNCS, Vol. 2031)*. Springer, 67–81. https://doi.org/10.1007/3-540-45319-9_6

[22] Patrick Cousot. 2005. Proving Program Invariance and Termination by Parametric Abstraction, Lagrangian Relaxation and Semidefinite Programming. In *VMCAI (LNCS, Vol. 3385)*. Springer, 1–24. https://doi.org/10.1007/978-3-540-30579-8_1

[23] Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *POPL*. ACM, 238–252. https://doi.org/10.1145/512950.512973

[24] Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. 2005. The ASTREÉ Analyzer. In *ESOP (LNCS, Vol. 3444)*. Springer, 21–30. https://doi.org/10.1007/978-3-540-31987-0_3

[25] Patrick Cousot and Nicolas Halbwachs. 1978. Automatic Discovery of Linear Restraints Among Variables of a Program. In *POPL*. ACM Press, 84–96. https://doi.org/10.1145/512760.512770

[26] Christoph Csallner, Nikolai Tillmann, and Yannis Smaragdakis. 2008. DySy: dynamic symbolic execution for invariant inference. In *ICSE*. ACM, 281–290. https://doi.org/10.1145/1368088.1368127

[27] Cristina David, Pascal Kesseli, Daniel Kroening, and Matt Lewis. 2016. Danger Invariants. In *FM (LNCS, Vol. 9995)*. 182–198. https://doi.org/10.1007/978-3-319-48989-6_12

[28] Steven de Oliveira, Saddek Bensalem, and Virgile Prevosto. 2016. Polynomial Invariants by Linear Algebra. In *ATVA (LNCS, Vol. 9938)*. 479–494. https://doi.org/10.1007/978-3-319-46520-3_30

[29] Steven de Oliveira, Saddek Bensalem, and Virgile Prevosto. 2017. Synthesizing Invariants by Solving Solvable Loops. In *ATVA (LNCS, Vol. 10482)*. Springer, 327–343. https://doi.org/10.1007/978-3-319-68167-2_22

[30] Isil Dillig, Thomas Dillig, Boyang Li, and Kenneth L. McMillan. 2013. Inductive invariant generation via abductive inference. In *OOPSLA*. ACM, 443–456. https://doi.org/10.1145/2509136.2509511

[31] J. Farkas. 1894. A Fourier-féle mechanikai elv alkalmazásai (Hungarian). *Mathematikaiés Természettudományi Értesitö* 12 (1894), 457–472.

[32] Azadeh Farzan and Zachary Kincaid. 2015. Compositional Recurrence Analysis. In *FMCAD*. IEEE, 57–64.

[33] Yijun Feng, Lijun Zhang, David N. Jansen, Naijun Zhan, and Bican Xia. 2017. Finding Polynomial Loop Invariants for Probabilistic Programs. In *ATVA (LNCS, Vol. 10482)*. Springer, 400–416. https://doi.org/10.1007/978-3-319-68167-2_26

[34] Robert W. Floyd. 1967. Assigning meanings to programs. *Mathematical Aspects of Computer Science* 19 (1967), 19–33.

[35] Ting Gan, Bican Xia, Bai Xue, Naijun Zhan, and Liyun Dai. 2020. Nonlinear Craig Interpolant Generation. In *CAV (LNCS, Vol. 12224)*. Springer, 415–438. https://doi.org/10.1007/978-3-030-53288-8_20

[36] Pranav Garg, Christof Löding, P. Madhusudan, and Daniel Neider. 2014. ICE: A Robust Framework for Learning Invariants. In *CAV (LNCS, Vol. 8559)*. Springer, 69–87. https://doi.org/10.1007/978-3-319-08867-9_5

[37] Pranav Garg, Daniel Neider, P. Madhusudan, and Dan Roth. 2016. Learning invariants using decision trees and implication counterexamples. In *POPL*. ACM, 499–512. https://doi.org/10.1145/2837614.2837664

[38] Roberto Giacobazzi and Francesco Ranzato. 1997. Completeness in Abstract Interpretation: A Domain Perspective. In *AMAST (LNCS, Vol. 1349)*. Springer, 231–245. https://doi.org/10.1007/BFb0000474

[39] Sumit Gulwani, Saurabh Srivastava, and Ramarathnam Venkatesan. 2009. Constraint-Based Invariant Inference over Predicate Abstraction. In *VMCAI (LNCS, Vol. 5403)*. Springer, 120–135. https://doi.org/10.1007/978-3-540-93900-9_13

[40] Ashutosh Gupta and Andrey Rybalchenko. 2009. InvGen: An Efficient Invariant Generator. In *CAV (LNCS, Vol. 5643)*. Springer, 634–640. https://doi.org/10.1007/978-3-642-02658-4_48

[41] Nicolas Halbwachs, Yann-Erick Proy, and Patrick Roumanoff. 1997. Verification of Real-Time Systems using Linear Relation Analysis. *Formal Methods Syst. Des.* 11, 2 (1997), 157–185. https://doi.org/10.1023/A:1008678014487

[42] Jingxuan He, Gagandeep Singh, Markus Püschel, and Martin T. Vechev. 2020. Learning fast and precise numerical analysis. In *PLDI*. ACM, 1112–1127. https://doi.org/10.1145/3385412.3386016

[43] Ehud Hrushovski, Joël Ouaknine, Amaury Pouly, and James Worrell. 2018. Polynomial Invariants for Affine Programs. In *LICS*. ACM, 530–539. https://doi.org/10.1145/3209108.3209142

[44] Andreas Humenberger, Maximilian Jaroschek, and Laura Kovács. 2017. Automated Generation of Non-Linear Loop Invariants Utilizing Hypergeometric Sequences. In *ISSAC*. ACM, 221–228. https://doi.org/10.1145/3087604.3087623

[45] Andreas Humenberger and Laura Kovács. 2021. Algebra-Based Synthesis of Loops and Their Invariants (Invited Paper). In *VMCAI (LNCS, Vol. 12597)*. Springer, 17–28. https://doi.org/10.1007/978-3-030-67067-2_2

[46] Rui-Juan Jing, Marc Moreno Maza, and Delaram Talaashrafi. 2020. Complexity Estimates for Fourier-Motzkin Elimination. In *CASC (LNCS, Vol. 12291)*. Springer, 282–306. https://doi.org/10.1007/978-3-030-60026-6_16

[47] Deepak Kapur. 2005. Automatically Generating Loop Invariants Using Quantifier Elimination. In *Deduction and Applications (Dagstuhl Seminar Proceedings, Vol. 05431)*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. http://drops.dagstuhl.de/opus/volltexte/2006/511

[48] Joost-Pieter Katoen, Annabelle McIver, Larissa Meinicke, and Carroll C. Morgan. 2010. Linear-Invariant Generation for Probabilistic Programs: - Automated Support for Proof-Based Methods. In *SAS (LNCS, Vol. 6337)*. Springer, 390–406. https://doi.org/10.1007/978-3-642-15769-1_24

[49] Zachary Kincaid, Jason Breck, Ashkan Forouhi Boroujeni, and Thomas W. Reps. 2017. Compositional recurrence analysis revisited. In *PLDI*. ACM, 248–262. https://doi.org/10.1145/3062341.3062373

[50] Zachary Kincaid, John Cyphert, Jason Breck, and Thomas W. Reps. 2018. Non-linear reasoning for invariant synthesis. *Proc. ACM Program. Lang.* 2, POPL (2018), 54:1–54:33. https://doi.org/10.1145/3158142

[51] D.A. Kohler. 1967. *Projections of convex polyhedral sets.* Technical Report. California University at Berkeley, Operations Research Center.

[52] Leslie Lamport. 1987. A Fast Mutual Exclusion Algorithm. *ACM Trans. Comput. Syst.* 5, 1 (1987), 1–11. https://doi.org/10.1145/7351.7352

[53] Wang Lin, Min Wu, Zhengfeng Yang, and Zhenbing Zeng. 2014. Proving total correctness and generating preconditions for loop programs via symbolic-numeric computation methods. *Frontiers Comput. Sci.* 8, 2 (2014), 192–202.

[54] Zohar Manna and Amir Pnueli. 1995. *Temporal verification of reactive systems - safety.* Springer.

[55] Kenneth L. McMillan. 2008. Quantified Invariant Generation Using an Interpolating Saturation Prover. In *TACAS (LNCS, Vol. 4963)*, C. R. Ramakrishnan and Jakob Rehof (Eds.). Springer, 413–427. https://doi.org/10.1007/978-3-540-78800-3_31

[56] Markus Müller-Olm and Helmut Seidl. 2004. Computing polynomial program invariants. *Inf. Process. Lett.* 91, 5 (2004), 233–244. https://doi.org/10.1016/j.ipl.2004.05.004

[57] ThanhVu Nguyen, Deepak Kapur, Westley Weimer, and Stephanie Forrest. 2012. Using dynamic analysis to discover polynomial and array invariants. In *ICSE*. IEEE Computer Society, 683–693. https://doi.org/10.1109/ICSE.2012.6227149

[58] Oded Padon, Kenneth L. McMillan, Aurojit Panda, Mooly Sagiv, and Sharon Shoham. 2016. Ivy: safety verification by interactive generalization. In *PLDI*. ACM, 614–630. https://doi.org/10.1145/2908080.2908118

[59] Andreas Podelski and Andrey Rybalchenko. 2004. A Complete Method for the Synthesis of Linear Ranking Functions. In *VMCAI (LNCS, Vol. 2937)*. Springer, 239–251. https://doi.org/10.1007/978-3-540-24622-0_20

[60] ppl 2021. Parma Polyhedra Library, PPL 1.2. https://www.bugseng.com/parma-polyhedra-library.

[61] Enric Rodríguez-Carbonell and Deepak Kapur. 2004. An Abstract Interpretation Approach for Automatic Generation of Polynomial Invariants. In *SAS (LNCS, Vol. 3148)*. Springer, 280–295. https://doi.org/10.1007/978-3-540-27864-1_21

[62] Enric Rodríguez-Carbonell and Deepak Kapur. 2004. Automatic Generation of Polynomial Loop Invariants: Algebraic Foundations. In *ISSAC*. ACM, 266–273. https://doi.org/10.1145/1005285.1005324

[63] Enric Rodríguez-Carbonell and Deepak Kapur. 2007. Automatic generation of polynomial invariants of bounded degree using abstract interpretation. *Sci. Comput. Program.* 64, 1 (2007), 54–75. https://doi.org/10.1016/j.scico.2006.03.003

[64] Sriram Sankaranarayanan, Henny Sipma, and Zohar Manna. 2004. Non-linear loop invariant generation using Gröbner bases. In *POPL*. ACM, 318–329. https://doi.org/10.1145/964001.964028

[65] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. 2004. Constraint-Based Linear-Relations Analysis. In *SAS (LNCS, Vol. 3148)*. Springer, 53–68. https://doi.org/10.1007/978-3-540-27864-1_7

[66] Alexander Schrijver. 1999. *Theory of linear and integer programming.* Wiley.

[67] Rahul Sharma and Alex Aiken. 2016. From invariant checking to invariant inference using randomized search. *Formal Methods Syst. Des.* 48, 3 (2016), 235–256. https://doi.org/10.1007/s10703-016-0248-5

[68] Rahul Sharma, Saurabh Gupta, Bharath Hariharan, Alex Aiken, Percy Liang, and Aditya V. Nori. 2013. A Data Driven Approach for Algebraic Loop Invariants. In *ESOP (LNCS, Vol. 7792)*. Springer, 574–592. https://doi.org/10.1007/978-3-642-37036-6_31

[69] Sparse [n.d.]. Sparse: C language semantic parser. https://lwn.net/Articles/689907/.

[70] StInG [n.d.]. StInG: Stanford Invariant Generator. http://theory.stanford.edu/~srirams/Software/sting.html.

[71] Jun Sun, Yang Liu, Jin Song Dong, and Xian Zhang. 2009. Verifying Stateful Timed CSP Using Implicit Clocks and Zone Abstraction. In *Formal Methods and Software Engineering, 11th International Conference on Formal Engineering Methods, ICFEM 2009, Rio de Janeiro, Brazil, December 9-12, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5885)*, Karin K. Breitman and Ana Cavalcanti (Eds.). Springer, 581–600. https://doi.org/10.1007/978-3-642-10373-5_30

[72] Jinyi Wang, Yican Sun, Hongfei Fu, Krishnendu Chatterjee, and Amir Kafshdar Goharshady. 2021. Quantitative analysis of assertion violations in probabilistic programs. In *PLDI*. ACM, 1171–1186. https://doi.org/10.1145/3453483.3454102

[73] Peixin Wang, Hongfei Fu, Amir Kafshdar Goharshady, Krishnendu Chatterjee, Xudong Qin, and Wenjun Shi. 2019. Cost analysis of nondeterministic probabilistic programs. In *PLDI*. ACM, 204–220. https://doi.org/10.1145/3314221.3314581

[74] Rongchen Xu, Fei He, and Bow-Yaw Wang. 2020. Interval counterexamples for loop invariant learning. In *ESEC/FSE*. ACM, 111–122. https://doi.org/10.1145/3368089.3409752

[75] Lu Yang, Chaochen Zhou, Naijun Zhan, and Bican Xia. 2010. Recent advances in program verification through computer algebra. *Frontiers Comput. Sci. China* 4, 1 (2010), 1–16. https://doi.org/10.1007/s11704-009-0074-7

[76] Jianan Yao, Gabriel Ryan, Justin Wong, Suman Jana, and Ronghui Gu. 2020. Learning nonlinear loop invariants with gated continuous logic networks. In *PLDI*. ACM, 106–120. https://doi.org/10.1145/3385412.3385986

Table 6. Experimental Results for Polyhedron Projection

Benchmarks					Our Approach							
Name		Loc	Dim	Line	Time (Expan-Order A)				Time (Expan-Order B)			
					No-Proj	FME	Kohler	Kohler*	No-Proj	FME	Kohler	Kohler*
Origin Scheduler	2p	2	16	-	0.01	0.01	0.03	0.02	0.01	0.01	0.02	0.02
	3p	3	33	-	0.06	0.12	0.25	0.05	0.06	0.08	0.27	0.08
	4p	4	56	-	0.22	0.55	1.38	0.47	0.24	0.52	1.42	0.46
	5p	5	85	-	0.48	1.31	3.88	1.20	0.38	1.32	4.03	1.07
	6p	6	120	-	-	-	-	-	38.59	48.88	28.01	11.20
	7p	7	161	-	-	-	-	-	62,852.98	67,350.57	182.71	114.64
Fixed Scheduler	3p	3	33	336	0.05	0.09	0.28	0.08	0.05	0.09	0.16	0.08
	4p	4	56	609	0.16	0.32	1.14	0.32	0.16	0.41	1.20	0.34
	5p	5	85	1017	1.22	1.83	4.86	1.49	1.40	2.12	5.71	1.63
	6p	6	120	1587	34.07	44.46	21.67	6.35	36.54	46.59	23.98	6.60
	7p	7	161	2352	-	-	-	-	67,671.98	72,406.55	156.64	80.51

Table 7. Experimental Results for Parallel Computation (Expan-Order A: No-Proj)

Benchmarks					Time (Expan-Order A: No-Proj)														
Name		Loc	Dim	Line	1	2	3	4	5	6	7	8	9	10	11	12	Max	Speedup(original)	Speedup
Origin Scheduler	2p	2	16	-	0.01	0.01	-	-	-	-	-	-	-	-	-	-	0.01	0.50X	1.00X
	3p	3	33	-	0.04	0.04	0.04	-	-	-	-	-	-	-	-	-	0.04	1.41X	4.25X
	4p	4	56	-	0.51	1.27	0.93	0.21	-	-	-	-	-	-	-	-	1.27	20.82X	47.88X
	5p	5	85	-	2.36	29.98	81.85	102.53	11.96	-	-	-	-	-	-	-	102.53	32.51X	72.52X
	6p	6	120	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	7p	7	161	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Fixed Scheduler	3p	3	33	336	0.04	0.04	0.03	-	-	-	-	-	-	-	-	-	0.04	1.54X	4.25X
	4p	4	56	609	0.26	0.87	1.07	0.26	-	-	-	-	-	-	-	-	1.07	1.69X	3.88X
	5p	5	85	1017	3.24	45.94	91.71	82.26	4.01	-	-	-	-	-	-	-	91.71	0.59X	1.48X
	6p	6	120	1587	146.05	3,172.04	8,389.32	8,068.50	8,232.79	881.74	-	-	-	-	-	-	8,389.32	0.26X	0.89X
	7p	7	161	2352	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Fischer	6p	7	63	710	1.37	0.20	0.20	0.26	0.50	1.49	0.33	-	-	-	-	-	1.49	2.11X	6.16X
	7p	8	80	987	13.31	0.44	0.48	0.59	1.03	2.72	9.08	0.85	-	-	-	-	13.31	2.07X	4.44X
	8p	9	99	1327	56.16	0.96	1.02	1.25	2.12	5.05	15.65	55.38	2.24	-	-	-	56.16	2.67X	6.65X
	9p	10	120	1736	713.68	1.99	2.12	2.59	4.26	9.59	28.02	93.18	333.30	5.74	-	-	713.68	1.96X	3.28X
	10p	11	143	2218	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	11p	12	168	2780	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Fixed Cars	2p	3	27	216	0.01	<0.01	0.01	-	-	-	-	-	-	-	-	-	0.01	0.50X	1.00X
	3p	5	60	616	0.26	0.02	0.03	0.04	0.04	-	-	-	-	-	-	-	0.26	1,437.69X	2,156.53X
	4p	7	105	1283	42.89	0.33	0.47	0.34	0.49	0.36	0.51	-	-	-	-	-	42.89	>793.12X	>839.35X

Table 8. Experimental Results for Parallel Computation (Expan-Order A: FME)

Benchmarks					Time (Expan-Order A: FME)														
Name		Loc	Dim	Line	1	2	3	4	5	6	7	8	9	10	11	12	Max	Speedup(original)	Speedup
Origin Scheduler	2p	2	16	-	0.01	0.01	-	-	-	-	-	-	-	-	-	-	0.01	0.50X	1.00X
	3p	3	33	-	0.07	0.05	0.05	-	-	-	-	-	-	-	-	-	0.07	1.00X	2.42X
	4p	4	56	-	0.65	1.15	1.01	0.27	-	-	-	-	-	-	-	-	1.15	19.74X	52.87X
	5p	5	85	-	2.38	29.11	84.96	107.82	12.94	-	-	-	-	-	-	-	107.82	31.34X	68.96X
	6p	6	120	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	7p	7	161	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Fixed Scheduler	3p	3	33	336	0.05	0.05	0.04	-	-	-	-	-	-	-	-	-	0.05	1.21X	3.40X
	4p	4	56	609	0.33	0.93	1.11	0.31	-	-	-	-	-	-	-	-	1.11	1.55X	3.74X
	5p	5	85	1017	3.34	47.52	94.79	82.96	4.11	-	-	-	-	-	-	-	94.79	0.58X	1.43X
	6p	6	120	1587	145.78	3,172.68	8,402.87	8,058.62	7,947.10	884.46	-	-	-	-	-	-	8,402.87	0.26X	0.89X
	7p	7	161	2352	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Fischer	6p	7	63	710	1.28	0.22	0.24	0.29	0.54	1.53	0.35	-	-	-	-	-	1.53	2.06X	6.00X
	7p	8	80	987	13.47	0.51	0.55	0.66	1.13	2.81	9.33	0.93	-	-	-	-	13.47	2.01X	4.39X
	8p	9	99	1327	57.69	1.11	1.24	1.39	2.27	5.57	16.58	56.76	2.37	-	-	-	57.69	2.57X	6.47X
	9p	10	120	1736	721.78	2.26	2.38	2.91	4.56	10.02	28.55	95.98	348.36	6.64	-	-	721.78	1.91X	3.25X
	10p	11	143	2218	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	11p	12	168	2780	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Fixed Cars	2p	3	27	216	0.01	<0.01	0.01	-	-	-	-	-	-	-	-	-	0.01	0.50X	1.00X
	3p	5	60	616	0.27	0.03	0.03	0.04	0.04	-	-	-	-	-	-	-	0.27	1,367.56X	2076.66X
	4p	7	105	1283	44.08	0.39	0.54	0.42	0.55	0.44	0.57	-	-	-	-	-	44.08	>766.12X	>816.69X

A DETAILED EXPERIMENTAL RESULTS

Table 9. Experimental Results for Parallel Computation (Expan-Order A: Kohler)

Benchmarks					Time (Expan-Order A: Kohler)														
Name		Loc	Dim	Line	1	2	3	4	5	6	7	8	9	10	11	12	Max	Speedup(original)	Speedup
Origin Scheduler	2p	2	16	-	0.01	0.02	-	-	-	-	-	-	-	-	-	-	0.02	0.33X	0.50X
	3p	3	33	-	0.11	0.11	0.11	-	-	-	-	-	-	-	-	-	0.11	0.51X	1.54X
	4p	4	56	-	0.79	1.38	1.24	0.49	-	-	-	-	-	-	-	-	1.38	15.59X	44.06X
	5p	5	85	-	2.89	29.64	82.10	103.74	12.67	-	-	-	-	-	-	-	103.74	32.18X	71.68X
	6p	6	120	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	7p	7	161	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Fixed Scheduler	3p	3	33	336	0.09	0.11	0.12	-	-	-	-	-	-	-	-	-	0.12	0.53X	1.41X
	4p	4	56	609	0.50	1.12	1.31	0.53	-	-	-	-	-	-	-	-	1.31	1.20X	3.17X
	5p	5	85	1017	3.93	46.51	92.50	82.02	4.71	-	-	-	-	-	-	-	92.50	0.59X	1.46X
	6p	6	120	1587	146.57	3,228.52	8,468.86	8,105.17	8,171.11	868.50	-	-	-	-	-	-	8,468.86	0.26X	0.89X
	7p	7	161	2352	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Fischer	6p	7	63	710	1.31	0.26	0.28	0.33	0.58	1.56	0.39	-	-	-	-	-	1.56	1.94X	5.88X
	7p	8	80	987	13.36	0.58	0.61	0.74	1.19	2.88	9.42	1.02	-	-	-	-	13.36	1.98X	4.42X
	8p	9	99	1327	57.09	1.22	1.27	1.51	2.34	5.37	16.20	56.62	2.58	-	-	-	57.09	2.59X	6.54X
	9p	10	120	1736	728.55	2.38	2.53	3.02	4.63	10.08	28.60	94.00	339.17	6.19	-	-	728.55	1.92X	3.22X
	10p	11	143	2218	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	11p	12	168	2780	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Fixed Cars	2p	3	27	216	<0.01	0.01	0.01	-	-	-	-	-	-	-	-	-	0.01	0.50X	1.00X
	3p	5	60	616	0.27	0.04	0.04	0.05	0.06	-	-	-	-	-	-	-	0.27	1,218.91X	2,076.66X
	4p	7	105	1283	44.04	0.49	0.64	0.51	0.66	0.55	0.68	-	-	-	-	-	44.04	>756.77X	>817.43X

Table 10. Experimental Results for Parallel Computation (Expan-Order A: Kohler*)

Benchmarks					Time (Expan-Order A: Kohler*)														
Name		Loc	Dim	Line	1	2	3	4	5	6	7	8	9	10	11	12	Max	Speedup(original)	Speedup
Origin Scheduler	2p	2	16	-	0.01	0.01	-	-	-	-	-	-	-	-	-	-	0.01	0.50X	1.00X
	3p	3	33	-	0.06	0.04	0.04	-	-	-	-	-	-	-	-	-	0.06	1.21X	2.83X
	4p	4	56	-	0.58	1.15	0.98	0.26	-	-	-	-	-	-	-	-	1.15	20.47X	52.87X
	5p	5	85	-	2.36	28.93	81.02	103.14	12.04	-	-	-	-	-	-	-	103.14	32.68X	72.09X
	6p	6	120	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	7p	7	161	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Fixed Scheduler	3p	3	33	336	0.05	0.06	0.04	-	-	-	-	-	-	-	-	-	0.06	1.13X	2.83X
	4p	4	56	609	0.30	0.91	1.11	0.29	-	-	-	-	-	-	-	-	1.11	1.59X	3.74X
	5p	5	85	1017	3.38	45.85	91.55	81.78	3.94	-	-	-	-	-	-	-	91.55	0.59X	1.48X
	6p	6	120	1587	144.89	3,177.27	8,386.05	8,137.26	8,376.15	864.79	-	-	-	-	-	-	8,386.05	0.25X	0.89X
	7p	7	161	2352	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Fischer	6p	7	63	710	1.51	0.22	0.23	0.29	0.53	1.52	0.37	-	-	-	-	-	1.52	1.96X	6.03X
	7p	8	80	987	13.35	0.50	0.54	0.65	1.12	2.80	9.29	0.93	-	-	-	-	13.35	2.02X	4.43X
	8p	9	99	1327	56.87	1.07	1.13	1.36	2.20	5.17	15.94	55.96	2.35	-	-	-	56.87	2.63X	6.56X
	9p	10	120	1736	735.70	2.17	2.31	2.78	4.43	9.87	28.55	105.75	350.42	6.26	-	-	735.70	1.87X	3.18X
	10p	11	143	2218	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	11p	12	168	2780	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Fixed Cars	2p	3	27	216	<0.01	0.01	<0.01	-	-	-	-	-	-	-	-	-	0.01	1.00X	1.00X
	3p	5	60	616	0.27	0.03	0.03	0.04	0.05	-	-	-	-	-	-	-	0.27	1335.00X	2,076.66X
	4p	7	105	1283	44.61	0.39	0.53	0.41	0.55	0.43	0.57	-	-	-	-	-	44.61	>758.05X	>806.99X

Table 11. Experimental Results for Parallel Computation (Expan-Order B: No-Proj)

Benchmarks					Time (Expan-Order B: No-Proj)														
Name		Loc	Dim	Line	1	2	3	4	5	6	7	8	9	10	11	12	Max	Speedup(original)	Speedup
Origin Scheduler	2p	2	16	-	0.01	0	-	-	-	-	-	-	-	-	-	-	0.01	1.00X	1.00X
	3p	3	33	-	0.04	0.03	0.03	-	-	-	-	-	-	-	-	-	0.04	1.70X	4.25X
	4p	4	56	-	0.40	0.45	0.32	0.21	-	-	-	-	-	-	-	-	0.45	44.06X	135.13X
	5p	5	85	-	6.93	5.81	7.36	4.54	1.88	-	-	-	-	-	-	-	7.36	280.40X	1010.37X
	6p	6	120	-	42.30	178.32	310.36	216.76	114.69	30.32	-	-	-	-	-	-	310.36	>40.32X	>115.99X
	7p	7	161	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Fixed Scheduler	3p	3	33	336	0.03	0.04	0.03	-	-	-	-	-	-	-	-	-	0.04	1.70X	4.25X
	4p	4	56	609	0.23	0.27	0.26	0.22	-	-	-	-	-	-	-	-	0.27	4.24X	15.40X
	5p	5	85	1017	2.40	3.29	4.16	2.83	1.29	-	-	-	-	-	-	-	4.16	9.72X	32.64X
	6p	6	120	1587	27.75	47.90	80.32	55.99	44.81	21.19	-	-	-	-	-	-	80.32	27.13X	93.89X
	7p	7	161	2352	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Fischer	6p	7	63	710	0.40	0.19	0.20	0.26	0.47	1.21	0.34	-	-	-	-	-	1.21	2.99X	7.58X
	7p	8	80	987	0.90	0.44	0.45	0.53	0.85	2.17	7.40	0.86	-	-	-	-	7.40	4.35X	7.99X
	8p	9	99	1327	1.78	0.94	0.98	1.12	1.67	3.89	12.34	45.49	2.21	-	-	-	45.49	5.30X	8.21X
	9p	10	120	1736	3.31	1.89	1.95	2.22	3.33	7.15	21.29	74.17	277.24	5.71	-	-	277.24	5.89X	8.46X
	10p	11	143	2218	5.97	3.66	3.82	4.38	6.45	13.68	38.88	128.98	456.91	1,684.68	14.46	-	1,684.68	6.20X	8.70X
	11p	12	168	2780	10.42	6.96	7.25	8.55	12.97	27.77	76.58	240.98	813.28	2,841.46	10,224.63	36.55	10,224.63	>2.51X	>3.52X
Fixed Cars	2p	3	27	216	0.01	<0.01	<0.01	-	-	-	-	-	-	-	-	-	0.01	1.00X	1.00X
	3p	5	60	616	0.84	0.08	0.09	0.10	0.10	-	-	-	-	-	-	-	0.84	463.38X	667.50X
	4p	7	105	1283	76.99	1.55	1.48	1.69	1.67	1.72	1.73	-	-	-	-	-	76.99	>414.60X	>467.59X

Table 12. Experimental Results for Parallel Computation (Expan-Order B: FME)

Benchmarks					Time (Expan-Order B: FME)														
Name		Loc	Dim	Line	1	2	3	4	5	6	7	8	9	10	11	12	Max	Speedup(original)	Speedup
Origin Scheduler	2p	2	16	-	0.01	<0.01	-	-	-	-	-	-	-	-	-	-	0.01	1.00X	1.00X
	3p	3	33	-	0.05	0.04	0.05	-	-	-	-	-	-	-	-	-	0.05	1.21X	3.40X
	4p	4	56	-	0.48	0.50	0.36	0.26	-	-	-	-	-	-	-	-	0.50	38.00X	121.62X
	5p	5	85	-	7.04	5.93	7.46	4.66	2.02	-	-	-	-	-	-	-	7.46	274.30X	996.82X
	6p	6	120	-	42.26	181.22	313.45	220.01	116.21	33.12	-	-	-	-	-	-	313.45	>39.72X	>114.85X
	7p	7	161	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Fixed Scheduler	3p	3	33	336	0.04	0.05	0.04	-	-	-	-	-	-	-	-	-	0.05	1.30X	3.40X
	4p	4	56	609	0.29	0.32	0.30	0.27	-	-	-	-	-	-	-	-	0.32	3.52X	13.00X
	5p	5	85	1017	2.51	3.44	4.30	2.91	1.42	-	-	-	-	-	-	-	4.30	9.31X	31.58X
	6p	6	120	1587	28.43	51.57	80.06	58.66	43.09	23.49	-	-	-	-	-	-	80.06	26.43X	94.19X
	7p	7	161	2352	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Fischer	6p	7	63	710	0.41	0.23	0.23	0.28	0.46	1.25	0.35	-	-	-	-	-	1.25	2.85X	7.34X
	7p	8	80	987	0.91	0.51	0.52	0.60	0.93	2.23	7.54	0.94	-	-	-	-	7.54	4.17X	7.84X
	8p	9	99	1327	1.83	1.06	1.09	1.23	1.80	4.03	12.48	46.06	2.39	-	-	-	46.06	5.19X	8.11X
	9p	10	120	1736	3.55	2.11	2.18	2.46	3.51	7.36	21.81	75.38	284.34	6.09	-	-	284.34	5.73X	8.25X
	10p	11	143	2218	6.43	4.03	4.22	4.81	7.05	14.18	40.05	131.75	465.63	1,736.17	14.94	-	1,736.17	6.03X	8.44X
	11p	12	168	2780	10.96	7.47	7.83	9.11	13.64	28.67	78.04	244.84	822.43	2,869.50	10,339.85	37.42	10,339.85	>2.48X	>3.48X
Fixed Cars	2p	3	27	216	0.01	0.01	<0.01	-	-	-	-	-	-	-	-	-	0.01	0.50X	1.00X
	3p	5	60	616	0.85	0.09	0.09	0.11	0.11	-	-	-	-	-	-	-	0.85	448.56X	659.64X
	4p	7	105	1283	77.60	1.36	1.32	1.52	1.52	1.56	1.55	-	-	-	-	-	77.60	>416.52X	>463.91X

Table 13. Experimental Results for Parallel Computation (Expan-Order B: Kohler)

Benchmarks					Time (Expan-Order B: Kohler)														
Name		Loc	Dim	Line	1	2	3	4	5	6	7	8	9	10	11	12	Max	Speedup(original)	Speedup
Origin Scheduler	2p	2	16	-	0.01	0.02	-	-	-	-	-	-	-	-	-	-	0.02	0.33X	0.50X
	3p	3	33	-	0.11	0.10	0.10	-	-	-	-	-	-	-	-	-	0.11	0.54X	1.54X
	4p	4	56	-	0.65	0.69	0.59	0.48	-	-	-	-	-	-	-	-	0.69	25.23X	88.13X
	5p	5	85	-	7.68	6.62	8.21	5.34	2.64	-	-	-	-	-	-	-	8.21	243.89X	905.76X
	6p	6	120	-	44.24	177.94	313.50	220.60	112.74	18.25	-	-	-	-	-	-	313.50	>40.57X	>114.83X
	7p	7	161	-	871.68	3,517.01	5,463.73	4,203.43	2,678.93	5,671.06	430.29	-	-	-	-	-	5,671.06	>1.57X	>6.34X
Fixed Scheduler	3p	3	33	336	0.08	0.09	0.09	-	-	-	-	-	-	-	-	-	0.09	0.65X	1.88X
	4p	4	56	609	0.46	0.52	0.49	0.49	-	-	-	-	-	-	-	-	0.52	2.12X	8.00X
	5p	5	85	1017	3.11	4.07	4.87	3.50	2.00	-	-	-	-	-	-	-	4.87	7.73X	27.88X
	6p	6	120	1587	30.10	47.30	75.93	59.12	41.39	8.47	-	-	-	-	-	-	75.93	28.75X	99.32X
	7p	7	161	2352	463.98	806.16	1,371.55	915.13	774.16	531.02	45.03	-	-	-	-	-	1,371.55	>7.33X	>26.24X
Fischer	6p	7	63	710	0.43	0.25	0.27	0.30	0.50	1.29	0.40	-	-	-	-	-	1.29	2.66X	7.11X
	7p	8	80	987	0.96	0.58	0.59	0.67	1.00	2.30	7.61	1.02	-	-	-	-	7.61	4.01X	7.77X
	8p	9	99	1327	1.96	1.18	1.21	1.37	1.93	4.15	12.87	46.34	2.51	-	-	-	46.34	5.08X	8.06X
	9p	10	120	1736	3.74	2.38	2.38	2.67	3.76	7.68	22.17	76.12	286.63	6.23	-	-	286.63	5.66X	8.18X
	10p	11	143	2218	6.68	4.39	4.51	5.09	7.23	14.63	40.31	132.17	465.49	1,705.49	15.20	-	1,705.49	6.10X	8.59X
	11p	12	168	2780	11.44	8.01	8.34	9.66	14.19	29.08	78.58	246.01	823.62	2,871.39	10,345.63	37.95	10,345.63	>2.48X	>3.47X
Fixed Cars	2p	3	27	216	0.01	0.01	<0.01	-	-	-	-	-	-	-	-	-	0.01	0.50X	1.00X
	3p	5	60	616	1.01	0.12	0.12	0.15	0.15	-	-	-	-	-	-	-	1.01	361.74X	555.14X
	4p	7	105	1283	77.75	1.46	1.42	1.62	1.62	1.67	1.66	-	-	-	-	-	77.75	>412.84X	>463.02X

Table 14. Experimental Results for Parallel Computation (Expan-Order B: Kohler*)

Benchmarks					Time (Expan-Order B: Kohler*)														
Name		Loc	Dim	Line	1	2	3	4	5	6	7	8	9	10	11	12	Max	Speedup(original)	Speedup
Origin Scheduler	2p	2	16	-	0.01	0.01	-	-	-	-	-	-	-	-	-	-	0.01	0.50X	1.00X
	3p	3	33	-	0.05	0.04	0.05	-	-	-	-	-	-	-	-	-	0.05	1.21X	3.40X
	4p	4	56	-	0.41	0.47	0.36	0.26	-	-	-	-	-	-	-	-	0.47	40.54X	129.38X
	5p	5	85	-	7.04	6.01	7.52	4.70	2.00	-	-	-	-	-	-	-	7.52	272.69X	988.87X
	6p	6	120	-	41.77	175.80	308.48	216.41	110.84	15.22	-	-	-	-	-	-	308.48	>41.44X	>116.70X
	7p	7	161	-	981.03	3,669.94	5,413.91	4,025.82	2,567.09	5,424.19	402.58	-	-	-	-	-	5,424.19	>1.60X	>6.63X
Fixed Scheduler	3p	3	33	336	0.04	0.04	0.04	-	-	-	-	-	-	-	-	-	0.04	1.41X	4.25X
	4p	4	56	609	0.32	0.39	0.31	0.26	-	-	-	-	-	-	-	-	0.39	3.25X	10.66X
	5p	5	85	1017	2.63	3.54	4.90	3.37	1.38	-	-	-	-	-	-	-	4.90	8.58X	27.71X
	6p	6	120	1587	28.74	44.04	72.07	51.77	38.62	5.26	-	-	-	-	-	-	72.07	31.35X	104.64X
	7p	7	161	2352	443.54	787.35	1,350.66	892.31	745.12	505.90	27.13	-	-	-	-	-	1,350.66	>7.57X	>26.65X
Fischer	6p	7	63	710	0.40	0.22	0.23	0.26	0.44	1.22	0.38	-	-	-	-	-	1.22	2.91X	7.52X
	7p	8	80	987	0.92	0.53	0.54	0.62	1.00	2.53	7.75	1.02	-	-	-	-	7.75	3.96X	7.63X
	8p	9	99	1327	1.86	1.04	1.07	1.21	1.79	3.97	12.45	45.41	2.35	-	-	-	45.41	5.25X	8.22X
	9p	10	120	1736	3.48	2.10	2.12	2.40	3.48	7.71	25.06	77.82	283.95	5.96	-	-	283.95	5.66X	8.26X
	10p	11	143	2218	6.27	3.95	4.10	4.69	6.77	13.99	39.27	134.19	469.25	1,700.51	15.39	-	1,700.51	6.11X	8.62X
	11p	12	168	2780	10.91	7.46	7.77	9.10	13.51	28.25	77.18	241.64	813.60	2,841.93	10,230.51	36.98	10,230.51	>2.51X	>3.51X
Fixed Cars	2p	3	27	216	0.01	<0.01	<0.01	-	-	-	-	-	-	-	-	-	0.01	1.00X	1.00X
	3p	5	60	616	0.83	0.08	0.09	0.10	0.11	-	-	-	-	-	-	-	0.83	463.38X	675.54X
	4p	7	105	1283	75.83	1.33	1.32	1.48	1.47	1.54	1.52	-	-	-	-	-	75.83	>426.08X	>474.74X