Inductive Termination Proofs with Transition Invariants and Their Relationship to the Size-Change Abstraction

Florian Zuleger(✉)

TU Wien, Vienna, Austria
zuleger@forsyte.at

Abstract. Transition invariants are a popular technique for automated termination analysis. A transition invariant is a covering of the transitive closure of the transition relation of a program by a finite number of well-founded relations. The covering is usually established by an inductive proof using transition predicate abstraction. Such inductive termination proofs have the structure of a finite automaton. These automata, which we call transition automata, offer a rich structure that has not been exploited in previous publications. We establish a new connection between transition automata and the size-change abstraction, which is another widespread technique for automated termination analysis. In particular, we are able to transfer recent results on automated complexity analysis with the size-change abstraction to transition invariants.

1 Introduction

The last decade has seen considerable interest in automated techniques for proving the termination of programs. Notably, the TERMINATOR termination analyzer [14] has been able to analyze device drivers with several thousand lines of code. The analysis in [14] uses the termination criterion suggested by Rybalchenko and Podelski in [25] (for a discussion of earlier work that implicitly used the same principle we refer the reader to [6]): In order to show the well-foundedness of a relation R, it is sufficient to find a finite number of well-founded relations $R_1, \ldots, R_k$ with

$$R^+ \subseteq R_1 \cup \cdots \cup R_k \qquad (*)$$

where R^+ denotes the transitive closure of R.

An essential difficulty in using the above criterion lies in establishing the condition (*), as reasoning about the transitive closure R^+ usually requires induction. For this reason, not only the above criterion but also an inductive argument for establishing (*) was suggested in [25]. The inductive argument was further developed in [26], where the use of *transition predicate abstraction* (TPA) has been suggested for establishing condition (*). TPA is the basis for

A. Podelski (Ed.): SAS 2018, LNCS 11002, pp. 423–444, 2018.
https://doi.org/10.1007/978-3-319-99725-4_25

the termination analysis in TERMINATOR. The starting point of our research are the inductive termination proofs with TPA, which have the structure of finite automata (as already observed in [26]). These automata, which we call *transition automata*, offer a rich structure that has not been exploited in previous publications. It is precisely this automaton structure, which allows us to connect inductive termination proofs with TPA to the size-change abstraction, and transfer recent results on automated complexity analysis.

We contrast our approach with the fascinating line of work [6,30,32], which aims at bounding the height of the relation R in terms of the height of the relations $R_1, \ldots, R_k$. In order to derive such bounds, [6,30] replace Ramsey's theorem, which has been used to prove (*) in [25], by more fine-grained Ramsey-like arguments. In this paper, we show that *inductive* termination proofs with TPA do not need to rely on Ramsey's theorem and can be analyzed solely by *automata-theoretic techniques*.

Size-change abstraction (SCA), introduced by Ben-Amram, Lee and Jones in [22], is another wide-spread technique for automated termination analysis. SCA has been employed for the analysis of functional [22,23], logical [31] and imperative [3,10] programs and term rewriting systems [9], and is implemented in the industrial-strength systems ACL2 [23] and Isabelle [20]. Recently, SCA has also been used for resource bound and complexity analysis of imperative programs [34]. SCA is attractive because of several strong theoretical results on termination analysis [22], complexity analysis [12,33] and the existence of ranking functions [5,33]. The success of SCA has also inspired generalizations to richer classes of constraints [4,5,7]. The connection between TPA and SCA has been the subject of previous research [19], which contains first results but does not exploit the automaton structure of inductive termination proofs. In this paper, we make the following contributions:

Result 1: Our main result (Theorem 7) makes it possible to transfer recent results on automated complexity analysis with the size-change abstraction [12] to transition automata. In particular, we obtain a complete and effective characterization of asymptotic complexity analysis with transition automata. This result holds the potential for the design of new automated complexity analyzers, for example, by extracting complexity bounds from the inductive termination proofs computed by TERMINATOR. We illustrate our result in the following. We consider the programs P_1 and P_2 given by Examples 1 and 2 in Fig. 1. One can model the transition relation of P_1 by the predicate $x' = x - 1 \wedge y' = N \vee x' = x \wedge y' = y - 1$ and the transition relation of P_2 by the predicate $x' = x - 1 \wedge y' = y \vee x' = x \wedge y' = y - 1$. The two relations R_1 and R_2 given by the predicates $x' < x$ resp. $y' < y$ are a transition invariant for both programs; we give an inductive proof which establishes condition (*) for both programs in Sect. 3. For motivation of our results we state here the relation to [6]: With the program invariant $x \leq N \wedge y \leq N$ (which can be computed by standard techniques such as Octagon analysis [24]), the result of [6] allows us to obtain the quadratic bound $O(N^2)$ on the complexity of both programs from the transition invariant given by the relations R_1 and R_2. However, this bound is imprecise for

Example 1.

```
main(nat N) {
  nat x = N; nat y = N;
  while (x>0 ∧ y>0) {
     if(?){ //transition a1
        x--; y = N;
     }
     else { //transition a2
        y--;
} } }
```

Example 2.

```
main(nat N) {
  nat x = N; nat y = N;
  while (x>0 ∧ y>0) {
     if(?){ //transition a1
        x--;
     }
     else { //transition a2
        y--;
} } }
```

Fig. 1. The ? in the condition represents non-deterministic choice.

P_2, which has linear complexity. There is no hope in improving the bound for P_2, because the result of [6] just relies on R_1 and R_2. In this paper, we demonstrate that the inductive termination proof offers more structure. We show that just by analyzing the automaton structure of the proof we can deduce the linear bound $O(N)$ for P_2.

Result 2: Following [26] we examine a first termination criterion based on the universality of transition automata and show that the universality of the transition automaton implies the termination of the program under analysis (Theorem 2). We then show that transition automata admit a *more general* termination criterion based on the definition of an associated Büchi-automaton (Theorems 1 and 3). This more general termination criterion has the advantage that fewer predicates are needed for the termination proof (Example 7). We finally show that this new criterion is in fact the *most general* termination criterion admitted by transition automata (Theorem 4).

Result 3: We connect transition automata to the size-change abstraction in Sect. 6. In particular, we show how to transfer several results from the size-change abstraction to transition automata, demonstrating that techniques from SCA are applicable for the analysis of inductive termination proofs with transition predicate abstraction. This is of fundamental interest for understanding the relationship of both termination principles, because transition invariants have been suggested in [25] as a generalization of size-change termination proofs (and indeed later work has formally established that every size-change termination proof can be mimicked by a transition invariant termination proof [19]).

Organization of the Paper. Section 2 gives the basic definitions. Section 3 reviews transition predicate abstraction as introduced in [26]. Section 4 introduces transition automata and gives termination criteria. Section 5 reviews the size-change abstraction. Section 6 defines 'canonical' programs for transition automata and transfers results from the size-change abstraction to transition automata. Section 7 concludes.

2 Basic Definitions

We use $\circ$ to denote the usual *product* of relations, i.e., given two relations $B_1, B_2 \subseteq A \times A$ we define $B_1 \circ B_2 = \{(a_1, a_3) \mid \text{ there is an } a_2 \in A \text{ with } (a_1, a_2) \in B_1 \text{ and } (a_2, a_3) \in B_2\}$. Let $B \subseteq A \times A$ be a relation. B is *well-founded* if there is no infinite sequence of states $a_1 a_2 \cdots$ with $(a_i, a_{i+1}) \in B$ for all i. The *transitive closure* of B is defined by $B^+ = \bigcup_{i \geq 1} B^i$, where $B^0 = \{(a, a) \mid a \in A\}$, $B^{i+1} = B^i \circ B$. Let $B \subseteq A \times A$ be a well-founded relation. For every element $a \in A$ we inductively define its *ordinal height* $\|a\|_B$ by setting $\|a\|_B = \sup_{(a,b) \in B} \|b\|_B + 1$, where sup over the empty set evaluates to 0. We note that $\|\cdot\|_B$ is well-defined because B is well-founded. We define the *ordinal height* of relation B as $\|B\| = \sup_{a \in A} \|a\|_B + 1$.

2.1 Automata

A *finite automaton* $A = \langle Q, \Sigma, \delta, \iota, F\rangle$ consists of a finite set of *states*, a finite alphabet Σ, a *transition relation* $\delta : \Sigma \to \mathbf{2}^{Q \times Q}$, an *initial state* $\iota \in Q$, and a set of final states $F \subseteq Q$. Automaton A is *deterministic* if for every $\tau \in Q$ and $a \in \Sigma$ there is at most one $\tau' \in Q$ such that $(\tau, \tau') \in \delta(a)$. We also write $\tau \xrightarrow{a} \tau'$ for $(\tau, \tau') \in \delta(a)$. We extend the transition relation to words and define $\delta(w) = \delta(a_1) \circ \cdots \circ \delta(a_l)$ for every $w = a_1 \cdots a_l \in \Sigma^*$. A *run* of A is a finite sequence $r = \iota \xrightarrow{a_1} \tau_1 \xrightarrow{a_2} \tau_2 \cdots \xrightarrow{a_l} \tau_l$. r is *accepting* if $\tau_l \in F$. Automaton A *accepts* a finite word $w \in \Sigma^*$ if there is an accepting run $r = \iota \xrightarrow{a_1} \tau_1 \xrightarrow{a_2} \tau_2 \cdots \xrightarrow{a_l} \tau_l$ such that $w = a_1 \cdots a_l$. We denote by $\mathcal{L}(A) = \{w \in \Sigma^* \mid A \text{ accepts } w\}$ the *language* of words accepted by A. Automaton A is *universal* if $\mathcal{L}(A) = \Sigma^*$.

A *Büchi automaton* $A = \langle Q, \Sigma, \delta, \iota\rangle$ consists of a finite set of *states*, a finite alphabet Σ, a *transition relation* $\delta : \Sigma \to \mathbf{2}^{Q \times \{\geq, >\} \times Q}$, and an *initial state* $\iota \in Q$. We also write $\tau \xrightarrow[d]{a} \tau'$ for $(\tau, d, \tau') \in \delta(a)$. A *run* of A is an infinite sequence $r = \iota \xrightarrow[d_1]{a_1} \tau_1 \xrightarrow[d_2]{a_2} \tau_2 \cdots$. r is *accepting* if $d_i = >$ for infinitely many i. Automaton A *accepts* an infinite word $w \in \Sigma^\omega$ if there is an accepting run $r = \iota \xrightarrow[d_1]{a_1} \tau_1 \xrightarrow[d_2]{a_2} \tau_2 \cdots$ such that $w = a_1 a_2 \cdots$. We denote by $\mathcal{L}(A) = \{w \in \Sigma^\omega \mid A \text{ accepts } w\}$ the *language* accepted by A. Automaton A is *universal* if $\mathcal{L}(A) = \Sigma^\omega$.

Remark. We use this slightly unusual presentation of automata in order to conveniently represent the connection between automata and the size-change abstraction later on. In particular, this connection is the reason for using the symbols $\{\geq, >\}$ instead of $\{0, 1\}$ for (non-)accepting transitions.

2.2 Programs

A *program* $P = \langle St, I, \Sigma, \rho\rangle$ consists of a set of *states* St, a set of initial states $I \subseteq St$, a finite set of *transitions* Σ, and a *labeling function* $\rho : \Sigma \to \mathbf{2}^{St \times St}$, which maps every transition $a \in \Sigma$ to a *transition relation* $\rho(a) \subseteq St \times St$.

We extend the labeling function ρ to finite words over Σ and set $\rho(\pi) = \rho(a_1) \circ \rho(a_2) \circ \cdots \circ \rho(a_l)$ for a finite word $\pi = a_1 a_2 \cdots a_l$. A *computation* of P is a (finite or infinite) sequence $s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots$ such that $s_1 \in I$ and $(s_i, s_{i+1}) \in \rho(a_i)$ for all i. Program P *terminates* if there is no infinite computation of P. A relation $T \subseteq St \times St$ is a *transition invariant* for P if $(\bigcup_{a \in \Sigma} \rho(a))^+ \subseteq T$. For a finite computation $s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots s_{l+1}$ we call l the *length* of the computation.

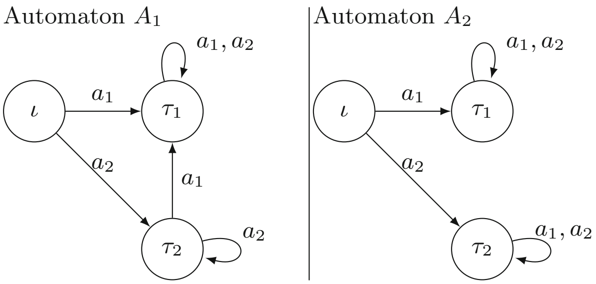

Fig. 2. Pictures of proof structures/transition automata.

Variables and Predicates. A common program model is to consider some finite set of *variables* Var and define the set of states $St = Var \to \alpha$ as the mappings from Var to some *domain* α. Sets of states can then be described by predicates over Var and transition relations by predicates over $Var \cup Var'$, where Var' denotes the set of *primed* versions of the variables in Var. Given a predicate p over Var, we write $\sigma \models p$ for $\sigma \in St$ if p is true when each variable $x \in Var$ is replaced by $\sigma(x)$; given a predicate p over $Var \cup Var'$, we write $\sigma, \varsigma \models p$ for $\sigma, \varsigma \in St$ if p is true when each variable $x \in Var$ is replaced by $\sigma(x)$ and each variable $x' \in Var'$ is replaced by $\varsigma(x)$. Given a set of predicates $Pred$ over Var, we write $Rel(Pred) = \{\sigma \in St \mid \sigma \models p \text{ for all } p \in Pred\}$ for the states which satisfy all predicates in $Pred$. Given a set of predicates $Pred$ over $Var \cup Var'$, we write $Rel(Pred) = \{(\sigma, \varsigma) \in St \times St \mid \sigma, \varsigma \models p \text{ for all } p \in Pred\}$ for the pairs of states which satisfy all predicates in $Pred$. We will also write $Rel_\alpha(Pred)$ in case we want to highlight the domain α.

Example 3. We now express the two programs from Fig. 1 in the above notation. For both programs, we consider the set of variables $Var = \{x, y\}$ and treat N as a symbolic constant. We choose the domain $\alpha = \omega$ according to the type `nat` of x and y. For both programs we model each branch of the if-statement as one transition. We set $P_i = \langle \{x, y\} \to \alpha, Rel_\alpha(\{x = N, y = N\}), \{a_1, a_2\}, \rho_i \rangle$, for $i = 1, 2$, where we define the labeling functions ρ_i using $C = \{x > 0, y > 0\}$:

$$\rho_1(a_1) = Rel_\alpha(C \cup \{x' = x - 1, y' = \mathbf{N}\}), \rho_1(a_2) = Rel_\alpha(C \cup \{x' = x, y' = y - 1\}),$$
$$\rho_2(a_1) = Rel_\alpha(C \cup \{x' = x - 1, y' = \mathbf{y}\}), \rho_2(a_2) = Rel_\alpha(C \cup \{x' = x, y' = y - 1\}),$$

3 Transition Predicate Abstraction

In this section, we review the definitions and results from [26] in order to motivate our generalizations in Sect. 4. The development in [26] also considers fairness requirements, which are not relevant for this paper and therefore left out.

Abstract-Transition Programs. We fix some program $P = \langle St, I, \Sigma, \rho\rangle$. We split up the definition of abstract-transition programs (see Definition 3 of [26]) into two parts: proof structures and proof labelings. A *proof structure* is a finite automaton $A = \langle Q, \Sigma, \delta, \iota, \_\rangle$, where $\delta(a) \subseteq Q \times (Q \setminus \{\iota\})$ for all $a \in \Sigma$. For the moment, we ignore the acceptance condition; we will use it later on. A *proof labeling* $rel : Q \rightarrow \mathbf{2}^{St \times St}$ maps every state $\tau \in Q$ of a proof structure to a *transition relation* $rel(\tau) \subseteq St \times St$. A proof labeling is *inductive* if

$$\begin{aligned} rel(\iota) = Id_{St}, &\qquad \text{and} \\ rel(\tau) \circ \rho(a) \subseteq rel(\tau'), &\qquad \text{for all } (\tau, \tau') \in \delta(a) \text{ and for all } a \in \Sigma, \end{aligned}$$

where Id_{St} is the identity relation over St. An *abstract-transition program* $P^{\#} = (A, rel)$ is a pair of a proof structure A and an inductive proof labeling.

Abstract-transition program are constructed from a fixed finite set of transition predicates that describe transition relations (see Sect. 4 of [26]). The resulting abstract-transition programs have the following properties:

– (P1) The proof structure is a *deterministic* automaton (see Sect. 5.1 of [26]).
– (P2) For every word $a_1 a_2 \cdots a_n$ with $\rho(a_1 a_2 \cdots a_n) \neq \emptyset$ there is a run $\iota \xrightarrow{a_1} \tau_1 \xrightarrow{a_2} \tau_2 \cdots \xrightarrow{a_n} \tau_n$ of A (see Lemma 1 from [26]).
– (P3) Every state $\tau \in Q \setminus \{\iota\}$ is reachable from ι (the reader can check that the abstraction algorithm of [26] starts from the initial state ι and adds only states which are reachable from ι).

We now state the core theorem of [26]; for illustration purposes, we also state its proof, which is based on condition (*), in the notation of this paper:

Theorem 1 (Theorem 1 of [26]). *Let $P^{\#} = (A, rel)$ be an abstract program with property (P2). Then, $\bigcup_{\tau \in Q \setminus \{\iota\}} rel(\tau)$ is a transition invariant for P. If $rel(\tau)$ is well-founded for every state $\tau \in Q \setminus \{\iota\}$, then P terminates.*

Proof. For the first claim, we consider some $(s, s') \in \rho(a_1 a_2 \cdots a_n)$ for some word $a_1 a_2 \cdots a_n$ with $n \geq 1$. By property (P2) we have that there is a run $\iota \xrightarrow{a_1} \tau_1 \xrightarrow{a_2} \tau_2 \cdots \xrightarrow{a_n} \tau_n$ of A. By the definition of an inductive proof labeling we have $\rho(a_1 a_2 \cdots a_n) \subseteq rel(\tau_n)$. Thus, we get that $(s, s') \in rel(\tau_n)$. Hence, we get $(\bigcup_{a \in \Sigma} \rho(a))^+ \subseteq \bigcup_{\tau \in Q \setminus \{\iota\}} rel(\tau)$. The second claim then directly follows from the first claim based on condition (*). □

Example 4. We will define an abstract-transition program for P_1. Let A_1 be the proof structure from Fig. 2. Let rel_1 be the proof labeling defined by $rel_1(\tau_1) = Rel_\alpha(\{x' < x\})$ and $rel_1(\tau_2) = Rel_\alpha(\{x' = x, y' < y\})$, where $\alpha = \omega$. It is easy to verify that rel_1 is inductive. Hence, $P_1^\# = (A_1, rel_1)$ is an abstract-transition program. Moreover, $rel_1(\tau_1)$ and $rel_1(\tau_2)$ are well-founded due to the predicates $x' < x$ and $y' < y$. The abstraction algorithm of [26] precisely computes $P_1^\#$ when called with the set of predicates $Pred = \{x' < x, x' = x, y' < y\}$.

Example 5. We will define an abstract-transition program for P_2. Let A_2 be the proof structure from Fig. 2. Let rel_2 be the proof labeling defined by $rel_2(\tau_1) = Rel_\alpha(\{x' < x\})$ and $rel_2(\tau_2) = Rel_\alpha(\{y' < y\})$, where $\alpha = \omega$. It is easy to verify that rel_2 is inductive. Hence, $P_2^\# = (A_2, rel_2)$ is an abstract-transition program. Moreover, $rel_2(\tau_1)$ and $rel_2(\tau_2)$ are well-founded due to the predicates $x' < x$ and $y' < y$. The abstraction algorithm of [26] precisely computes $P_2^\#$ when called with the set of predicates $Pred = \{x' < x, y' < y\}$.

Remark. The above proof of Theorem 1 only relies on property (P2). However, properties (P1) and (P3) explain the requirement that every non-initial state needs to be labelled by a well-founded relation: by (P3) every state $\tau \in Q \setminus \{\iota\}$ is reachable by some word $a_1 a_2 \cdots a_n$; by (P1) the word $a_1 a_2 \cdots a_n$ necessarily reaches τ; hence, τ needs to be labelled by some well-founded relation. In this paper, we will generalize Theorem 1 of [26] to non-deterministic proof structures; for such proof structures it will make sense to also consider proof labelings where not every state is labelled by some well-founded relation.

Remark. We further note that we can w.l.o.g. strengthen property (P2) to property (P2'): For every word $a_1 a_2 \cdots a_n$ there is a run $\iota \xrightarrow{a_1} \tau_1 \xrightarrow{a_2} \tau_2 \cdots \xrightarrow{a_n} \tau_n$ of A. We show the following: Let $P^\# = (A, rel)$ be an abstract-transition program with property (P2). Then we can extend $P^\#$ to some abstract-transition program (A', rel') with property (P2'). Further, if $rel(\tau)$ is well-founded for every non-initial state τ, then $rel'(\tau)$ is well-founded for every non-initial state τ.

We extend A to A' by adding a sink state $\tau_\emptyset$, which has self-loops for every $a \in \Sigma$; for every state τ and $a \in \Sigma$ we add an a-transition from τ to $\tau_\emptyset$ if τ does not have a a-successor. We extend rel to rel' by setting $rel'(\tau_\emptyset) = \emptyset$. It is easy to see that (P1)–(P3) ensure that rel' is inductive and that (A', rel') has property (P2'). Further $rel'(\tau_\emptyset) = \emptyset$ is well-founded; hence, the second claims holds.

Invariants. An *invariant* for a program $P = \langle St, I, \Sigma, \rho\rangle$ is a set $Inv \subseteq St$ such that (1) $I \subseteq Inv$ and (2) $\{\sigma \in St \mid \text{there is a } \sigma' \in Inv \text{ with } (\sigma', \sigma) \in \rho(a)\} \subseteq Inv$ for all $a \in \Sigma$. For example, $Inv = Rel_\alpha(\{x \leq N, y \leq N\})$ is an invariant for P_1 and P_2. Invariants can be used to strengthen the transition relations of a program by restricting the transition relations to states from the invariant: Given an invariant Inv for P we define $P_{strengthen} = \langle St, I, \Sigma, \rho_{strengthen}\rangle$, where $\rho_{strengthen}(a) = \rho(a) \cap (Inv \times Inv)$ for all $a \in \Sigma$. Clearly, P and $P_{strengthen}$ have the same computations. However, working with $P_{strengthen}$ for termination resp. complexity analysis is often beneficial because of the restricted transition relations. Indeed, strengthening the transition relation is often necessary to find a termination proof. For example, the TERMINATOR termination analyzer [14] alternates between strengthening the transition relation and constructing a transition invariant. Similarly, complexity analyzers from the literature commonly employ invariant analysis as a subroutine either before or during the analysis [1,2,16–18,29,34]. The problem of computing invariants is orthogonal to the development in this paper. In our examples on complexity analysis we assume that appropriate invariants – such as $Inv = Rel_\alpha(\{x \leq N, y \leq N\})$ for P_1 and P_2 – can be computed by standard techniques such as Octagon analysis [24].

4 Transition Abstraction

In this section, we take another view on the result of [26] that we presented in the last section. On the one hand we aim at generalizing the termination analysis of [26] to non-deterministic proof structures. On the other hand we do not only want to reason about a single proof labeling but all possible proof labelings; to this end we will define a minimal inductive proof labeling. We fix a program $P = \langle St, I, \Sigma, \rho\rangle$ for the rest of this section.

A *transition automaton* $A = \langle Q, \Sigma, \delta, \iota, F\rangle$ is a finite automaton, where $\delta(a) \subseteq Q \times (Q \setminus \{\iota\})$ for all $a \in \Sigma$ and $F \subseteq Q \setminus \{\iota\}$. We point out that a transition automaton is a proof structure with final states.

Let $A = \langle Q, \Sigma, \delta, \iota, F\rangle$ be a transition automaton. We define a proof labeling $rel_{min} : Q \to \mathbf{2}^{St \times St}$ which precisely follows the structure of A: We set $rel_{min}(\iota) = Id_{St}$, and for each $\tau \in Q \setminus \{\iota\}$ we set

$$rel_{min}(\tau) = \bigcup_{\text{word } \pi \text{ with } (\iota,\tau)\in\delta(\pi)} \rho(\pi),$$

i.e., $rel_{min}(\tau)$ is the union of the transition relations along all words with a run from the initial state to τ.

We now state the central definition of this section:

Definition 1 (Transition Abstraction). A transition automaton A is a *transition abstraction* of program P if $rel_{min}(\tau)$ is well-founded for each $\tau \in F$.

The notion of transition automata is motivated by Theorem 2, which extends the termination criterion of [26] to non-deterministic proof structures. Proposition 3 below states that Theorem 2 indeed is an extension of Theorem 1 of [26].

Theorem 2. *Let A be a transition automaton that is a transition abstraction of program P. If A is universal, then P terminates.*

Proof (Sketch). The theorem can be proved in the same way as Theorem 1 of [26] whose proof we presented in Sect. 3 based on an application of condition (*); we will later give a proof purely based on automata-theoretic techniques. □

We first show that rel_{min} is the minimal inductive proof labeling:

Proposition 1. *rel_{min} is inductive.*

Proof. We consider some $(\tau, \tau') \in \delta(a)$. We consider some word π with $(\iota, \tau) \in \delta(\pi)$. Then, πa is a word with $(\iota, \tau') \in \delta(\pi a)$. Hence, $\rho(\pi a) \subseteq rel_{min}(\tau')$. Because this holds for all such words π, we get $rel_{min}(\tau) \circ \rho(a) \subseteq rel_{min}(\tau')$. □

Proposition 2. *Let $rel : Q \to \mathbf{2}^{St \times St}$ be some inductive proof labeling. Then, $rel_{min}(\tau) \subseteq rel(\tau)$ for all $\tau \in Q$.*

Proof. We note that $rel_{min}(\iota) = rel(\iota) = Id_{St}$. We will show that for all non-empty words π that $(\iota, \tau) \in \delta(\pi)$ implies $\rho(\pi) \subseteq rel(\tau)$. The proof proceeds by induction on the length of the word. For the induction start, we consider a word $\pi = a$ consisting of a single letter: Because rel is inductive, we have $\rho(a) = Id_{St} \circ \rho(a) = rel(\iota) \circ \rho(a) \subseteq rel(\tau)$ for all $(\iota, \tau) \in \delta(a)$. For the induction step, we consider a word $\pi = \pi' a$ with non-empty π': We fix some $(\iota, \tau) \in \delta(\pi' a)$. There is some $(\tau, \tau') \in \delta(a)$ with $(\tau', \tau) \in \delta(a)$ and $(\iota, \tau') \in \delta(\pi')$. By induction assumption we have $\rho(\pi') \subseteq rel(\tau')$. Because rel is inductive, we have $rel(\tau') \circ \rho(a) \subseteq rel(\tau)$. Thus, $\rho(\pi' a) = \rho(\pi') \circ \rho(a) \subseteq rel(\tau)$. $\square$

With Proposition 2 we are now able to relate transition automata to the abstract-transition programs presented in the last section:

Proposition 3. *Let $A = \langle Q, \Sigma, \delta, \iota, \_ \rangle$ be a proof structure with property (P2'). Let rel be an inductive proof labeling such that $rel(\tau)$ is well-founded for every state $\tau \in Q \setminus \{\iota\}$. With the set of final states $F = Q \setminus \{\iota\}$, the proof structure A is a transition abstraction of program P; further, A is universal.*

Proof. By Proposition 2 we have $rel_{min}(\tau) \subseteq rel(\tau)$ for all $\tau \in Q$. Hence, A is a transition automaton. By property (P2'), the automaton A has a run for every word; with $F = Q \setminus \{\iota\}$ each such run is accepting. Hence, A is universal. $\square$

Example 6. In Examples 4 and 5 we have argued that $P_1^{\#} = (A_1, rel_1)$ and $P_2^{\#} = (A_2, rel_2)$ are abstract-transition programs for P_1 resp. P_2. We now consider A_1 and A_2 as transition automata, defining the final states by $F = \{\tau_1, \tau_2\}$. By Proposition 3, A_1 and A_2 are transition abstractions for P_1 resp. P_2 and Theorem 2 can be applied.

We now define a transition automaton for program P_1 that is different from the transition automaton A_1 considered in Example 6:

Example 7. Let A_3 be the automaton from Fig. 2 with the set of final states $F = \{\tau_1, \tau_2\}$. We now argue that the transition automaton A_3 is a transition abstraction of P_1. In order to reason about the well-foundedness of $rel_{min}(\tau_1)$ and $rel_{min}(\tau_2)$, which are required by the definition of transition abstraction, we make use of Proposition 2 as a proof principle: it is sufficient to define an inductive proof labeling rel_3 and argue that $rel_3(\tau_1)$ and $rel_3(\tau_2)$ are well-founded.

We define rel_3 by setting $rel_3(\tau_1) = Rel_\alpha(\{x' < x\})$ and $rel_3(\tau_2) = Rel_\alpha(\{y' < y\})$ with $\alpha = \omega$. It is easy to verify that rel_3 is inductive. Moreover, $rel_3(\tau_1)$ and $rel_3(\tau_2)$ are well-founded due to the predicates $x' < x$ and $y' < y$. We conclude that A_3 is a transition abstraction of P_1. We observe that automaton A_3 (resp. A_3') is not universal, and Theorem 2 cannot be applied.

Remark. We relate A_3 to the abstraction algorithm of [26]. We extend A_3 to the automaton A_3' by adding a non-final state τ_{true}; we add an a_1-transition from τ_2

to τ_{true} and self-loops to τ_{true} for a_1 and a_2. We set $rel_3(\tau_{true}) = Rel_\alpha(\{true\}) = St \times St$ (note that $St \times St$ is not well-founded). The abstraction algorithm of [26] will exactly compute the abstract-transition program $P_3^\# = (A_3', rel_3)$ when called with the set of predicates $Pred = \{x' < x, y' < y\}$; we work with automaton A_3 instead of A_3' because it has one state less and is easier to represent.

Remark. In the next subsection, we will establish the more general criterion of factor-termination, which is satisfied by automaton A_3 (resp. A_3'). Hence, we obtain a new termination proof for the program P_1, which has the advantage to use fewer predicates than the termination proof in Example 4: we contrast the set of predicates $Pred = \{x' < x, y' < y\}$ used in Example 7 with the set $Pred = \{x' < x, x' = x, y' < y\}$ used in Example 4.

4.1 Factor Termination

In this section, we introduce the criterion of factor-termination. We first introduce the criterion and then argue that factor-termination is a more general termination criterion than universality. Finally, we state that factor-termination is in fact the most general termination criterion based on transition abstraction.

The intuition behind the criterion of factor-termination is as follows: Given a transition automaton $A = \langle Q, \Sigma, \delta, \iota, F \rangle$, we directly use the well-foundedness of the relations $rel_{min}(\tau)$, for final state $\tau \in F$. We check for every infinite word $\pi \in \Sigma^\omega$ if there is a $\tau \in F$ and a factorization $\pi = \pi_0\pi_1\pi_2 \cdots$ into finite words π_i such that A has a run from ι to τ on π_i for all $i \geq 1$. Such a factorization implies that there cannot be an infinite sequence of states $s_1 s_2 \ldots$ with $(s_i, s_{i+1}) \in \delta(\pi_i) \subseteq rel_{min}(\tau)$ because this would contradict the well-foundedness of $rel_{min}(\tau)$.

We implement the above idea with Büchi-automata. We fix some transition automaton $A = \langle Q, \Sigma, \delta, \iota, F \rangle$ for which we will define a Büchi-automaton $\mathcal{F}(A)$, which is composed of Büchi-automata A_τ, for every $\tau \in F$, and an additional initial state κ. $\mathcal{F}(A)$ can non-deterministically wait in κ a finite amount of time before moving to one of the automata A_τ. Each A_τ checks for a factorization with regard to $\tau \in F$. We first formally define the automata A_τ and then $\mathcal{F}(A)$.

We start with an intuition for the construction of A_τ. We take a copy of A where all copied transitions are non-accepting. We obtain A_τ by adding additional accepting transitions that allow the automaton A_τ to move back to the initial state whenever it could move to τ. The additional transitions allow A_τ to guess the beginning of a new factor; the Büchi-condition guarantees that an accepting run factorizes an infinite word into infinitely many finite words.

Formally, we define $A_\tau = \langle Q \times \{\tau\}, \Sigma, \delta_\tau, (\iota, \tau) \rangle$, where for all $a \in \Sigma$ we set

$$\delta_\tau(a) = \{((\tau', \tau), \geq, (\tau'', \tau)) \mid (\tau', \tau'') \in \delta(a)\} \cup \{((\tau', \tau), >, (\iota, \tau)) \mid (\tau', \tau) \in \delta(a)\}.$$

We state the main property of the automata A_τ:

Proposition 4. *A_τ accepts $\pi \in \Sigma^\omega$ iff there is a factorization $\pi = \pi_1\pi_2 \cdots$ into finite words π_i such that A has a run from ι to τ on π_i for all i.*

Proof. Let r be an accepting run of A_τ on π. Hence, we can factor $\pi = \pi_1\pi_2\cdots$ into finite words π_i such that the accepting transitions of r exactly correspond to the last letters of the words π_i. We observe that the only accepting transitions are of shape $((\tau', \tau), >, (\iota, \tau))$ for $(\tau', \tau) \in \delta(a)$ (we denote this condition by (#)). Further, automaton A_τ mimics A on the non-accepting transitions. Hence, on each word π_i the run r mimics a run of A except for the last transition; however, the condition (#) guarantees that A can move to τ with the last letter of π_i. □

The *factorization automaton* is the Büchi-automaton $\mathcal{F}(A) = \langle G, \Sigma, \Gamma, \kappa\rangle$, where the set of states $G = (Q \times F) \cup \{\kappa\}$ consists of pairs of an automaton state and a final state plus a fresh initial state κ. We define the transition relation Γ by $\Gamma(a) = \Gamma_1(a) \cup \Gamma_2(a) \cup \Gamma_3(a)$ for all $a \in \Sigma$, where

$$\Gamma_1(a) = \bigcup_{\tau \in F} \delta_\tau(a), \Gamma_2(a) = \{(\kappa, \geq, \kappa)\}, \text{ and } \Gamma_3(a) = \{(\kappa, \geq, (\iota, \tau)) \mid \tau \in F\}.$$

The factorization automaton $\mathcal{F}(A)$ can be understood as the disjoint union of the initial state κ and the Büchi-automata A_τ; the state κ allows $\mathcal{F}(A)$ to wait in κ a finite amount of time before moving to the initial state of some A_τ.

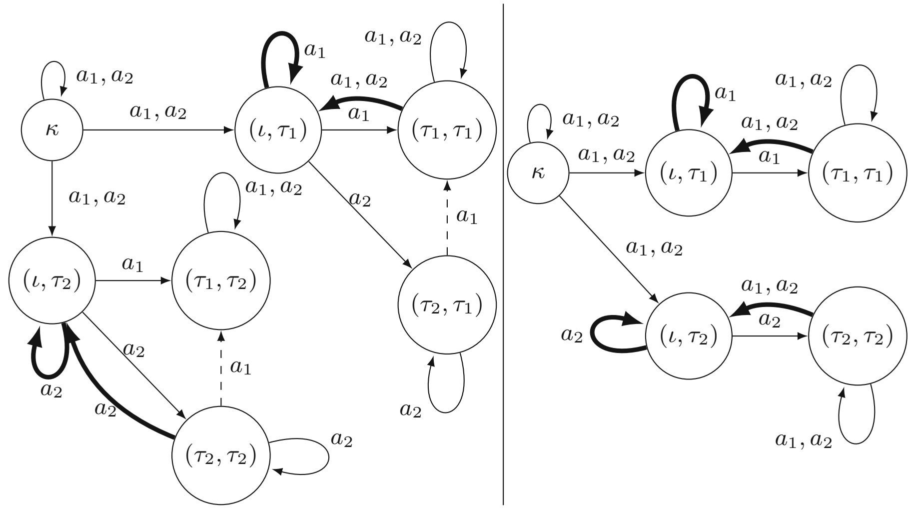

Fig. 3. On the left: Automata $\mathcal{F}(A_1)$ and $\mathcal{F}(A_3)$, which have the same states and transitions except for the dashed transitions which only belong to $\mathcal{F}(A_1)$. On the right: Automaton $\mathcal{F}(A_2)$. Bold arrows denote accepting transitions.

Example 8. We draw the factor-automata of A_1, A_2 and A_3 in Fig. 3.

We are now able to formally state our new termination criterion: Transition automaton A satisfies the *factor-termination* criterion if $\mathcal{F}(A)$ is universal. This notion is justified by Theorem 3 below:

Theorem 3. *Let $A = \langle Q, \Sigma, \delta, \iota, F \rangle$ be a transition automaton and let $P = \langle St, I, \Sigma, \rho \rangle$ be a program such that A is a transition abstraction of P. If A satisfies the factor-termination criterion, then P terminates.*

Proof. We assume that $\mathcal{F}(A)$ is universal and that P does not terminate. Then there is an infinite computation $t = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots$ of P. We consider the associated word $\pi = a_1 a_2 \cdots$. Because $\mathcal{F}(A)$ is universal, the word π is accepted by some run r. Word $\pi = \pi_a \pi_b$ can be split in a finite prefix π_a and an infinite suffix π_b such that $\mathcal{F}(A)$ stays in κ while reading π_a before leaving κ and then reading π_b. We further see that while reading π_b, $\mathcal{F}(A)$ stays within A_τ for some $\tau \in F$. By Proposition 4, there is a factorization $\pi_b = \pi_1 \pi_2 \cdots$ such that A has a run on each π_j from ι to τ. We split t into corresponding subcomputations

$$t_j = s_{i_j} \xrightarrow{a_{i_j}} \cdots s_{i_{j+1}-1} \xrightarrow{a_{i_{j+1}-1}} s_{i_{j+1}}$$

with $\pi_j = a_{i_j} \cdots a_{i_{j+1}-1}$. Hence, we have $(s_{i_j}, s_{i_{j+1}}) \in \rho(\pi_j) \subseteq rel_{min}(\tau)$ for all j. This gives us an infinite sequence $s_{i_1} s_{i_2} \ldots$ with $(s_{i_j}, s_{i_j+1}) \in rel_{min}(\tau)$. However, this results in a contradiction, because $rel_{min}(\tau)$ is well-founded by the assumption that A is a transition abstraction of P. $\square$

Next, we show that the universality of a transition automaton A implies the factor-termination of A; the proof uses the fundamental fact that a Büchi-automaton is universal iff it accepts all ultimately-periodic words:

Lemma 1. *Let A be a transition automaton. If A is universal, then A satisfies the factor-termination criterion.*

Proof. We assume that A is universal. We will show that $\mathcal{F}(A)$ accepts all ultimately-periodic words. Let u, v be two finite words over Σ and consider the ultimately-periodic word uv^ω. Since A is universal there is an accepting run of A ending in some final state $\tau \in F$. We will use this run to construct an accepting run of $\mathcal{F}(A)$. In order to accept uv^ω, the automaton $\mathcal{F}(A)$ reads the word u staying in the initial state κ and moving to (ι, τ) with the last letter of u (we tacitly assume here that the length of u is at least one; however this is without loss of generality as we can consider the word uv instead of u); $\mathcal{F}(A)$ then reads the word v, mimicking the accepting run of A in A_τ, and moving to state (ι, τ) with the last letter of v; A_τ then reads the next occurrence of v in the same way; we note that the last transition, with which the automaton returns to the initial state (ι, τ), is accepting; thus the constructed run on uv^ω is accepting. $\square$

Remark. The combination of Theorem 3 and Lemma 1 provides an alternative proof of Theorem 2. We highlight that the proof of Lemma 1 proceeds purely by automata-theoretic techniques and does not make use of condition (*); in particular, Ramsey's theorem is not needed to prove Theorem 1 of [26].

We now establish that factor-termination is a *strictly* more general termination criterion than universality:

Example 9. Let A_3 be the automaton from Example 7, where we have established that A_3 is a transition abstraction of P_1 and that A_3 is not universal. We have drawn $\mathcal{F}(A_3)$ in Fig. 3. It remains to argue that $\mathcal{F}(A_3)$ is universal.

We show that $\mathcal{F}(A_3)$ is universal by a case distinction: Assume a word contains infinitely many a_1. $\mathcal{F}(A_3)$ waits for the first a_1 and moves to (ι, τ_1) just before the first a_1; with the first a_1, $\mathcal{F}(A_3)$ moves to (τ_1, τ_1); then $\mathcal{F}(A_3)$ again waits for the next a_1, moving to (ι, τ_1) just before the next a_1, and so on. An infinite word that does not contain infinitely many a_1, only contains a_2 from some point on; $\mathcal{F}(A_3)$ accepts such a word by waiting in the initial state κ until there are only a_2 left and then moves to (ι, τ_2); $\mathcal{F}(A_3)$ then can stay in (ι, τ_2) while continuing to read the letters a_2.

We finally state that factor-termination is the most general termination criterion based on transition abstraction:

Theorem 4. *Let A be a transition automaton that does not satisfy the factor-termination criterion. Then there is a program P such that A is a transition abstraction of P, but P does not terminate.*

We prove Theorem 4 (see Corollary 2) and further results in Sect. 6 based on the close relationship of factorization automata and the size-change abstraction. We first introduce the size-change abstraction in the next subsection.

5 Size-Change Abstraction

Size-change abstraction (SCA) can be seen as an instantiation of (transition-) predicate abstraction with a restricted class of predicates: a *size-change predicate* over some set of variables Var is an inequality $x \triangleright y'$ with $x, y \in Var$, where $\triangleright$ is either $>$ or $\geq$ (recall that $y' \in Var'$ denotes the primed version of y). A *size-change relation* (SCR) is a set of size-change predicates over Var. A *size-change system* (SCS) $S = \langle Var, \Sigma, \lambda \rangle$ consists of a set of *variables* Var, a finite set of *transitions* Σ and a *labeling function* λ, which maps every transition $a \in \Sigma$ to a SCR $\lambda(a)$ over Var.

The SCA methodology requires an abstraction mechanism that abstracts programs to SCSs. Various static analyzes have been proposed in the literature which perform such an abstraction [3,9,10,20,22,23,31,34]. In this paper, we are not concerned with how to abstract programs to SCSs (and thus we do not describe an abstraction mechanism for programs). Rather, we will use results on the strength of SCA [12,21] for the analysis of transition automata.

Results on the strength of SCA directly interpret SCSs as (abstract) programs, which can be seen as 'most general programs' that satisfies all the size-change predicates. We now state the interpretation of SCSs as programs for which we make use of the variable mappings and predicate interpretations defined in

Sect. 3. An SCS $S = \langle Var, \Sigma, \lambda \rangle$ defines a program $\mathcal{P}_\alpha(S) = \langle St, St, \Sigma, \rho \rangle$, where $St = Var \rightarrow \alpha$ and $\rho(a) = Rel_\alpha(\lambda(a))$ for all $a \in \Sigma$; the program $\mathcal{P}_\alpha(S)$ is parameterized by some domain α that we require to be well-founded.

We will build on theoretical results for SCA which have been obtained by automata-theoretic techniques (we refer the interested reader to [13] for an overview). We begin by stating the syntactic termination criterion of [22]. Let $S = \langle Var, \Sigma, \lambda \rangle$ be an SCS. We define the Büchi-automaton $DESC(S) = \langle D, \Sigma, \mu, \kappa \rangle$, where the set of states $D = Var \cup \{\kappa\}$ consists of the variables and a fresh initial state κ, the alphabet Σ is the same as the alphabet of S, the transition relation μ is defined by $\mu(a) = \mu_1(a) \cup \mu_2(a) \cup \mu_3(a)$ for all $a \in \Sigma$, where $\mu_1(a) = \lambda(a)$, $\mu_2(a) = \{(\kappa, \geq, \kappa)\}$ and $\mu_3(a) = \{(\kappa, \geq, x) \mid x \in Var\}$. Intuitively the automaton $DESC(S)$ waits a finite amount of time in the initial state κ and then starts to trace a chain of inequalities $x_1 \triangleright_1 x_2 \triangleright_2 x_3 \cdots$ between the variables of S. The Büchi-acceptance condition ensures that $\triangleright_i = >$ infinitely often. Now we are ready to define the syntactic termination criterion of [22]: SCS S has *infinite descent* if $DESC(S)$ is universal. This criterion is sound and complete:

Theorem 5 **([21,22]).** *S has infinite descent iff $\mathcal{P}_\alpha(S)$ terminates over all domains α. Moreover, if S does not have infinite descent, then $\mathcal{P}_\alpha(S)$ does not terminate for some domain $\alpha < \omega$ (i.e., $\mathcal{P}_\alpha(S)$ does not terminate when variables take values in some initial segment $\alpha = [0, N]$ of the natural numbers).*

While the original motivation for studying SCA has been termination analysis, we recently extended the theoretical results on SCA to complexity analysis:

Theorem 6 **([12]).** *Let S be an SCS that is size-change terminating. Then there effectively is a rational number $z \geq 1$ such that the length of the longest run of $\mathcal{P}_{[0,N]}(S)$ is of asymptotic order $\Theta(N^z)$ for natural numbers N.*

Our result provides a complete characterization of the complexity bounds arising from SCA and gives an effective algorithm for computing the exact asymptotic bound of a given abstract program. The proof of Theorem 6 proceeds by rephrasing the question of complexity analysis for SCSs as a question about the asymptotic behaviour of max-plus automata. The main induction of the proof relies on the Factorization Forest Theorem [28], which is a powerful strengthening of Ramsey's Theorem for finite semigroups that offers a deep insight into their structure (see [11] for an overview).

6 Canonical Programs for Transition Automata

In this section, we will relate transition abstraction and SCA. We will describe the extraction of a size-change system $S = \mathcal{S}(A)$ from a transition automaton A. We will argue that the associated program $\mathcal{P}_\alpha(S)$ is *canonical* for A. We will prove three results that justify the use of the word 'canonical':

1. We show that the criterion of factor-termination for A agrees with the criterion of infinite descent for S (Corollary 1).

2. We show that A is a transition abstraction of $\mathcal{P}_\alpha(S)$ for all domains α (Proposition 5). This result allows us to establish that factor-termination is the most general termination criterion (Corollary 2).
3. If A is a transition abstraction for some program P, then every run of P can be mimicked by a run of $\mathcal{P}_\alpha(S)$, where the domain α depends on P and needs to be chosen appropriately (Lemma 3). This result allows us to transfer the result on complexity analysis for SCSs (see Theorem 6) to transition automata (Theorem 7).

6.1 Extracting Size-Change Systems from Transition Automata

We fix some transition automaton $A = \langle Q, \Sigma, \delta, \iota, F\rangle$. Let $\mathcal{F}(A) = \langle G, \Sigma, \Gamma, \kappa\rangle$ be the associated factorization automaton, where $G = Q \times F \cup \{\kappa\}$ and $\Gamma(a) = \Gamma_1(a) \cup \Gamma_2(a) \cup \Gamma_3(a)$ for all $a \in \Sigma$. We extract the associated size-change system from $\mathcal{F}(A)$ and define $\mathcal{S}(A) = \langle \mathit{Var}, \Sigma, \lambda\rangle$ by setting $\mathit{Var} = Q \times F$ and $\lambda(a) = \Gamma_1(a)$ for all $a \in \Sigma$ (i.e., $\mathcal{S}(A)$ is obtained from automaton $\mathcal{F}(A)$ by restriction to the non-initial states).

Example 10. We consider the transition automaton A_2. We have drawn $\mathcal{F}(A_2)$ in Fig. 3. We now state the size-change system extracted from $\mathcal{F}(A_2)$: We have $\mathcal{S}(A_2) = \langle\{\iota, \tau_1, \tau_2\} \times \{\tau_1, \tau_2\}, \{a_1, a_2\}, \lambda\rangle$, where λ is given by

- $\lambda(a_1) = \{(\iota, \tau_1) \geq (\tau_1, \tau_1)', (\tau_1, \tau_1) \geq (\tau_1, \tau_1)', (\tau_2, \tau_2) \geq (\tau_2, \tau_2)',$
  $(\iota, \tau_1) > (\iota, \tau_1)', (\tau_1, \tau_1) > (\iota, \tau_1)', (\tau_2, \tau_2) > (\iota, \tau_2)'\},$
- $\lambda(a_2) = \{(\tau_1, \tau_1) \geq (\tau_1, \tau_1)', (\iota, \tau_2) \geq (\tau_2, \tau_2)', (\tau_2, \tau_2) \geq (\tau_2, \tau_2)',$
  $(\tau_1, \tau_1) > (\iota, \tau_1)', (\iota, \tau_2) > (\iota, \tau_2)', (\tau_2, \tau_2) > (\iota, \tau_2)'\}.$

Example 11. We consider the transition automaton A_3. We have drawn $\mathcal{F}(A_3)$ in Fig. 3. We now state the size-change system extracted from $\mathcal{F}(A_3)$. We have $\mathcal{S}(A_3) = \langle\{\iota, \tau_1, \tau_2\} \times \{\tau_1, \tau_2\}, \{a_1, a_2\}, \lambda\rangle$, where λ is given by

- $\lambda(a_1) = \{(\iota, \tau_1) \geq (\tau_1, \tau_1)', (\tau_1, \tau_1) \geq (\tau_1, \tau_1)', (\iota, \tau_2) \geq (\tau_1, \tau_2)',$
  $(\tau_1, \tau_2) \geq (\tau_1, \tau_2)', (\iota, \tau_1) > (\iota, \tau_1)', (\tau_1, \tau_1) > (\iota, \tau_1)'\},$
- $\lambda(a_2) = \{(\tau_1, \tau_1) \geq (\tau_1, \tau_1)', (\iota, \tau_1) \geq (\tau_2, \tau_1)', (\tau_2, \tau_1) \geq (\tau_2, \tau_1)',$
  $(\tau_1, \tau_2) \geq (\tau_1, \tau_2)', (\iota, \tau_2) \geq (\tau_2, \tau_2)', (\tau_2, \tau_2) \geq (\tau_2, \tau_2)',$
  $(\tau_1, \tau_1) > (\iota, \tau_1)', (\iota, \tau_2) > (\iota, \tau_2)', (\tau_2, \tau_2) > (\iota, \tau_2)'\}.$

We comment on the intuition behind the definition of the SCS $S = \mathcal{S}(A)$. The underlying idea has been to obtain a close correspondence between $\mathit{DESC}(S)$ and $\mathcal{F}(A)$. Indeed, $\mathit{DESC}(S)$ and $\mathcal{F}(A)$ are almost identical, the only difference is that the initial state of $\mathit{DESC}(S)$ allows moving to every state, whereas the initial state of $\mathcal{F}(A)$ only allows moving to the initial states of the components A_τ. However, this difference does not change the set of accepted words, as we prove in the next lemma:

Lemma 2. *Let $S = \mathcal{S}(A)$ be the SCS extracted from A. Then $\mathcal{L}(\mathcal{F}(A)) = \mathcal{L}(\mathit{DESC}(S))$.*

Proof. We recall $DESC(S) = \langle D, \Sigma, \mu, \kappa \rangle$, where $D = Var \cup \{\kappa\}$ and $\mu(a) = \mu_1(a) \cup \mu_2(a) \cup \mu_3(a)$ for all $a \in \Sigma$. We see that both automata have the same set of states $G = D = Q \times F \cup \{\kappa\}$. From the definition of $\mathcal{F}(A)$ and $DESC(S)$ we further have that $\Gamma_1(a) = \mu_1(a)$, $\Gamma_2(a) = \mu_2(a)$ and $\Gamma_3(a) \subseteq \mu_3(a)$ for all $a \in \Sigma$.

Thus, we get $\mathcal{L}(\mathcal{F}(A)) \subseteq \mathcal{L}(DESC(S))$ because every run of A is also a run of $DESC(S)$. We now show $\mathcal{L}(\mathcal{F}(A)) \supseteq \mathcal{L}(DESC(S))$: Let π be some word accepted by $DESC(S)$ and let r be an accepting run of $DESC(S)$ on π. We can choose some factorization $\pi = \pi_1\pi_2$ such that the last transition in r when reading π_1 is accepting. We note that after reading π_1, $DESC(S)$ must be in some state $(\iota, \_)$ because accepting transition always move to some state where the first component is ι. We further note that while reading π_2, $DESC(S)$ only uses transitions from μ_1, because there is no transition returning to κ. Hence, the accepting run r of $DESC(S)$ can be mimicked by $\mathcal{F}(A)$ as follows: $\mathcal{F}(A)$ waits in the initial state κ while reading π_1 and then moves to the state $(\iota, \_)$ with the last letter of π_1. After that $\mathcal{F}(A)$ follows the accepting run of $DESC(S)$ on π_2, which can be done because of $\Gamma_1 = \mu_1$. □

As immediate corollary we get the equivalence of the termination conditions:

Corollary 1. *A has factor termination iff S has infinite descent.*

6.2 Factor-Termination Is the Most General Termination Criterion

We consider the size-change system $S = \mathcal{S}(A)$ extracted from transition automaton A. Our next result is that A is a transition abstraction for the program $\mathcal{P}_\alpha(S)$ associated to S. The crucial insight is that S exactly implements the minimal requirements to satisfy the condition of transition abstraction: the inequalities of S exactly follow the transition relation of A, where strict inequalities ensure that the value of variable (ι, τ) decreases iff A visits an accepting state τ.

Proposition 5. *A is a transition abstraction of $\mathcal{P}_\alpha(S)$ for all domains α.*

Proof. Let α be some well-founded domain. We will show that A is a transition abstraction of $\mathcal{P}_\alpha(S)$ using Proposition 2 as proof principle. For this we define a size-change relation T_τ for each $\tau \in Q \setminus \{\iota\}$. We set $T_\tau = \{(\iota, \tau') \geq (\tau, \tau') \mid \tau' \in F\} \cup T_\tau^{dec}$, where $T_\tau^{dec} = \{(\iota, \tau) > (\iota, \tau)\}$, if $\tau \in F$, and $T_\tau^{dec} = \emptyset$, otherwise. It is easy to check that we have $Rel_\alpha(T_\tau) \circ Rel_\alpha(\lambda(a)) \subseteq Rel_\alpha(T_{\tau'})$ for all $(\tau, \tau') \in \delta(a)$. We now apply Proposition 2 and get $rel_{min}(\tau) \subseteq Rel_\alpha(T_\tau)$ for all $\tau \in Q$.

It remains to argue that the relations $rel_{min}(\tau)$ are well-founded for all $\tau \in F$. This follows from $rel_{min}(\tau) \subseteq Rel_\alpha(T_\tau)$ and the fact that $Rel_\alpha(T_\tau)$ is well-founded due to the predicate T_τ^{dec}, which ensures the decrease of variable (ι, τ). □

We are now in a position to prove Theorem 4, i.e., that factor termination is the most general termination criterion for transition abstraction:

Corollary 2. *Let A be a transition automaton that does not satisfy the factor-termination criterion. Then A is a transition abstraction of $\mathcal{P}_\alpha(S)$ for all domains α, but $\mathcal{P}_\alpha(S)$ does not terminate for some $\alpha < \omega$.*

Proof. From Corollary 1 we get that S does not satisfy the infinite descent criterion because A does not satisfy the factor-termination criterion. By Theorem 5 we know that the program $\mathcal{P}_\alpha(S)$ does not terminate for some $\alpha < \omega$ because S does not size-change terminate. We have that A is a transition abstraction of $\mathcal{P}_\alpha(S)$ by Proposition 5. □

6.3 Complexity Analysis with Transition Automata

Let $A = \langle Q, \Sigma, \delta, \iota, F\rangle$ be a transition automaton and $P = \langle St, I, \Sigma, \rho\rangle$ be a program such that A is a transition abstraction of P. Let $S = \mathcal{S}(A) = \langle Var, \Sigma, \lambda\rangle$ be the SCS extracted from A. We will show that every run of P can be mimicked by a run of $\mathcal{P}_\alpha(S)$, where the domain α depends on P and needs to be chosen appropriately. We first introduce the machinery necessary to define α.

We define the *height* of a transition abstraction as the maximum of the heights of the well-founded relations $rel_{min}(\tau)$, i.e., we set

$$height(A, P) = \max_{\tau \in F} \|rel_{min}(\tau)\| .$$

We set $height^\bullet(A, P) = height(A, P) + 1$; we work with $height^\bullet(A, P)$, which differs from $height(A, P)$ by plus one for technical convenience; however, the difference of plus one is not important for our results on asymptotic complexity analysis.

We introduce another auxiliary definition. For every pair $(\tau', \tau) \in Q \times F$ we define a relation $Succ_P(\tau', \tau) \subseteq St \times St$ by setting

$$Succ_P(\tau', \tau) = \bigcup_{\text{word } \pi \text{ with } (\tau', \tau) \in \delta(\pi)} \rho(\pi).$$

We note that $Succ_P(\iota, \tau) = rel_{min}(\tau)$ for all $\tau \in F$.

For every pair $(\tau', \tau) \in Q \times F$ we define a function $rank_{\tau',\tau} : St \to height^\bullet(A, P)$ that maps a state $s \in St$ to an ordinal below $height^\bullet(A, P)$, by setting

$$rank_{\tau',\tau}(s) = \sup_{(s,s') \in Succ_P(\tau',\tau)} \|s'\|_{rel_{min}(\tau)} + 1,$$

where the sup over the empty set evaluates to 0. The following proposition is immediate from the definitions:

Proposition 6. *We have $rank_{\iota,\tau}(s) = \|s\|_{rel_{min}(\tau)}$ for all $s \in St$.*

Proof. Let $s \in St$ be some state. From the definition of $Succ_P$ we get $Succ_P(\iota,\tau) = rel_{min}(\tau)$. Thus, we get $rank_{\iota,\tau}(s) = \sup_{(s,s')\in Succ_P(\iota,\tau)} \|s'\|_{rel_{min}(\tau)} + 1 = \sup_{(s,s')\in rel_{min}(\tau)} \|s'\|_{rel_{min}(\tau)} + 1 = \|s\|_{rel_{min}(\tau)}$. ☐

For every $s \in St$ we define a valuation $\sigma_s : Q \times F \to height^\bullet(A,P)$ by setting $\sigma_s(\tau',\tau) = rank_{\tau',\tau}(s)$.

Lemma 3. *Let $\alpha = height^\bullet(A,P)$. For all pairs of states $(s,s') \in \rho(a)$, where $a \in \Sigma$, we have $(\sigma_s,\sigma_{s'}) \in Rel_\alpha(\lambda(a))$.*

Proof. Let $a \in \Sigma$ be some transition and let $(s,s') \in \rho(a)$ be a pair of states in the associated transition relation.

We consider an inequality $(\tau,\tau'') \geq (\tau',\tau'')' \in \lambda(a)$. By definition of $\lambda(a)$ we have $(\tau,\tau') \in \delta(a)$. From this we get $\{(s,s')\} \circ Succ_P(\tau',\tau'') \subseteq Succ_P(\tau,\tau'')$ because for every word π such that $(\tau',\tau'') \in \delta(\pi)$ we have that $(\tau,\tau'') \in \delta(a\cdot\pi)$ and thus $(s',s'') \in \rho(\pi)$ implies $(s,s'') \in \rho(a\cdot\pi)$. Hence, we get $\sigma_s(\tau,\tau'') = rank_{\tau,\tau''}(s) = \sup_{(s,s'')\in Succ_P(\tau,\tau'')} \|s''\|_{rel_{min}(\tau'')} + 1 \geq \sup_{(s',s'')\in Succ_P(\tau',\tau'')} \|s''\|_{rel_{min}(\tau'')} + 1 = rank_{\tau',\tau''}(s') = \sigma_{s'}(\tau',\tau'')$.

We consider an inequality $(\tau',\tau) > (\iota,\tau)' \in \lambda(a)$. By definition of $\lambda(a)$ we have $(\tau',\tau) \in \delta(a)$. From this we get $(s,s') \in \rho(a) \subseteq Succ_P(\tau',\tau)$. From Proposition 6 we have $rank_{\iota,\tau}(s') = \|s'\|_{rel_{min}(\tau)}$. Hence, we get $\sigma_s(\tau',\tau) = rank_{\tau',\tau}(s) = \sup_{(s,s'')\in Succ_P(\tau',\tau)} \|s''\|_{rel_{min}(\tau)} + 1 > \|s'\|_{rel_{min}(\tau)} = rank_{\iota,\tau}(s') = \sigma_{s'}(\iota,\tau)$. ☐

We immediately obtain the following corollary:

Corollary 3. *Let $\alpha = height^\bullet(A,P)$. Let $s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots$ be a computation of P. Then, $\sigma_{s_1} \xrightarrow{a_1} \sigma_{s_2} \xrightarrow{a_2} \cdots$ is a computation of $\mathcal{P}_\alpha(S)$.*

Finally, we are in a position to transfer Theorem 6:

Theorem 7. *Let A be a transition automaton that satisfies the factor-termination termination criterion. Let $S = \mathcal{S}(A)$. Let z be the rational number obtained from Theorem 6 for S.*

Let $P = \langle St, I_N, \Sigma, \rho\rangle$ be a program whose set of initial states I_N is parameterized by natural number $N \in \mathbb{N}$, such that A is a transition abstraction of P and $height(A,P) = O(N)$. Then, the length of the longest computation of P is of asymptotic order $O(N^z)$.

Moreover, A is a transition abstraction for $\mathcal{P}_{[0,N]}(S)$ and the length of the longest computation of $\mathcal{P}_{[0,N]}(S)$ is of asymptotic order $\Theta(N^z)$.

Proof. By Proposition 5, A is a transition abstraction of $\mathcal{P}_{[0,N]}(S)$ for all $N \in \mathbb{N}$. From Theorem 6 we have that the longest computation of $\mathcal{P}_{[0,N]}(S)$ is of asymptotic order $\Theta(N^z)$.

Because of $height(A,P) = O(N)$, we can find some $a,b \in \mathbb{N}$ such that $height(A,P) \leq a\cdot N + b$ for all $N \in \mathbb{N}$. By Corollary 3, for every computation of P_N there is a computation of $\mathcal{P}_{[0,a\cdot N+b]}(S)$ of equal length. Hence, the longest computation of P_N is of asymptotic order $O((a\cdot N+b)^z) = O(N^z)$. ☐

We highlight that Theorem 7 gives a complete characterization of the complexity bounds obtainable with transition abstraction and provides an effective algorithm for computing these complexity bounds.

Theorem 7 allows us to derive the precise complexity for P_1 and P_2:

Example 12. We consider the size-change system $S = \mathcal{S}(A_2)$, which we have extracted in Example 10 from transition automaton A_2. Theorem 6 allows us to derive that $\mathcal{P}_{[0,N]}(S)$ has complexity $\Theta(N)$. In Example 5 we defined an abstract-transition program (A_2, rel_2) for P_2; the inductive proof labeling rel_2 in conjunction with the invariant $Inv = Rel_\alpha(\{x \leq N, y \leq N\})$ implies that $height(A_2, P_2) = N$. Hence, we can apply Theorem 7 and infer that P_2 has complexity $O(N)$, which is the precise asymptotic complexity of P_2.

We consider the size-change system $S = \mathcal{S}(A_3)$, which we have extracted in Example 11 from transition automaton A_3. Theorem 6 allows us to derive that $\mathcal{P}_{[0,N]}(S)$ has complexity $\Theta(N^2)$. In Example 4 we defined an abstract-transition program (A_1, rel_1) for P_1; the inductive proof labeling rel_1 in conjunction with the invariant $Inv = Rel_\alpha(\{x \leq N, y \leq N\})$ implies that $height(A_1, P_1) = N$. Hence, we can apply Theorem 7 and infer that P_2 has complexity $O(N^2)$, which is the precise asymptotic complexity of P_2.

7 Future Directions and Conclusion

In this paper, we have established a new connection between transition automata and the size-change abstraction. Our results suggest that all tools which implement termination analysis with transition invariants based on an inductive argument (such as Terminator) can be retro-fitted to be complexity analyzers, which is an interesting direction for further research: While this paper has investigated what information can be extracted from a fixed proof (i.e., from a fixed set of transition predicates), there is also the question of what strategy for predicate selection gives the best results. We have seen that the predicates $x' < x$ and $y' < y$ allow inferring the linear complexity of P_2; these predicates are simple and can be extracted from the if- resp. else branch of P_2 by simple heuristics. On the other hand, the predicate $x' + y' < x + y$ allows establishing the linear complexity of P_2 using a single predicate; this predicate, however, is more complex and requires more complicated heuristics for extraction. Finding the right balance in predicate selection is an interesting topic for future research.

Ranking function construction is an alternative technique for termination proofs: [33] states a complete construction for deterministic size-change systems. [8,15] describes practical but incomplete constructions for general programs based on transition predicate abstraction. [15] states an example which has a transition invariant but no lexicographic ranking function over linear expressions; it is interesting to better understand the connection between the different termination proof techniques and investigate under which conditions ranking functions can be constructed.

Our results on transition abstraction and the previous results on size-change abstraction heavily rely on automata-theoretic techniques. We speculate that the study of the automaton structure of other inductive proofs, such as *cyclic proofs* [27], might also yield interesting results.

Acknowledgements. This article is dedicated to the memory of Helmut Veith who proposed to me the PhD topic of automatic derivation of loop bounds. Our initial idea was to extend the termination analysis of Terminator. With this article I managed to return to this original idea.

References

1. Albert, E., Arenas, P., Genaim, S., Puebla, G., Zanardini, D.: Cost analysis of object-oriented bytecode programs. Theor. Comput. Sci. **413**(1), 142–159 (2012)
2. Alias, C., Darte, A., Feautrier, P., Gonnord, L.: Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs. In: Cousot, R., Martel, M. (eds.) SAS 2010. LNCS, vol. 6337, pp. 117–133. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15769-1_8
3. Anderson, H., Khoo, S.-C.: Affine-based size-change termination. In: Ohori, A. (ed.) APLAS 2003. LNCS, vol. 2895, pp. 122–140. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-40018-9_9
4. Ben-Amram, A.M.: Size-change termination with difference constraints. ACM Trans. Program. Lang. Syst. **30**(3), 1–30 (2008)
5. Ben-Amram, A.M.: Monotonicity constraints for termination in the integer domain. Log. Methods Comput. Sci. **7**(3), 1–43 (2011)
6. Blass, A., Gurevich, Y.: Program termination and well partial orderings. ACM Trans. Comput. Log. **9**(3), 18:1–18:26 (2008)
7. Bozzelli, L., Pinchinat, S.: Verification of gap-order constraint abstractions of counter systems. In: VMCAI, pp. 88–103 (2012)
8. Brockschmidt, M., Cook, B., Fuhs, C.: Better termination proving through cooperation. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 413–429. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_28
9. Codish, M., Fuhs, C., Giesl, J., Schneider-Kamp, P.: Lazy abstraction for size-change termination. In: Fermüller, C.G., Voronkov, A. (eds.) LPAR 2010. LNCS, vol. 6397, pp. 217–232. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16242-8_16
10. Codish, M., Gonopolskiy, I., Ben-Amram, A.M., Fuhs, C., Giesl, J.: Sat-based termination analysis using monotonicity constraints over the integers. TPLP **11**(4–5), 503–520 (2011)
11. Colcombet, T.: Factorisation forests for infinite words. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) FCT 2007. LNCS, vol. 4639, pp. 226–237. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74240-1_20
12. Colcombet, T., Daviaud, L., Zuleger, F.: Size-change abstraction and max-plus automata. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014. LNCS, vol. 8634, pp. 208–219. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44522-8_18
13. Colcombet, T., Daviaud, L., Zuleger, F.: Automata and program analysis. In: Klasing, R., Zeitoun, M. (eds.) FCT 2017. LNCS, vol. 10472, pp. 3–10. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-55751-8_1
14. Cook, B., Podelski, A., Rybalchenko, A.: Termination proofs for systems code. In: PLDI, pp. 415–426 (2006)

15. Cook, B., See, A., Zuleger, F.: Ramsey vs. lexicographic termination proving. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 47–61. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36742-7_4
16. Flores-Montoya, A., Hähnle, R.: Resource analysis of complex programs with cost equations. In: Garrigue, J. (ed.) APLAS 2014. LNCS, vol. 8858, pp. 275–295. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12736-1_15
17. Giesl, J., Aschermann, C., Brockschmidt, M., Emmes, F., Frohn, F., Fuhs, C., Hensel, J., Otto, C., Plücker, M., Schneider-Kamp, P., Ströder, T., Swiderski, S., Thiemann, R.: Analyzing program termination and complexity automatically with aprove. J. Autom. Reason. **58**(1), 3–31 (2017)
18. Gulwani, S., Zuleger, F.: The reachability-bound problem. In: PLDI, pp. 292–304 (2010)
19. Heizmann, M., Jones, N.D., Podelski, A.: Size-change termination and transition invariants. In: Cousot, R., Martel, M. (eds.) SAS 2010. LNCS, vol. 6337, pp. 22–50. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15769-1_4
20. Krauss, A.: Certified size-change termination. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 460–475. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73595-3_34
21. Lee, C.S.: Ranking functions for size-change termination. ACM Trans. Program. Lang. Syst. **31**(3), 10:1–10:42 (2009)
22. Lee, C.S., Jones, N.D., Ben-Amram, A.M.: The size-change principle for program termination. In: POPL, pp. 81–92 (2001)
23. Manolios, P., Vroon, D.: Termination analysis with calling context graphs. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 401–414. Springer, Heidelberg (2006). https://doi.org/10.1007/11817963_36
24. Miné, A.: The octagon abstract domain. High. Order Symb. Comput. **19**(1), 31–100 (2006)
25. Podelski, A., Rybalchenko, A.: Transition invariants. In: LICS, pp. 32–41 (2004)
26. Podelski, A., Rybalchenko, A.: Transition predicate abstraction and fair termination. ACM Trans. Program. Lang. Syst. **29**(3), 15 (2007)
27. Rowe, R.N.S., Brotherston, J.: Automatic cyclic termination proofs for recursive procedures in separation logic. In: CPP, pp. 53–65 (2017)
28. Simon, I.: Factorization forests of finite height. Theor. Comput. Sci. **72**(1), 65–94 (1990)
29. Sinn, M., Zuleger, F., Veith, H.: Complexity and resource bound analysis of imperative programs using difference constraints. J. Autom. Reason. **59**(1), 3–45 (2017)
30. Steila, S., Yokoyama, K.: Reverse mathematical bounds for the termination theorem. Ann. Pure Appl. Logic **167**(12), 1213–1241 (2016)
31. Vidal, G.: Quasi-terminating logic programs for ensuring the termination of partial evaluation. In: PEPM, pp. 51–60 (2007)
32. Vytiniotis, D., Coquand, T., Wahlstedt, D.: Stop when you are almost-full: adventures in constructive termination. In: Beringer, L., Felty, A. (eds.) ITP 2012. LNCS, vol. 7406, pp. 250–265. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32347-8_17

33. Zuleger, F.: Asymptotically precise ranking functions for deterministic size-change systems. In: Beklemishev, L.D., Musatov, D.V. (eds.) CSR 2015. LNCS, vol. 9139, pp. 426–442. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20297-6_27
34. Zuleger, F., Gulwani, S., Sinn, M., Veith, H.: Bound analysis of imperative programs with the size-change abstraction. In: Yahav, E. (ed.) SAS 2011. LNCS, vol. 6887, pp. 280–297. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23702-7_22