Size-Change Termination and Transition Invariants

Matthias Heizmann[1], Neil D. Jones[2], and Andreas Podelski[1]

[1] University of Freiburg, Germany
[2] University of Copenhagen, Denmark

Abstract. Two directions of recent work on program termination use the concepts of size-change termination resp. transition invariants. The difference in the setting has as consequence the inherent incomparability of the analysis and verification methods that result from this work. Yet, in order to facilitate the crossover of ideas and techniques in further developments, it seems interesting to identify which aspects in the respective formal foundation are related. This paper presents initial results in this direction.

1 Introduction

There have been rapid advances in methods for automatically proving program termination in recent years, both in theoretical research and in applications as practical as finding termination bugs in device drivers. A recent wave of activity began with the work on *size-change termination* from [22]. Related work and further developments include, e.g., [7, 20, 22, 28]). A branch of this work is based on the concept of *transition invariants* from [26]; see, e.g., [11, 13, 14, 17, 21, 27]). The motivation behind the work in [26] was to carry over the ideas of [22] to verification methods in the style of *software model checking* [3, 4]. This goal entailed going from a *decidable* program analysis problem (for functional programs) to an *undecidable* verification problem (for imperative and concurrent programs). The change of setting has as consequence the inherent *incomparability* of the methods that result from the work on size-change termination resp. transition invariants. Yet, in order to facilitate the crossover of ideas and techniques in further developments of such methods, it seems interesting to identify which aspects in the respective formal foundation are related. This paper presents three initial results. They concern 1. the proof rule, 2. the abstract domain, and 3. the base algorithm.

1. If we take the proof rule that implicitly underlies the soundness proof for the size-change termination analysis in [22] and the transition invariant-based proof rule from [26], then the premise of the former is strictly stronger than the premise of the latter and the conclusion of the former is strictly stronger than the conclusion of the latter.

   In detail: The size-change termination analysis in [22] decides *size-change termination*, a property strictly stronger than termination. The intermediate result of the analysis is a set of *size-change graphs*. The analysis gives a yes-answer if *some* of the graphs (the idempotent ones) denote a well-founded relation. But then, perhaps surprisingly, *all* of the graphs must denote a well-founded relation. This means that the premise in the (complete) proof rule for termination from [26] is satisfied.

2. The abstract domain of *size-change graphs* in [22] corresponds to a specific parameter for the *transition predicate abstraction* used in [11, 13, 14, 17, 27].
   In detail: we can fix a specific set of *transition predicates* such that each size-change graph can be translated to an equivalent conjunction of transition predicates in this set, and vice versa. In fact, the arcs correspond to the conjuncts.
3. When we categorize the base algorithm in the termination analyses by the decision problem that it solves, we can establish the formal connection between the base algorithms.
   In detail: We define two decision problems, one for size-change termination and one for transition invariants; let us call them SCT and TI, for short. Then SCT belongs to a special case of a third decision problem which, in the special case, can be formulated in terms of TI (in general, the third decision problem has a strictly higher complexity than TI).

The special case of the third decision problem (in Point 3.) is defined by the associativity of the *abstract composition* of relations. The associativity is responsible not only for the lower complexity but also for the already (in Point 1.) mentioned feature of size-change termination analysis. I.e., among the elements in the output of the base algorithm, only the subset of *idempotent* elements has to be inspected for well-foundedness (if the binary operation over the elements is associative).

The definition of the special case thus abstracts away from the graph representation and helps us to identify the associativity of their composition as the crucial property of size-change graphs.[3]

The question left open by this paper is whether the notions of associativity and idempotency have correspondent notions for a similar optimization in transition invariant-based termination analyses.

Roadmap. The first part of this paper presents what we believe is the essence of size-change termination (Section 2) and transition invariants and transition predicate abstraction (Section 3). Points 1. and 2. from above are covered in Section 4. The reader who in interested only in Point 3. can jump directly to Section 5, which we tried to keep self-contained. The paper ends with a discussion of the qualitative differences that result from the different settings of the methods. An extended version of this paper is available online [18].

[3] The associativity of the composition is lost in the extension of size-change graphs with finitely many arc weights in the style of [5] (where, for example, the weighted arc $\mathtt{x} \xrightarrow{k} \mathtt{x}$ means that the value of $\mathtt{x}$ decreases by at least the integer k).

2 Size-change termination (SCT)

2.1 A running example

Example 1. Figure 1 is a program example similar to one in [20]. It is a first-order tail-recursive functional program with three function calls labeled `1`, `2` and `3`. Argument values range over the natural numbers $I\!N$, ordered as usual.

Figure 2 contains the program's "control flow graph" with the calling function and called function of each call, e.g., $\mathtt{1} : \mathtt{f} \to \mathtt{g}$. It also associates with each call τ a "size-change graph", e.g., G_τ. Example: G_1 abstracts the tuple of data flow size changes that occur in call `1` from `f` to `g`. Symbol $\downarrow$ in G_1, G_2, G_3 indicates a value decrease, and symbol $\overset{=}{\downarrow}$ indicates a decrease or equality.

```
f(x,y)   = if x=0 then y else 1: g(x,y,y)
g(u,v,w) = if v>0 then 2: g(u,v-1,2*w) else 3: f(u-1,w)
```

Fig. 1. Example of a first order tail-recursive functional program.

Informal SCT termination reasoning for the running example. Suppose (hypothetically) there is *an infinite call sequence* $\pi = \tau_1\tau_2\tau_3\ldots$ that follows program P's control flow. We argue that any computation following π would have an infinitely descending sequence of variable values. But this would contradict the well-foundedness of set $I\!N$. Conclusion: program P terminates.

Case 1: $\pi = \ldots 2^\omega$ ends in infinitely many 2's. By safety of graph G_2, this implies that the values of *variable* `v` *descend infinitely.*

Case 2: Since π is infinite, the only other possibility is that it has the form $\pi = \ldots (12^*3)^\omega$. Again by safety, this implies that the values of *variable* `u` *descend infinitely* (once each time loop 12*3 is traversed).
Therefore a call of *any program function with any data will terminate.*

Paper [22] shows two different approaches to make such reasoning algorithmic: One is based on Büchi automata, and the other computes the *closure* of the given set of graphs as follows (Section 1.2 of [22], and Section 2.4 below).

2.2 Some size-change definitions

Program semantics: [22] is about first-order functional programs, and contains both syntax and a denotational (big-step) call-by-value semantics. Given a set *Value* containing values of expressions, the semantic function has type

$$\mathcal{E}[\![\_]\!] : \mathit{Expression} \to (\mathit{Value}^n \to \mathit{Value} \cup \{\bot\})$$

If `e` is an expression, then $\mathcal{E}[\![\mathtt{e}]\!]\boldsymbol{v}$ is the value of expression `e`, given an an environment $\boldsymbol{v}$ containing values of the variables occurring in `e`. We omit the completely standard definition of $\mathcal{E}[\![\_]\!]$, see [22] or a textbook on semantics for details.

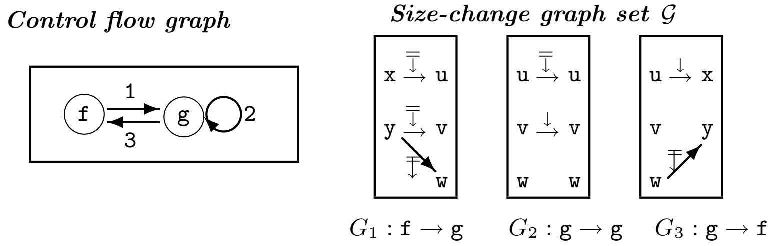

Control flow graph
f
g
1
2
3
Size-change graph set $\mathcal{G}$
$\mathtt{x} \xrightarrow{\overset{=}{\downarrow}} \mathtt{u}$
$\mathtt{y} \xrightarrow{\overset{=}{\downarrow}} \mathtt{v}$
$\overset{=}{\downarrow}$
w
$\mathtt{u} \xrightarrow{\overset{=}{\downarrow}} \mathtt{u}$
$\mathtt{v} \xrightarrow{\downarrow} \mathtt{v}$
w w
$\mathtt{u} \xrightarrow{\downarrow} \mathtt{x}$
v y
$\overset{=}{\downarrow}$
w
$G_1 : \mathtt{f} \to \mathtt{g}$
$G_2 : \mathtt{g} \to \mathtt{g}$
$G_3 : \mathtt{g} \to \mathtt{f}$

Fig. 2. Size-change graphs for the running example

Size changes: we assume given a well-founded order $>$ on *Value*.

Definition 2. *Suppose functions* f, g *are defined in* P. *A* size-change graph $G : \mathtt{f} \to \mathtt{g}$ *for* P *is a set of labeled arcs* $x \xrightarrow{r} y$ *where* $r \in \{\overset{=}{\downarrow}, \downarrow\}$, $x \in \mathsf{V}\mathit{ariables}(\mathtt{f})$, $y \in \mathsf{V}\mathit{ariables}(\mathtt{g})$, *and* G *does not contain both* $x \xrightarrow{\overset{=}{\downarrow}} y$ *and* $x \xrightarrow{\downarrow} y$ *for any* x, y.

Functions f and g are respectively called the *source* and the *target* of G. We will sometimes elide f and g, writing G rather than $G : \mathtt{f} \to \mathtt{g}$.

Definition 3. *Let* $\mathcal{G} = \{G_\tau \mid \tau \text{ is a call in } P\}$ *be a set of size-change graphs for program* P.

1. *Suppose the definition of* f *contains a call to* g *labeled* τ:

$$\mathtt{f}(\mathtt{x}_1, \ldots, \mathtt{x}_m) = \ldots \tau : \mathtt{g}(\mathtt{e}_1, \ldots, \mathtt{e}_n) \ldots$$

   The phrase "arc $\mathtt{f}^{(i)} \xrightarrow{r} \mathtt{g}^{(j)}$ *safely describes the* $\mathtt{f}^{(i)}$*-*$\mathtt{g}^{(j)}$ *size relation in call* τ*" means: For every* $v \in$ *Value and* $\boldsymbol{v} = (v_1, \ldots, v_m)$, *if* $\mathcal{E}[\![\mathtt{e}_j]\!]\boldsymbol{v} = v$ *is defined, then*

$$r = \downarrow \text{ implies } v_i > v \text{ ; and } r = \overset{=}{\downarrow} \text{ implies } v_i \geq v$$

2. *Size-change graph* G_τ *is* safe for call $\tau : \mathtt{f} \to \mathtt{g}$ *if every arc in* G_τ *is a safe description as just defined.*
3. *Set* $\mathcal{G}$ *of size-change graphs is a* safe description of program P *if graph* G_τ *is safe for every call* τ.

Assuming values are natural numbers, it is easy to see that all the size-change graphs shown in Example 1 are safe for their respective calls. No size relation in $\{\overset{=}{\downarrow}, \downarrow\}$ can be safely asserted about argument w of call 2, since 2*w may exceed the current value of w. According to Definition 3, G_2 safely models the parameter size-changes caused by call 2.

Definition 4. *A* multipath $\mathcal{M}$ *is a graph sequence* $G_1, G_2, G_3, \ldots$ *such that* $\text{target}(G_i) = \text{source}(G_{i+1})$ *for* $i = 1, 2, \ldots$ *A* thread *is a connected path of arcs in* $\mathcal{M}$ *that starts at some* G_t, $t \geq 1$: $th = z_{i_t} \xrightarrow{r_t} z_{i_{t+1}} \xrightarrow{r_{t+1}} z_{i_{t+2}} \xrightarrow{r_{t+2}} \ldots$ *with each* $r_{t+j} \in \{\overset{=}{\downarrow}, \downarrow\}$. *The thread has* infinite descent *if it contains infinitely many* $\downarrow$*'s.*

For example, G_2, G_3, G_1 is a multipath in Figure 2. It contains one thread with 3 arcs, namely $\mathtt{u} \xrightarrow{\stackrel{=}{\downarrow}} \mathtt{u} \xrightarrow{\downarrow} \mathtt{x} \xrightarrow{\stackrel{=}{\downarrow}} \mathtt{u}$.

Definition 5 (Size-change terminating program). *(Section 1.2 of [22]) Let $\mathcal{T}$ be the set of calls in program P. Suppose each size-change graph $G_\tau : \mathtt{f} \to \mathtt{g}$ is safe for every call τ in*

$$\mathcal{G} = \{G_\tau \mid \tau \in \mathcal{T}\}$$

Define P to be size-change terminating *if, for any infinite call sequence $\pi = \tau_1\tau_2\tau_3\ldots$ that follows P's control flow, there is a thread of infinite descent in the multipath $\mathcal{M}_\pi = G_{\tau_1}, G_{\tau_2}, G_{\tau_3}, \ldots$.*

2.3 Composition of size-change graphs

Definition 6. *The* composition *of two size-change graphs $G : \mathtt{f} \to \mathtt{g}$ and $G' : \mathtt{g} \to \mathtt{h}$ is $G; G' : \mathtt{f} \to \mathtt{h}$ with arc set E defined below. Notation: write $\mathtt{x} \xrightarrow{r} \mathtt{y} \xrightarrow{r'} \mathtt{z}$ if $\mathtt{x} \xrightarrow{r} \mathtt{y}$ and $\mathtt{y} \xrightarrow{r'} \mathtt{z}$ are respectively arcs of G and G'.*

$$\begin{aligned} E = &\{\mathtt{x} \xrightarrow{\downarrow} \mathtt{z} \mid \exists \mathtt{y}, r \ .\ \mathtt{x} \xrightarrow{\downarrow} \mathtt{y} \xrightarrow{r} \mathtt{z} \textit{ or } \mathtt{x} \xrightarrow{r} \mathtt{y} \xrightarrow{\downarrow} \mathtt{z}\} \\ &\bigcup \{\mathtt{x} \xrightarrow{\stackrel{=}{\downarrow}} \mathtt{z} \mid (\exists \mathtt{y} \ .\ \mathtt{x} \xrightarrow{\stackrel{=}{\downarrow}} \mathtt{y} \xrightarrow{\stackrel{=}{\downarrow}} \mathtt{z}) \textit{ and } (\forall \mathtt{y}, r, r' \ .\ \mathtt{x} \xrightarrow{r} \mathtt{y} \xrightarrow{r'} \mathtt{z} \textit{ implies } r = r' = \stackrel{=}{\downarrow})\} \end{aligned}$$

Further, we define:

- *Size-change graph G is* idempotent *if $G; G = G$.*
- *$G_\pi = G_{\tau_1}; \ldots; G_{\tau_n}$ for any finite call sequence $\pi = \tau_1 \ldots \tau_n \in \mathcal{T}^*$.*

Lemma 7. *The composition operator "$;$" is associative.*

2.4 A closure algorithm to decide the SCT property

Definition 8. *The* closure *of a set $\mathcal{G}$ of size-change graphs is the smallest set $cl(\mathcal{G})$ such that*

- *$\mathcal{G} \subseteq cl(\mathcal{G})$*
- *If $G_1 : \mathtt{f} \to \mathtt{f}'$ and $G_2 : \mathtt{f}' \to \mathtt{f}''$ are in $cl(\mathcal{G})$, then $G_1; G_2 \in cl(\mathcal{G})$.*

In the worst case, $cl(\mathcal{G})$ can be exponentially larger than $\mathcal{G}$, see [22].

Example 9. Suppose $\mathcal{G} = \{G_1, G_2, G_3\}$ as in Example 1. Its closure is

$$cl(\mathcal{G}) = \{G_1, G_2, G_3, G_{12}, G_{123}, G_{1231}, G_{13}, G_{131}, G_{1312}, G_{23}, G_{231}, G_{31}\}$$

Each graph in $cl(\mathcal{G})$ is the composition G_π for a finite P call sequence π, e.g.,

$$G_{231} = G_2; G_3; G_1$$

for $\pi = 231$ has $\mathtt{f}$ as both source and target, and contains one arc: $\mathtt{u} \xrightarrow{\downarrow} \mathtt{u}$.

Theorem 10. *Program P is SCT terminating iff every idempotent G in $cl(\mathcal{G})$ has an arc $z \overset{\downarrow}{\rightarrow} z$.*

Proof. This is Theorem 4 from [22]. For "only if" ($\Rightarrow$), suppose P is size-change terminating and that G_π in $cl(\mathcal{G})$ is idempotent: $G_\pi = G_\pi; G_\pi$. By Definition 5, the infinite call sequence $\pi^\omega = \pi, \ldots, \pi, \pi, \ldots$ has an infinitely descending thread. Consider this thread's position at the start of each π in π^ω. There are finitely many variables, so the thread must visit some variable $\mathtt{x}$ infinitely often. Thus there must be $n, \mathtt{x}$ such that π^n has a thread from $\mathtt{x}$ to $\mathtt{x}$ containing $\mathtt{x} \overset{\downarrow}{\rightarrow} \mathtt{x}$. By Definition 6, arc $\mathtt{x} \overset{\downarrow}{\rightarrow} \mathtt{x}$ is in G_{π^n}. Idempotence of G_π implies $G_{\pi^n} = G_\pi; G_\pi; \ldots; G_\pi = G_\pi$, so $\mathtt{x} \overset{\downarrow}{\rightarrow} \mathtt{x}$ is in G_π.

"If" ($\Leftarrow$): we show that if P is *not* size-change terminating, there exists an idempotent $G \in cl(\mathcal{G})$ without an arc $z \overset{\downarrow}{\rightarrow} z$. Assuming P is not size-change terminating, by Definition 5 there is an infinite call sequence $\pi = \tau_1 \tau_2 \ldots$ such that multipath $\mathcal{M}_\pi = G_{\tau_1}, G_{\tau_2}, \ldots$ has no infinitely descending thread. Define

$$h(k, \ell) = G_{\tau_k}; \ldots; G_{\tau_{\ell-1}}$$

for $k, \ell \in \mathbb{N}$ with $0 < k < \ell$. Define equivalence relation $\simeq$ on h's domain by

$$(k, \ell) \simeq (k', \ell') \text{ if and only if } h(k, \ell) = h(k', \ell')$$

Relation $\simeq$ is of finite index since the closure set $cl(\mathcal{G})$ is finite. By Ramsey's Theorem there exists an infinite set $K \subseteq \mathbb{N}$ and fixed $m, n \in \mathbb{N}$ such that $(k, \ell) \simeq (m, n)$ for any $k, \ell \in K$ with $k < \ell$. Expanding the definition of $\simeq$ gives

$$G_{\tau_k}; \ldots; G_{\tau_{\ell-1}} = G_{\tau_m}; \ldots; G_{\tau_{n-1}}$$

Let $G^\circ = h(m, n)$. By associativity of ;, $p, q, r \in K$, with $p < q < r$ implies

$$\begin{aligned} G^\circ &= G_{\tau_p}; \ldots; G_{\tau_{r-1}} \\ &= (G_{\tau_p}; \ldots; G_{\tau_{q-1}}); (G_{\tau_q}; \ldots; G_{\tau_{r-1}}) \\ &= G^\circ; G^\circ \end{aligned}$$

so G° is idempotent (and so $G^\circ : \mathtt{f} \rightarrow \mathtt{f}$ for some $\mathtt{f}$).

If G° had an arc $z \overset{\downarrow}{\rightarrow} z$, then the multipath $G_{\tau_m}, \ldots, G_{\tau_{n-1}}$ would have a descending thread from z to z. This would imply $\mathcal{M}_\pi$ has an infinite descending thread, violating the assumption about π. □

Theorem 11. *The problem of deciding SCT termination is in* PSPACE *(as a function of program size).*

Proof. (Sketch) First, we argue that SCT termination is a path property in a certain graph. Since the composition operator " ; " is associative, a graph G is in $cl(\mathcal{G})$ iff $G = G_\pi$ for some π. Thus by Theorem 10

> P is size-change terminating iff there exists no call sequence π such that G_π is idempotent and G_π contains no arc $z \overset{\downarrow}{\rightarrow} z$.

This is a reachability problem in a directed graph (call it Γ). Each node of Γ is a size-change graph G, and each arc is from G_π to $G_{\pi\,\tau}$ where $\pi \in \mathcal{T}^*, \tau \in \mathcal{T}$. The number of nodes in Γ is the number of possible size-change graphs G for program P.

A well-known result by Savitch is that existence of a path in a directed graph with m nodes can be decided[4] in space $O(\log^2 m)$. (See [19] for the "divide-and-conquer" proof.) The number of size-change graphs is bounded by 3^{p^2} where p is the number of variables in P. (Reasoning: between any two variables there may be no arc, or one arc labeled by $\downarrow$, or one labeled by $\overset{=}{\downarrow}$.) Thus the graph may, by Savitch's result, be searched using memory space $O(\log^2(3^{p^2}))$. This is clearly bounded by a polynomial in the number of variables of program P. $\square$

[22] shows PSPACE to be a lower bound, so the problem is PSPACE-complete.

3 Transition invariants (TI)

3.1 Programs defined by transitions

Following [26, 23], in order to abstract away from the syntax of imperative programs we use transitions to formalize programs. A transition τ can be thought of as a label of a statement.

Definition 12 (Transition-based program). *We define a* program *to be a triple*

$$P = (\Sigma, \mathcal{T}, \rho),$$

consisting of:

- *a set of* states Σ,
- *a finite set of* transitions $\mathcal{T}$*, and*
- *a function ρ that assigns to each transition a binary relation over states,*

$$\rho_\tau \subseteq \Sigma \times \Sigma, \qquad \text{for } \tau \in \mathcal{T}.$$

The transition relation of P*, denoted R_P, comprises the transition relations ρ_τ of all transitions $\tau \in \mathcal{T}$, i.e.,*

$$R_P = \bigcup_{\tau \in \mathcal{T}} \rho_\tau.$$

A program P is *terminating* if its transition relation R_P is well-founded. This means there is no infinite computation

$$s_1 \xrightarrow{\tau_1} s_2 \xrightarrow{\tau_2} s_3 \xrightarrow{\tau_3} \dots$$

i.e., there is no sequence of states $s_1, s_2, \dots$ and transitions $\tau_1, \tau_2, \dots$ such that for every $i \in \mathbb{N}$, the state pair (s_i, s_{i+1}) is contained in the transition relation ρ_{τ_i}.

[4] A key point is that the entire graph Γ is not held in storage at any time, but just the nodes currently being investigated. See [19] for the "divide-and-conquer" proof.

States. Imperative programs in most references use a more concrete version of states:

$$\Sigma = \mathsf{Loc} \times (\mathsf{Var} \rightarrow \mathsf{Value})$$

where $\mathsf{Loc}, \mathsf{Var}$ are finite sets (of *locations* and *variables*), and Value is a perhaps infinite set of *values*. Typical elements (perhaps decorated) are: $\ell \in \mathsf{Loc}, z \in \mathsf{Var}$ and $v \in \mathsf{Value}$. A state in Σ has form $s = (\ell, \sigma)$ where $\sigma : \mathsf{Var} \rightarrow \mathsf{Value}$.

We only deal with one program at a time, so the objects $P, \mathsf{Loc}, \mathsf{Var}, \mathsf{Value}$ will have fixed values. Letting $\mathsf{Var} = \{z_1, \ldots, z_m\}$, a state can be written as $s = (\ell, \sigma)$ or

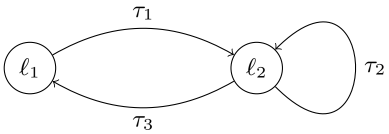
$$s = (\ell, v_1, \ldots, v_m)$$

Here ℓ is the current location; $v_1, \ldots, v_m \in \mathsf{Value}$ are the respective values of variables $z_1, \ldots, z_m$.

Extensional versus intensional representation. Program P's semantics is by Definition 12 its transition relation $R_P \subseteq \Sigma \times \Sigma$, that is, a set of pairs of states (s, s'), where s is the "current state" and s' is the "next state". Natural operations on such sets include boolean operations $\cup, \cap, \setminus$ and set-formers. This corresponds to an *extensional* view of semantics.

For practical uses (e.g., in a theorem prover) many writers use as alternative an *intensional* view of semantics, and represent a set of state-pairs, i.e., a transition relation, by a logic formula with implicit universal quantification over free logical variables. The universe of discourse (for a given, fixed, program P): a formula will *always* denote a subset of $\Sigma \times \Sigma$. Formulas are built using logical operations $\vee, \wedge, \Rightarrow, \neg$. These correspond exactly to $\cup, \cap, \subseteq$ and $\Sigma \setminus \_$.

We follow the usual convention of naming values in s by unprimed logical variables, and values in s' by primed logical variables. Logical variables are program counters pc, pc' and the variables of program P. Locations: the atomic formula $pc = \ell$ means that the control location of s is ℓ; and $pc' = \ell'$ means that the control location of s' is ℓ'. Formula variables other than pc, pc' are program variables ranging over Value. Formulas can represent state pair sets compactly, since variables not occurring in a formula are simply not constrained (they range over all of Value or Loc).



$$\begin{array}{lllllll}
\rho_{\tau_1} & \text{is} & pc = \ell_1, pc' = \ell_2,\ x \neq 0,\ x' = x, & y' = y,\ u' = x,\ v' = y, & w' = y \\
\rho_{\tau_2} & \text{is} & pc = \ell_2, pc' = \ell_2,\ v > 0,\ x' = x, & y' = y,\ u' = u,\ v' = v - 1, & w' = 2 * w \\
\rho_{\tau_3} & \text{is} & pc = \ell_2, pc' = \ell_1,\ v = 0,\ x' = u - 1, & y' = w,\ u' = u,\ v' = v, & w' = w
\end{array}$$

Fig. 3. Transition relation and corresponding control flow graph of a program $P = (\Sigma, \mathcal{T}, \rho)$ where the set of states Σ is $\{\ell_1, \ell_2\} \times I\!N^5$, the set of transitions $\mathcal{T}$ is $\{\tau_1, \tau_2, \tau_3\}$ and the program's transition relation R_P is $\rho_{\tau_1} \vee \rho_{\tau_2} \vee \rho_{\tau_3}$

Example 13. Figure 3 expresses a transition-based program in the sense of Definition 12, using an intensional representation, and comma to abbreviate conjunction $\wedge$. As we will see in Section 4, the program stems from translating the functional program from **Example 1**.

3.2 Termination by transition invariants

In this section we give a brief description of terminology and results of [26] restricted to termination ([26] also deals with general liveness properties and fairness). We write r^+ to denote the transitive closure of a relation r.

Definition 14 (Transition invariant). *Given a program $P = (\Sigma, \mathcal{T}, \rho)$, a transition invariant T is a binary relation over states T that contains the transitive closure R_P^+ of the program's transition relation R_P, i.e.,*

$$R_P^+ \subseteq T.$$

Definition 15 (Disjunctively well-founded relation). *A relation T is* disjunctively well-founded *if it is a finite union of well-founded relations:*

$$T = T_1 \cup \cdots \cup T_n$$

Theorem 16 (Proof rule for termination). *A program P is terminating if and only if there exists a disjunctively well-founded transition invariant for P.*

Using the above proof rule, we can prove termination of a program P as follows.

1. Find a finite number of relations $T_1, \ldots, T_n$.
2. Show that the inclusion $R_P^+ \subseteq T_1 \cup \cdots \cup T_n$ holds.
3. Show that each relation $T_1, \ldots, T_n$ is well-founded.

Proof. This is Theorem 1 from [26]. "Only if" ($\Rightarrow$) is trivial: if P is terminating, then both R_P and R_P^+ are well-founded. Choose $n = 1$ and $T_1 = R_P^+$.

"If" ($\Leftarrow$): we show that if P is *not* terminating and $T_1 \cup \cdots \cup T_n$ is a transition invariant, then some T_i is not well-founded. Nontermination of P means there exists an infinite computation:

$$s_0 \overset{\tau_1}{\rightarrow} s_1 \overset{\tau_2}{\rightarrow} s_3 \overset{\tau_3}{\rightarrow} \ldots$$

Let choice function f satisfy

$$f(k, \ell) \in \{\ T_i \mid (s_k, s_\ell) \in T_i\ \}$$

for $k, \ell \in I\!N$ with $k < \ell$. (The condition $R_P^+ \subseteq T_1 \cup \cdots \cup T_n$ implies that f exists, but does not define it uniquely.) Define equivalence relation $\simeq$ on f's domain by

$$(k, \ell) \simeq (k', \ell') \text{ if and only if } f(k, \ell) = f(k', \ell')$$

Relation $\simeq$ is of finite index since the set of T's is finite. By Ramsey's Theorem there exists an infinite sequence of natural numbers $k_1 < k_2 < \ldots$ and fixed $m, n \in I\!N$ such that

$$(k_i, k_{i+1}) \simeq (m, n) \qquad\qquad \text{for all } i \in I\!N.$$

Hence $(s_{k_i}, s_{k_{i+1}}) \in T_{mn}$ for all i. This is a contradiction: T_{mn} is not well-founded. □

In comparison to Theorem 10 the proof of Theorem 16 uses a weaker version of Ramsey's theorem. The weak version of Ramsey's theorem states that every infinite complete graph that is colored with finitely many colors contains a monochrome infinite path.

Example 17. Consider the program P in Figure 3 and the binary relations $T_1, \ldots, T_5$ given by the five formulas below.

$$\begin{array}{ll}
T_1: & pc = \mathtt{f} \ \wedge\ pc' = \mathtt{f} \ \wedge\ x > x', \\
T_2: & pc = \mathtt{g} \ \wedge\ pc' = \mathtt{g} \ \wedge\ v > v', \\
T_3: & pc = \mathtt{g} \ \wedge\ pc' = \mathtt{g} \ \wedge\ u > u', \\
T_4: & pc = \mathtt{f} \ \wedge\ pc' = \mathtt{g}, \\
T_5: & pc = \mathtt{g} \ \wedge\ pc' = \mathtt{f},
\end{array}$$

The union of these relation is a transition invariant for P, i.e., the inclusion $R_P^+ \subseteq T_1 \cup \cdots \cup T_5$ holds. Since every T_i is well-founded, their union is a disjunctively well-founded transition invariant and hence, by Theorem 16 the program P is terminating.

3.3 Transition predicate abstraction (TPA)

Transition predicate abstraction [27] is a method to compute transition invariants, just as predicate abstraction is a method to compute invariants. The method takes as input a selection of finitely many binary relation over states. We call these relations *transition predicates.* For this section we fix a finite set of transition predicates $\mathcal{P}$. We usually refer to a transition predicate by the formula that defines it.

Definition 18 (Set of abstract transitions $\mathcal{T}_{\mathcal{P}}^{\#}$). *Given the set of transition predicates $\mathcal{P}$, the set of* abstract transitions *$\mathcal{T}_{\mathcal{P}}^{\#}$ is the set that contains the conjunction of every subset of transition predicates $\{p_1, \ldots p_m\} \subseteq \mathcal{P}$, i.e.,*

$$\mathcal{T}_{\mathcal{P}}^{\#} = \{p_1 \wedge \ldots \wedge p_m \mid p_i \in \mathcal{P},\ 0 \leq m, 1 \leq i \leq m\}$$

Clearly $\mathcal{T}_{\mathcal{P}}^{\#}$ is closed under intersection, and the set of all binary relations over states $\Sigma \times \Sigma$ is a member of $\mathcal{T}_{\mathcal{P}}^{\#}$ (the case $m = 0$).

Example 19. Consider the following set of transition predicates.

$$\mathcal{P} = \{x = x', x > x', y > y'\}$$

The set of abstract transitions $\mathcal{T}^{\#}_{\mathcal{P}}$ is

$$\{\mathsf{true},\ \ x = x',\ \ \ x > x',\ \ \ y > y',\ \ \ x = x' \wedge y > y',\ \ \ x > x' \wedge y > y',\ \ \mathsf{false}\}$$

The abstract transition written as true is the set of all state pairs $\Sigma \times \Sigma$ and is the empty conjunction of transition predicates. The abstract transition false is the empty relation; e.g., the conjunction of $x = x'$ and $x > x'$ is false.

We next define a function that assigns to a binary relation T over states the least (wrt. inclusion) abstract transition that is a superset of T.

Definition 20 (Abstraction function α). *A set of transition predicates $\mathcal{P}$ defines the* abstraction function

$$\alpha : 2^{\Sigma \times \Sigma} \to \mathcal{T}^{\#}_{\mathcal{P}}$$

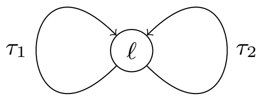
that assigns to a relation $r \subseteq \Sigma \times \Sigma$ the smallest abstract transition that is a superset of r, i.e.,

$$\alpha(r) = \bigwedge\{p \in \mathcal{P} \mid r \subseteq p\}.$$

We note that α is extensive, i.e., the inclusion

$$r \subseteq \alpha(r)$$

holds for any binary relation over states $r \subseteq \Sigma \times \Sigma$.



ρ_{τ_1} is $pc = \ell \ \wedge\ pc' = \ell \ \wedge\ x' = x - 1$
ρ_{τ_2} is $pc = \ell \ \wedge\ pc' = \ell \ \wedge\ x' = x \ \wedge\ y' = y - 1$

Fig. 4. Transition relation and corresponding control flow graph of a program $P = (\Sigma, \mathcal{T}, \rho)$ where the set of states Σ is $\{\ell\} \times \mathbb{N} \times \mathbb{N}$, the set of transitions $\mathcal{T}$ is $\{\tau_1, \tau_2\}$ and the program's transition relation R_P is $\rho_{\tau_1} \vee \rho_{\tau_2}$.

Example 21. Application of the abstraction function α to the transition relations ρ_1 and ρ_2 of the program in Figure 4 results in the following abstract transitions.

$$\begin{array}{lll} \alpha(\rho_{\tau_1}) & \text{is} & x > x' \\ \alpha(\rho_{\tau_2}) & \text{is} & x = x' \wedge y > y' \end{array}$$

We next present an algorithm that uses the abstraction α to compute (a set of abstract transitions that represents) a transition invariant. The algorithm terminates because the set of abstract transitions $\mathcal{T}^{\#}_{\mathcal{P}}$ is finite.

Algorithm (TPA)
Transition invariants via transition predicate abstraction.

```
Input:  program $P = (\Sigma, \mathcal{T}, \rho)$
        set of transition predicates $\mathcal{P}$
        abstraction $\alpha$ defined by $\mathcal{P}$ (according to Def. 20)
Output: set of abstract transitions $P^\# = \{T_1, \dots, T_n\}$
        such that $T_1 \cup \dots \cup T_n$ is a transition invariant

$P^\# := \{\alpha(\rho_\tau) \mid \tau \in \mathcal{T}\}$
repeat
    $P^\# := P^\# \cup \{\alpha(T \circ \rho_\tau) \mid T \in P^\#,\ \tau \in \mathcal{T},\ T \circ \rho_\tau \neq \emptyset\}$
until no change
```

Our notation $P^\#$ for the set of abstract transitions computed by Algorithm TPA stems from [27]. There, $P^\#$ is called an abstract transition program. In contrast to [27] we do not consider edges between the abstract transitions.

Theorem 22 (TPA). *Let $P^\# = \{T_1, \dots, T_n\}$ be the set of abstract transitions computed by Algorithm* TPA*. If every abstract relation $T_1, \dots, T_n$ is well-founded, then program P is terminating.*

Proof. The union of the abstract relations $T_1 \cup \dots \cup T_n$ is a transition invariant. If every abstract relation $T_1, \dots, T_n$ is well-founded, the union $T_1 \cup \dots \cup T_n$ is a disjunctively well-founded transition invariant and by Theorem 16 the program P is terminating. □

Example 23. Consider the program P in Figure 4 and the set of transition predicates $\mathcal{P}$ in Example 19. The output of Algorithm TPA is

$$P^\# = \{x > x', \quad x = x' \wedge y > y'\}$$

Both abstract transitions in $P^\#$ are well-founded. Hence P is terminating.

4 Proof rules and abstract domains

In this section, we compare methods based on size-change termination and transition invariants wrt. the underlying proof rules (Section 4.2) and wrt. the underlying abstract domains (Section 4.3). As a preliminary, we translate the setting of size-change termination to transition-based programs (Section 4.1).

We have defined size-change termination for functional programs, and transition invariants for transition-based programs. For the purpose of comparison, we now restrict size-change termination to the transition-based programs that we obtain from translating functional programs.

From now on, we deal only with *tail-recursive* functional programs where all functions use a common variable name space (the latter condition is not a proper restriction since we can always add redundant parameters, and rename parameters if necessary to ensure uniqueness).

Under this restriction, the translation of a functional program into a transition-based program $P = (\Sigma, \mathcal{T}, \rho)$ with the same termination behavior is straightforward:

- the set of states Σ is the Cartesian product of the set of locations Loc and the data domains for the function parameters; we have a location $\ell_{\mathtt{f}}$ in Loc for every function $\mathtt{f}$,
- the set of transitions $\mathcal{T}$ contains a transition τ_c for each call c,
- the transition relation ρ_{τ_c} of the transition τ_c is defined by

$$\rho_{\tau_c} = \{((\ell_{\mathtt{f}}, \boldsymbol{v}), (\ell_{\mathtt{g}}, \boldsymbol{w})) \mid \mathcal{E}[\![\mathtt{e}_1]\!]\boldsymbol{v} = w_1, \ldots, \mathcal{E}[\![\mathtt{e}_n]\!]\boldsymbol{v} = w_n\}.$$

  if the call c occurs in a function definition of the form

$$\mathtt{f}(\mathtt{x}_1, \ldots, \mathtt{x}_n) = \ldots c : \mathtt{g}(\mathtt{e}_1, \ldots, \mathtt{e}_n) \ldots \ .$$

From now on we fix a transition-based program P which stems from the translation of a (tail-recursive) functional program. The size-change termination of P is equivalent to the size-change termination of the original functional program.

Example 24. The transition-based program in Figure 3 is obtained by translating the functional program in Figure 1 after adding parameters in order to obtain a common variable name space.

```
f(x,y,u,v,w) = if x=0 then y else 1: g(x,y,x,y,y)
g(x,y,u,v,w) = if v>0 then 2: g(x,y,u,v-1,2*w)
                          else 3: f(u-1,w,u,v,w)
```

Fig. 5. The functional program of Figure 1, modified to have a common name space.

4.1 From graphs to transition relations

Suppose size-change graph G safely describes call c as in Definition 3. Clearly G expresses a conjunction of relations (each one either $\downarrow$ or $\overset{=}{\downarrow}$) between some parameters of source $\mathtt{f}$ and some parameters of target $\mathtt{g}$. By the common name space assumption, $\mathtt{f}$ and $\mathtt{g}$ have the same parameters. This section shows that graph G defines an abstraction of c's transition relation ρ_{τ_c}.

Since a graph is not a set of pairs of states (and not a formula either), we devise a notation $\Phi(G)$ for the set of state pairs described by size-change graph G. Therefore we define as a first step a class of binary relations that represent the atomic pieces of information contained in a size-change graph (which are: source, target, value decrease and strict value decrease).

Definition 25 (Set of size-change predicates $\mathcal{P}_{SCT}$). *We call a binary relation a* size-change predicate *if it is defined by one of the formulas*

$$\begin{array}{l} pc = \ell \\ pc' = \ell \\ z_i \geq z'_j \\ z_i > z'_j \end{array}$$

where the variable pc ranges over the set of program locations Loc*, and z_i and z_j are program variables. We use $\mathcal{P}_{SCT}$ for the (finite) set of all size-change predicates for the current program P.*

As a second step, we define the relation $\Phi(G)$ to be a conjunction of these size-change predicates (parallel to Definition 2).

Definition 26. *Given a size-change graph $G : \ell \rightarrow \ell'$ with arc set E, define the binary relation over states $\Phi(G) \subseteq \Sigma \times \Sigma$ by the following formula.*

$$pc = \ell \wedge pc' = \ell' \wedge \bigwedge\{z_i \geq z'_j \mid (z_i, \overset{=}{\downarrow}, z_j) \in E\} \wedge \bigwedge\{z_i > z'_j \mid (z_i, \downarrow, z_j) \in E\}$$

Example 27. The binary relations over states assigned to the size-change graphs of Figure 2 are the following.

$\Phi(G_1)$ is $pc = \mathtt{f} \wedge pc' = \mathtt{g} \wedge\ x \geq x' \wedge\ y \geq y' \wedge\ \ x \geq u' \wedge\ y \geq v' \wedge\ y \geq w'$
$\Phi(G_2)$ is $pc = \mathtt{g} \wedge pc' = \mathtt{g} \wedge\ x \geq x' \wedge\ y \geq y' \wedge\ \ u \geq u' \wedge\ v > v'$
$\Phi(G_3)$ is $pc = \mathtt{g} \wedge pc' = \mathtt{f} \wedge\ u > x' \wedge\ w \geq y' \wedge\ u \geq u' \wedge\ v \geq v' \wedge\ w \geq w'$

The inclusion $R_P \subseteq \Phi(G_1) \vee \Phi(G_2) \vee \Phi(G_3)$ means that the transition relation R_P is approximated by the set $\{G_1, G_2, G_3\}$ of size-change graphs. The inclusion is strict, meaning that the approximation loses precision. An instance of precision loss: the set $\mathcal{P}_{SCT}$ does not contain any of the transition predicates $x \neq 0$, $v > 0$, $v = 0$ that account for the tests.

The following definition extends Definition 3 from a functional program to its translation to a transition-based program P.

Definition 28. *Let G_τ be the size-change graph assigned to the transition τ of program P. We say that G_τ is safe for τ if the inclusion $\rho_\tau \subseteq \Phi(G_\tau)$ holds. A set of graphs $\{G_\tau \mid \tau \in \mathcal{T}\}$ is a* safe description *of program P if G_τ is safe for τ for every transition τ of P.*

We now consider the composition of size-change graphs (Definition 6).

Lemma 29. *The composition of the two size-change graphs $G_1 : \ell \rightarrow \ell'$ and $G_2 : \ell' \rightarrow \ell''$ overapproximates the composition of the relations they define, i.e.,*

$$\Phi(G_1) \circ \Phi(G_2) \subseteq \Phi(G_1; G_2).$$

Corollary 30. *If G_τ is a size-change graph that is safe for τ, then for every transition relation T and every size-change graph G such that $G; G_\tau$ is defined*

$$T \subseteq \Phi(G) \textit{ implies } T \circ \rho_\tau \subseteq \Phi(G; G_\tau)$$

Proof. $\rho_\tau \subseteq \Phi(G_\tau)$ by Definition 28, so $T \circ \rho_\tau \subseteq \Phi(G) \circ \Phi(G_\tau) \subseteq \Phi(G; G_\tau)$ by Lemma 29. □

Lemma 31. *Let G be a size-change graph such that source and target of G coincide. If G has an arc of form $x \xrightarrow{\downarrow} x$ then the relation $\Phi(G)$ is well-founded.*

Proof. Let G be a size change graph with an an arc $x \xrightarrow{\downarrow} x$. By Definition 26 the relation $\Phi(G)$ is a subset of the relation $x' < x$. Since $x' < x$ is well-founded, all subsets are also well-founded. □

Remark: The converse of Lemma 31 is false, e.g., for the arc set $\{x \xrightarrow{\downarrow} y, y \xrightarrow{\downarrow} x\}$.

4.2 SCT and disjunctive well-foundedness

Theorem 32 (Idempotence and well-foundedness). *If every* idempotent *size-change graph in the closure $cl(\mathcal{G})$ of the set of size-change graphs $\mathcal{G}$ defines a well-founded relation, i.e.,*

$$\forall G \in cl(\mathcal{G}) : \; G; G = G \implies \Phi(G) \textit{ well-founded}$$

then $\Phi(G)$ is well-founded for every *graph in $cl(\mathcal{G})$.*

Proof. Let $G \in cl(\mathcal{G})$ be a size-change graph.

Case 1: Source and target of G do not coincide.
Then there exist two different locations ℓ, ℓ' such that $\Phi(G) \subseteq pc = \ell \wedge pc' = \ell'$ and therefore $\Phi(G) \circ \Phi(G) = \emptyset$ which implies that $\Phi(G)$ is well-founded.

Case 2: Source and target of G coincide.
Then G^n is defined for all $n \in I\!N$. The semigroup $(\{G^n \mid n \in \mathbb{N}\}, ;)$ is finite and has therefore an idempotent element G^k (since every finite semigroup has an idempotent element). By assumption $\Phi(G^k)$ is well-founded. By induction over k and Lemma 29 the inclusion $\Phi(G)^k \subseteq \Phi(G^k)$ holds. Hence $\Phi(G)^k$ is well-founded and therefore also $\Phi(G)$ is well-founded (Reason: If a relation r is not well-founded, then for all $n \in \mathbb{N}$, r^n is not well-founded.)

Therefore, for every $G \in cl(\mathcal{G})$ the transition $\Phi(G)$ is well-founded. □

Since size-change termination is equivalent to the premise of Theorem 32, and its conclusion can be expressed in terms of disjunctive well-foundedness, we obtain the following statement directly.

Corollary 33 (SCT and disjunctive well-foundedness). *Let $\mathcal{G}$ be a set of size-change graphs that is a safe description of program P. If program P is size-change terminating for a set of size-change graphs $\mathcal{G}$ that is a safe description of P, then the relation defined by its closure $cl(\mathcal{G})$*

$$\bigcup \{\Phi(G) \mid G \in cl(\mathcal{G})\}$$

is a disjunctively well-founded transition invariant for P.

Proof. We first show, that the disjunction is a transition invariant, i.e.,

$$R_P^+ \subseteq \bigcup \{\Phi(G) \mid G \in cl(\mathcal{G})\}.$$

Let $(s, s') \in R_P^+$. By definition of R_P^+ there is a sequence of transition relations $\rho_{\tau_1}, \rho_{\tau_2}, \ldots, \rho_{\tau_n}$ such that (s, s') is contained in the composition $\rho_{\tau_1} \circ \rho_{\tau_2} \circ \cdots \circ \rho_{\tau_n}$.

For every such sequence there is a size-change graph $G \in cl(\mathcal{G})$ such that the inclusion $\rho_{\tau_1} \circ \rho_{\tau_2} \circ \cdots \circ \rho_{\tau_n} \subseteq \Phi(G)$ holds. This can be shown by induction over n, where the induction basis holds by Definition 28 and the induction step follows from Corollary 30. Hence $(s, s') \in \Phi(G)$ for some $G \in cl(\mathcal{G})$, so $(s, s') \in \bigcup\{\Phi(G) \mid G \in cl(\mathcal{G})\}$.

Since P is size-change terminating for $\mathcal{G}$, every idempotent size change-graph $G \in cl(\mathcal{G})$ contains an arc of form $x \overset{\downarrow}{\rightarrow} x$ (by Theorem 10, or Theorem 4 of [22]). By Lemma 31, for every idempotent size change-graph $G \in cl(\mathcal{G})$ the relation $\Phi(G^k)$ is well-founded. Hence by Theorem 32 for every size change-graph $G \in cl(\mathcal{G})$ the relation $\Phi(G^k)$ is well-founded. Therefore $\bigcup \{\Phi(G) \mid G \in cl(\mathcal{G})\}$ is a disjunctively well-founded transition invariant for P. □

4.3 Size-change graphs and transition predicate abstraction

Lemma 34. *Let α be the abstraction function for the set of size-change predicates $\mathcal{P}_{SCT}$. If the size-change graph G denotes a superset of a binary relation over states T, then the size-change graph G denotes a superset of the abstract transition $\alpha(T)$, i.e.*

$$T \subseteq \Phi(G) \quad \textit{implies} \quad \alpha(T) \subseteq \Phi(G)$$

Proof. For every size-change graph G, the relation $\Phi(G)$ is a conjunction of size-change predicates. Therefore the inclusion $\alpha(T) \subseteq \Phi(G)$ holds if for every $p \in \mathcal{P}_{SCT}$ the inclusion $\Phi(G) \subseteq p$ implies the inclusion $\alpha(T) \subseteq p$. Let p be a size-change predicate. Let $\Phi(G) \subseteq p$. Assume that the inclusion $T \subseteq \Phi(G)$ holds. Then the inclusion $T \subseteq p$ holds and by Definition 20 the inclusion $\alpha(T) \subseteq p$ holds.

Corollary 35. *Let α be the abstraction function for the set of size-change predicates $\mathcal{P}_{SCT}$. The abstract transition $\alpha(\rho_\tau)$ is a subset of the denotation of any size-change graph G_τ that is safe for τ, formally*

$$\alpha(\rho_\tau) \subseteq \Phi(G_\tau)$$

This inclusion can be strict in case G_τ is not the “best” description of ρ_τ. An extreme example: G_τ has the empty set of arcs.

Lemma 36. *Let $cl(\mathcal{G})$ be the closure (Definition 8) for a set of size-change graphs $\mathcal{G}$ that is a safe description of program P. Let $P^\#$ be a set of abstract transitions computed by Algorithm* TPA *for the set of size-change predicates $\mathcal{P}_{SCT}$.*

For every abstract transition T in $P^\#$ there exists a size-change graph G in $cl(\mathcal{G})$ that contains T, formally

$$\Phi(G) \supseteq T.$$

Proof. Let $T \in P^\#$. By Algorithm TPA there is a sequence of transitions $\tau_1, \dots, \tau_2$ such that the equation

$$T = \alpha(\dots\alpha(\alpha(\rho_{\tau_1}) \circ \rho_{\tau_2}) \dots \circ \rho_{\tau_n})$$

holds. Let G_{τ_i} be a graph that is safe for τ_i and G be a size-change graph defined by the following equation.

$$G = G_{\tau_1}; \dots; G_{\tau_n}$$

The inclusion $\Phi(G) \supseteq T$ holds by induction, where the induction basis holds by Definition 28 and Lemma 34 and the induction step follows from Corollary 30.

Theorem 37. *Let $\mathcal{G}$ be a set of size-change graphs that is a safe description of program P. Let $P^\#$ be a set of abstract transitions computed by Algorithm* TPA *for the set of size-change predicates $\mathcal{P}_{SCT}$. If P is size-change terminating for $\mathcal{G}$ then $P^\#$ defines a disjunctively well-founded transition invariant.*

Proof. The output of Algorithm TPA $P^\#$ defines a transition invariant for P. If P is size-change terminating, then by Corollary 33 every element of $\{\Phi(G) \mid G \in cl(\mathcal{G})\}$ is well-founded. Hence by Lemma 36 every element of $P^\#$ is well-founded. Therefore $\bigcup P^\#$ is a disjunctively well-founded transition invariant. □

5 Base algorithms

In this section, we categorize the base algorithm in the different termination analyses by the decision problem that it solves, and then establish an formal connection between the decision problems.

Part of the input of those decision problems will be a *transition abstraction.* A transition abstraction fixes a set of abstract values $\mathcal{T}^\#$ and their meaning via the *denotation* function γ. Each abstract value a denotes a relation over states, i.e., $\gamma(a) \subseteq \Sigma \times \Sigma$. The transition abstraction fixes also a distinguished abstract value a_τ for every transition τ of the given program. A termination analysis starts with those values.

Programs, transition relations, states, etc. are as defined in Section 3.

Definition 38 (Transition Abstraction). *Given a program $P = (\Sigma, \mathcal{T}, \rho)$, a* transition abstraction *is a triple*

$$(\mathcal{T}^{\#}, \gamma, \{a_\tau \,|\, \tau \in \mathcal{T}\})$$

consisting of:

1. *a finite set $\mathcal{T}^{\#}$ of abstract values called* abstract relations*,*
2. *a* denotation *function γ that assigns to each abstract relation a relation over the program's states, i.e.,*

$$a \in \mathcal{T}^{\#} \implies \gamma(a) \subseteq \Sigma \times \Sigma$$

3. *a set of distinguished abstract values indexed by transitions τ of the program, i.e.,*

$$a_\tau \in \mathcal{T}^{\#}, \quad \textit{for} \quad \tau \in \mathcal{T}.$$

The abstract relation for the transition τ *must safely abstract the transition relation defined by τ, formally*

$$\rho_\tau \subseteq \gamma(a_\tau)$$

for each transition τ in $\mathcal{T}$.

A set X of abstract relations denotes their union, i.e.,

$$\gamma(X) = \bigcup\{\gamma(a) \mid a \in X\}, \quad \text{for} \quad X \subseteq \mathcal{T}.$$

Example 39 (SCT). In order to rephrase *size-change analysis* as presented in Section 2, one may use the transition abstraction where:

- the abstract relations $a \in \mathcal{T}^{\#}$ are size-change graphs G,
- the denotation function γ is the function Φ of Definition 26, i.e., a graph G denotes the transition relation defined by the formula $\Phi(G)$,
- the distinguished abstract transitions a_τ for transitions τ are exactly the size-change graphs G_τ for calls τ in the set $\mathcal{G}$ fixed in Definition 5.

Since we translate the function Φ on size-change graphs to the denotation function γ, the safety required for the size-changes graphs G_τ translates directly to the safety requirement for the a_τ in Definition 38; see Definition 28.

Example 40 (TPA). In order to rephrase *transition predicate abstraction* as presented in Section 3.3, one may use the transition abstraction where:

- the abstract relations $a \in \mathcal{T}^{\#}$ are the *abstract transitions* $p_1 \wedge \ldots \wedge p_m$, which are conjunctions of transition predicates $p_j \in \mathcal{P}$, for the given set of transition predicates $\mathcal{P}$.

$$\mathcal{T}^{\#} = \{p_1 \wedge \ldots \wedge p_m \mid p_1, \ldots, p_m \in \mathcal{P},\ 0 \leq m\}$$

- the denotation function γ is essentially the identity function, i.e., the denotation of a conjunction of transition predicates is the intersection of the transition relations they denote,
- the abstract transition a_τ is the abstraction α applied to the transition relation ρ_τ. This is the strongest abstract transition that contains ρ_τ, or, equivalently, is the conjunction of all transition predicates in $\mathcal{P}$ that contain ρ_τ; see Definition 20.

$$a_\tau = \alpha(\rho_\tau) \; (= \bigwedge\{p \in \mathcal{P} \mid \rho_\tau \subseteq p\})$$

5.1 Transformer on abstract relations

Given a transition τ of the program, we consider a function $F^\#_\tau$ that assigns to each abstract relation a another abstract relation $a' = F^\#_\tau(a)$. The idea is that the function $F^\#_\tau$ abstracts the relational composition with the transition relation ρ_τ (i.e., it abstracts the function F_τ such that $F_\tau(T) = T \circ \rho_\tau$).

For better legibility, we write $F^\#(a, \tau)$ instead of $F^\#_\tau(a)$. We call $F^\#$ a (parametrized) abstract-relation transformer.

In this section, we fix a program $P = (\Sigma, \mathcal{T}, \rho)$ and a transition abstraction $(\mathcal{T}^\#, \gamma, \{a_\tau \,|\, \tau \in \mathcal{T}\})$ defining a set of abstract relations, their denotation, and a set of abstract relations for the transitions of the program.

Definition 41 (Abstract-relation transformer $F^\#$). *Given a program $P = (\Sigma, \mathcal{T}, \rho)$ and a transition abstraction $(\mathcal{T}^\#, \gamma, \{a_\tau \,|\, \tau \in \mathcal{T}\})$, an* abstract-relation transformer *is a function*

$$F^\# : \mathcal{T}^\# \times \mathcal{T} \to \mathcal{T}^\#$$

such that

$$\gamma(F^\#(a, \tau)) \;\supseteq\; \gamma(a) \circ \rho_\tau$$

In words, the application of $F^\#$ to the abstract relation a and the transition τ overapproximates the relational composition of the relation denoted by a with the transition relation defined by τ.

Example 42 (Continuing Example 40).

Continuing Example 40, where we use abstract transitions (i.e., conjunctions of transition predicates) as abstract relations, one may define the abstract-relation transformer by

$$F^\#(T, \tau) = \alpha(T \circ \rho_\tau),$$

where α is the abstraction function defined by the given set of transition predicates; see Definition 20.

We next introduce an expression to denote a set of abstract relations. We call it “the least fixpoint of the abstract-relation transformer $F^\#$” although, strictly speaking, it is not a fixpoint of the function $F^\#$. (Instead, it is the fixpoint of a functional that can be derived from $F^\#$. This functional ranges over the powerset lattice generated by the abstract relations. The least fixpoint is the *least* fixpoint of this functional *above* the set $\{a_\tau \mid \tau \in \mathcal{T}\}$, i.e., the set of abstract relations a_τ for the transitions of the given program P. For notational economy we will not formally define the lattice and the functional.)

Definition 43 (Least fixpoint of the abstract-relation transformer $F^\#$). *Given a program $P = (\Sigma, \mathcal{T}, \rho)$, a transition abstraction $(\mathcal{T}^\#, \gamma, \{a_\tau \mid \tau \in \mathcal{T}\})$, and an abstract-relation transformer $F^\#$, the* "least fixpoint of the abstract-relation transformer $F^\#$", *written*

$$lfp(\{a_\tau \mid \tau \in \mathcal{T}\}, F^\#),$$

is defined as the least set of abstract relations X such that

- *X contains the set of abstract relations a_τ for each transition τ,*

$$X \supseteq \{a_\tau \mid \tau \in \mathcal{T}\}$$

- *and X is closed under application of the abstract-relation transformer for every transition τ, i.e., the application of $F^\#$ to an abstract relation a in X and a transition τ is again an element of X, formally*

$$X \supseteq \{F^\#(a, \tau) \mid a \in X,\ \tau \in \mathcal{T}\}.$$

Example 44 (Continuing Example 42). The least fixpoint $lfp(\{a_\tau \mid \tau \in \mathcal{T}\}, F^\#)$ of the abstract-relation transformer $F^\#$ defined in Example 42 is the set of abstract transitions $P^\#$ which is the output of Algorithm TPA.

Lemma 45. *The least fixpoint of the abstract-relation transformer $F^\#$ can be indexed by the sequences of transitions $\tau_1, \ldots, \tau_n$, i.e.,*

$$lfp(\{a_\tau \mid \tau \in \mathcal{T}\}, F^\#) = \{a_{\tau_1 \ldots \tau_n} \mid n \geq 1, \tau_1, \ldots, \tau_n \in \mathcal{T}\}$$

where $a_{\tau_1 \tau_2} = F^\#(a_{\tau_1}, \tau_2)$, $a_{\tau_1 \tau_2 \tau_3} = F^\#(a_{\tau_1 \tau_2}, \tau_3)$, etc..

The following lemma states that we can use a transition abstraction and an abstract-relation transformer to compute a transition invariant for the program P.

Lemma 46 (Transition invariants via the abstract-relation transformer $F^\#$). *Given a program $P = (\Sigma, \mathcal{T}, \rho)$ with transition relation R_P, a transition abstraction $(\mathcal{T}^\#, \gamma, \{a_\tau \mid \tau \in \mathcal{T}\})$, and an abstract-relation transformer $F^\#$, the least fixpoint of the abstract-relation transformer $F^\#$ denotes a transition invariant for P, i.e.,*

$$R_P^+ \subseteq \gamma(lfp(\{a_\tau \mid \tau \in \mathcal{T}\}, F^\#)).$$

We next define a decision problem, and then characterize a specific class of termination analyses as decision procedures for the problem.

Problem: LFP CHECKING FOR ABSTRACT RELATIONS

Input:

- a program $P = (\Sigma, \mathcal{T}, \rho)$
- a transition abstraction $(\mathcal{T}^{\#}, \gamma, \{a_\tau \,|\, \tau \in \mathcal{T}\})$
- an abstract-relation transformer $F^{\#} : \mathcal{T}^{\#} \times \mathcal{T} \rightarrow \mathcal{T}^{\#}$
- a subset $\mathsf{GOOD} \subseteq \mathcal{T}^{\#}$ such that every element of GOOD denotes a well-founded relation

Property: $\mathit{lfp}(\{a_\tau \,|\, \tau \in \mathcal{T}\}, F^{\#}) \subseteq \mathsf{GOOD}$

Theorem 47 (Lfp Checking for Abstract Relations and Termination). *The program P is terminating if the decision procedure* LFP CHECKING FOR ABSTRACT RELATIONS *answers yes.*

This decision procedure is a *semi-test* for termination: a yes-answer is definite, a no-answer is no.

Proof. If the procedure answers yes, the least fixpoint of the abstract-relation transformer $F^{\#}$ is not only a transition invariant (by Lemma 46) but it is also disjunctively well-founded. Thus, Theorem 16 applies and P is terminating. □

Next, a complexity result. To make the statement simpler, we (reasonably) assume henceforth that the number of abstract relations is greater than the number of transitions, i.e., $|\mathcal{T}^{\#}| \geq |\mathcal{T}|$. In the setting of Examples 39, 40, and 42, the number of abstract relations is:

- (in the setting of SCT, as in Examples 39 and 42) exponential in the square of the size of the program (to be precise, it is bound by 3^{p^2} where p is the number of program variables),
- (in the setting of TPA, as in Examples 40 and 42) exponential in the number of transition predicates in $\mathcal{P}$.

Theorem 48. LFP CHECKING FOR ABSTRACT RELATIONS *is decidable in time polynomial in $|\mathcal{T}^{\#}|$, and (by another algorithm) in space $\mathcal{O}(log^2|\mathcal{T}^{\#}|)$.*

Proof. Consider a directed graph Γ. The nodes of Γ are the abstract relations in $\mathcal{T}^{\#}$ plus two special nodes *init* and *fin* so the graph Γ contains $|\mathcal{T}^{\#}| + 2$ nodes. Let $a, a' \in \mathcal{T}^{\#}$, we define that

- Γ contains an edge from a to a' if and only if there is a $\tau \in \mathcal{T}$ such that $F^{\#}(a, \tau) = a'$,
- Γ contains an edge from *init* to a if and only if $a \in \{a_\tau \,|\, \tau \in \mathcal{T}\}\}$,
- and Γ contains an edge from a to *fin* if and only if $a \notin \mathsf{GOOD}$.

We conclude: $\mathit{\Gamma}$ contains a path from *init* to a iff $a \in \mathit{lfp}(\{a_\tau \mid \tau \in \mathcal{T}\}, F^\#)$. Further, $\mathit{\Gamma}$ contains a path from *init* to *fin* iff $\mathit{lfp}(\{a_\tau \mid \tau \in \mathcal{T}\}, F^\#) \not\subseteq \mathsf{GOOD}$. For time: the graph can be searched by, for example, Dijkstra's algorithm. For space: a well-known result by Savitch is that existence of a path in a directed graph with n nodes can be decided in space $O(\log^2 n)$. □

5.2 Composition of abstract relations

In Section 5.1, we used the function $F^\#$ to abstract the relational composition of relations with the transition relations ρ_τ for the program transitions τ. In this section, we introduce a binary operator on abstract relations in order to abstract the binary relational composition operator.

Definition 49 (Abstract composition $\circ^\#$). *Given a program $P = (\Sigma, \mathcal{T}, \rho)$ and a transition abstraction $(\mathcal{T}^\#, \gamma, \{a_\tau \mid \tau \in \mathcal{T}\})$, an* abstract composition *is a binary operation on abstract relations,*

$$\circ^\# : \mathcal{T}^\# \times \mathcal{T}^\# \to \mathcal{T}^\#$$

such that

$$\gamma(a_1 \circ^\# a_2) \supseteq \gamma(a_1) \circ \gamma(a_2).$$

In words, the abstract composition of two abstract relations a_1 and a_2 overapproximates the relational composition of the two relations denoted by a_1 resp. a_2.

Example 50 (continuing Example 39). In the setting of size-change termination, the composition operator on size-change graphs (written $G_1; G_2$) is an abstract composition by Lemma 29.

Example 51 (continuing Example 40). In the setting of transition predicate abstraction, we can define the abstract composition over abstract transitions T_1 and T_2 (i.e., conjunctions of transition predicates) by $T_1 \circ^\# T_2 = \alpha(T_1 \circ T_2)$, where α is the abstraction function defined by the given set of transition predicates; see Definition 20. Note that, the abstract composition over abstract transitions is in general not associative.

Definition 52 (Closure of an abstract composition). *Given a program $P = (\Sigma, \mathcal{T}, \rho)$, a transition abstraction $(\mathcal{T}^\#, \gamma, \{a_\tau \mid \tau \in \mathcal{T}\})$, and an abstract composition $\circ^\#$, the* "closure of the abstract composition $\circ^\#$", *written*

$$\mathit{cl}(\{a_\tau \mid \tau \in \mathcal{T}\}, \circ^\#)$$

is the smallest set of abstract relations X such that

- *X contains the set of abstract relations a_τ for the transitions τ,*

$$X \supseteq \{a_\tau \mid \tau \in \mathcal{T}\}$$

- *and X is closed under abstract composition, i.e., the abstract composition of two abstract relations a_1 and a_2 in X is again an element in X:*

$$X \supseteq \{a_1 \circ^{\#} a_2 \mid a_1 \in X,\ a_2 \in X\}.$$

Example 53 (Continuing Example 50). If the abstract composition $\circ^{\#}$ is the composition operator on size-change graphs " ; ", the closure in the sense of Definition 52 is the closure in the sense of Definition 8.

The following lemma states (in analogy with Lemma 46) that we can use a transition abstraction and an abstract composition over abstract relations to compute a transition invariant for the program P.

Lemma 54 (Transition invariants via abstract composition $\circ^{\#}$). *Given a program $P = (\Sigma, \mathcal{T}, \rho)$ with transition relation R_P, a transition abstraction $(\mathcal{T}^{\#}, \gamma, \{a_\tau \,|\, \tau \in \mathcal{T}\})$, and an abstract composition $\circ^{\#}$, the closure of the abstract composition denotes a transition invariant for P, i.e.,*

$$R_P^+ \subseteq \gamma(cl(\{a_\tau \,|\, \tau \in \mathcal{T}\}, \circ^{\#})).$$

In analogy to Section 5.1, we state a decision problem. In contrast with Section 5.1, the input contains not an abstract-relation transformer $F^{\#}$, but an abstract composition $\circ^{\#}$. It is checked if the closure of $\circ^{\#}$ is a subset of GOOD.

This enables us to characterize a second class of termination analyses as decision procedures for this problem.

Problem: Closure Checking for Abstract Relations

Input:

- a program $P = (\Sigma, \mathcal{T}, \rho)$
- a transition abstraction $(\mathcal{T}^{\#}, \gamma, \{a_\tau \,|\, \tau \in \mathcal{T}\})$
- an abstract composition $\circ^{\#} : \mathcal{T}^{\#} \times \mathcal{T}^{\#} \to \mathcal{T}^{\#}$
- a subset $\mathsf{GOOD} \subseteq \mathcal{T}^{\#}$ such that every element of GOOD denotes a well-founded relation

Property: $cl(\{a_\tau \,|\, \tau \in \mathcal{T}\}) \subseteq \mathsf{GOOD}$

Theorem 55 (Closure Checking for Abstract Relations and Termination). *Program P is terminating if the decision procedure* Closure Checking for Abstract Relations *answers yes.*

Proof. Analogous to Theorem 47. □

The next result investigates the complexity of the decision problem Closure Checking for Abstract Relations .

Theorem 56. *The problem* Closure Checking for Abstract Relations *is* ptime*-complete in the number of transition relations* $|\mathcal{T}^{\#}|$.

Proof. First, the problem Closure Checking for Abstract Relations is in ptime, since a straightforward bottom-up algorithm can compute and test for well-foundedness all elements in $cl(\{a_\tau \,|\, \tau \in \mathcal{T}\})$. (Remark: we count the well-foundedness test $a \in \mathsf{GOOD}$? as one step.)

Second, we show the problem is ptime-hard by reduction from a known ptime-complete problem to Closure Checking for Abstract Relations. The problem gen is a membership problem for the closure of an operation, defined as follows. **Given:** A finite set W, a binary operation op on W, a subset $V \subseteq W$, and $w \in W$. **To decide:** Is $w \in cl(V, op)$?

Given a gen instance (W, op, V, w), let $P = (\Sigma, \mathcal{T}, \rho)$ be a program such that

- the set of states is the empty set
- the set of transitions $\mathcal{T}$ is V
- the transition relation ρ_τ of every transition τ is the empty set.

Let $(\mathcal{T}^{\#}, \gamma, \{a_\tau \,|\, \tau \in \mathcal{T}\})$ be a transition abstraction such that

- the set of abstract relations is $\mathcal{T}^{\#} = W$
- the denotation γ assigns to each abstract relation the empty set.
- the set of program transition relations is $\{a_\tau \,|\, \tau \in \mathcal{T}\} = V$.

Since every abstract relation denotes the empty relation, $op = \circ^{\#}$ is trivially an abstract composition. We choose $\mathsf{GOOD} = W \backslash \{w\}$. This is a valid choice since every abstract relation denotes a well-founded relation.

Clearly $w \notin cl(V, op)$ if and only if the inclusion $cl(\{a_\tau \,|\, \tau \in \mathcal{T}\}, op) \subseteq \mathsf{GOOD}$ holds. The complexity result follows since the negation of any ptime-complete problem is also ptime-complete. □

Abstract-relation transformers $F^{\#}$ *versus abstract composition* $\circ^{\#}$*: Precision.* A termination analysis A has *higher precision* than a termination analysis B if A returns a yes-answer whenever B does, and possibly strictly more often (a yes-answer is definite in proving termination of the input program).

One might expect, by the complexity results above, that a termination analysis based on abstract composition has higher precision than one based on abstract-relation transformers (as a trade-off for the higher complexity). In fact, one can always define a termination analysis based on abstract-relation transformers that has higher precision than one based on abstract composition, sometimes strictly higher.[5] We distinguish two distinct causes for the difference in precision.

- Both the abstract relation transformer $F^{\#}(a, \tau)$ and the abstract composition $a \circ^{\#} a_\tau$ define an abstraction of the relation $\gamma(a) \circ \rho_\tau$. However the former can be strictly more precise than the latter, since the abstract composition has to be an abstraction of a superset of $\gamma(a) \circ \gamma(a_\tau)$. In fact, there

[5] An example can be found in the online version of this paper [18].

are cases of abstract-relation transformers $F^{\#}$ with a yes-answer (proving that the input program terminates) such that no abstract composition $\circ^{\#}$ exists with a yes-answer.

- A set of abstract relations X that contains all elements $a \circ^{\#} a_\tau$ where $a \in X$ can be strictly smaller than one that contains all elements $a_1 \circ^{\#} a_2$ where $a_1 \in X$ and $a_2 \in X$.
  Even if we require the abstract-relation transformers $F^{\#}$ to be defined by $F^{\#}(a, \tau) = a \circ^{\#} a_\tau$, there are cases where the Lfp Checking for Abstract Relations returns a yes-answer but the Closure Checking for Abstract Relations returns a no-answer.

Finally, a potential advantage of abstract composition above abstract-relation transformers. The latter can be defined and constructed only once the input program with its transitions τ is known. The former can be defined and constructed in a pre-processing step, once the set of abstract relations $\mathcal{T}^{\#}$ is fixed.

5.3 Special case: associative composition of abstract relations

In this section we investigate the special case where the abstract composition $\circ^{\#}$ of abstract relations is associative. The example of size-change termination falls into this case, i.e., the composition of size-change graphs is associative. We will see that associativity has two consequences.

- The decision problem Closure Checking for Abstract Relations can be reduced to the decision problem Lfp Checking for Abstract Relations . We thus obtain a better upper bound for the complexity.
- The decision problem can be further reduced to a decision problem where the inclusion in the question "$cl(\{a_\tau \mid \tau \in \mathcal{T}\}) \subseteq \mathsf{GOOD}$" is restricted to a subset of abstract relations. The subset consists of *idempotent* elements a, i.e., where $a \circ^{\#} a = a$. Thus, we can replace the input parameter GOOD by a subset of GOOD (containing idempotent elements only), and reserve the well-foundedness check for only those elements.

We recall that both of the above decision problems require the well-foundedness of every relation denoted by an abstract relation a in GOOD.

Theorem 57. *The closure of an* associative *abstract composition $\circ^{\#}$ equals the least fixpoint of the abstract-relation transformer $F^{\#}$ defined by*

$$F^{\#}(a, \tau) = a \circ^{\#} a_\tau$$

i.e.,

$$lfp(\{a_\tau \,|\, \tau \in \mathcal{T}\}, F^{\#}) = cl(\{a_\tau \,|\, \tau \in \mathcal{T}\}, \circ^{\#}).$$

Corollary 58. *If the abstract composition $\circ^{\#}$ is associative,* Closure Checking for Abstract Relations *is decidable in space $\mathcal{O}(log^2 |\mathcal{T}^{\#}|)$.*

We note the correspondence to Theorem 11.

Theorem 59. *If every* idempotent *element in the closure $cl(\{a_\tau \,|\, \tau \in \mathcal{T}\}, \circ^\#)$ of an* associative *abstract composition denotes a well-founded relation, then* every *element (idempotent or not) denotes a well-founded relation.*

Proof. We show that whenever some element of $cl(\{a_\tau \,|\, \tau \in \mathcal{T}\})$ denotes a relation that is not well-founded, then $cl(\{a_\tau \,|\, \tau \in \mathcal{T}\})$ contains an idempotent element that denotes a non-well-founded relation.

Let $a \in \mathcal{T}^\#$ be an abstract relation. We define the following notation recursively for $n \geq 1$.

$$a^n = \begin{cases} a & \text{if } n = 1 \\ a^{n-1} \circ^\# a & \text{otherwise} \end{cases} \qquad \gamma(a)^n = \begin{cases} \gamma(a) & \text{if } n = 1 \\ \gamma(a)^{n-1} \circ \gamma(a) & \text{otherwise} \end{cases}$$

Since $(cl(\{a_\tau \,|\, \tau \in \mathcal{T}\}), \circ^\#)$ is a finite semigroup, $(\{a^n \mid n \geq 1\}, \circ^\#)$ is also a finite semigroup. By stepwise induction and definition of an abstract composition, we get that the inclusion

$$\gamma(a)^n \subseteq \gamma(a^n)$$

holds for $n \geq 1$.

A well-known result is that every finite semigroup has an idempotent element. Let $k \in I\!N$ be a natural number, such that a^k is idempotent. Assume the relation $\gamma(a)$ is not well-founded. Then the relation $\gamma(a)^k$ and its superset $\gamma(a^k)$ are also not well-founded. Hence the idempotent element a^k denotes a relation that is not well-founded. □

This proves a slightly stronger result than Theorem 59: a sufficient condition is associativity of the abstract composition on the elements of the closure.

In the decision problem we define below, one may obviously restrict the elements in the input parameter GOOD to idempotent elements.

Problem: Associative Closure Checking for Abstract Relations

Input:

- a program $P = (\Sigma, \mathcal{T}, \rho)$.
- a transition abstraction $(\mathcal{T}^\#, \gamma, \{a_\tau \,|\, \tau \in \mathcal{T}\})$.
- an *associative* abstract composition $\circ^\#$
- a subset $\mathsf{GOOD} \subseteq \mathcal{T}^\#$ such that every element of GOOD denotes a well-founded relation.

Property: $\{a \in cl(\{a_\tau \,|\, \tau \in \mathcal{T}\}) \mid a \text{ is idempotent}\} \subseteq \mathsf{GOOD}$

Example 60. In the setting of size-change termination, where the transitions τ are the calls, the abstract relations are the size-change graphs, the (associative!) abstract composition is the composition operator "$;$" of size-change graphs, we choose GOOD to be the set of all *idempotent* size-change graphs G with an arc $z \xrightarrow{\downarrow} z$ (the denotation of G is then a well-founded relation).

Theorem 61 (Associative Closure Checking for Abstract Relations and Termination). *Program P is terminating if the answer to the decision problem* ASSOCIATIVE CLOSURE CHECKING FOR ABSTRACT RELATIONS *is yes.*

Proof. by Theorem 55 (or Theorem 57 together with Theorem 47) and Theorem 59. □

6 Discussion: qualitative differences

The research on concepts and methods based on size-change termination (SCT) resp. transition invariants (TI) involves somewhat different assumptions. All are, however, related to linear *computational paths* and to relations among *first-order values.* This is in contrast to other approaches, for example Gödel's higher-order primitive recursive functions, and analyses of higher-order programs studied among others by Bohr and by Sereni [20, 28, 29]. In this section, we discuss qualitative differences between SCT and TI.

Analysis principles. The SCT analysis traces flow of data in a well-founded data set between *single variables* over all of a program's transition sequences. It reports termination if every infinite transition sequence would cause an infinitely descending value flow between variables. A TI analysis, in contrast, focuses on showing that the program's overall state transition relation is well-founded; there is no a-priori known well-founded data set in which to trace program data flow.

SCT models are uninterpreted. SCT program data may be any well-founded set, not necessarily well-ordered and not fixed, e.g., to the integers or natural numbers. Thus SCT analysis cannot conclude, e.g., that $x < y$ implies $x+1 \leq y$. The TI frameworks do not explicitly mention a value domain, although practical tools (based on RANKFINDER) search for ranking functions over the (positive or negative) integers.

Intensionality/extensionality: size-change analysis is *intensional*: it works by manipulating not semantic objects themselves, but rather a priori determined syntactic objects that describe them: size-change graphs. TI analyses, in contrast, are formulated *extensionally*, in terms of direct manipulation of semantic values, i.e., binary relations on states. In practice, formulas in first-order logic are used to denote these relations.

Decidability: The size-change termination property is decidable, and. its complexity is understood. The calculations in the SCT analysis are done according to fixed combinatorial techniques known in advance: the definition of " ; " and the recognition of in-place decreases $z \xrightarrow{\downarrow} z$.

In contrast, a TI analysis addresses an undecidable verification problem. As already mentioned, the very motivation behind the work in [26] was to carry over the ideas of [22] to verification methods in the style of *software model checking* [3, 4].[6] A software model checker uses theorem provers and decision procedures as oracles that 'solve' potentially undecidable problems to implement *predicate abstraction* and *counterexample-guided abstraction refinement* (see [3, 4]).

Parametrisation: SCT is a relatively rigid framework. It uses a generic set of building blocks to define size-change graphs for every program. Thus, for example, SCT never traces the flow of values that may increase. It is also insensitive to tests in the program being analysed.

In contrast, the starting point of the TI analysis based on transition predicate abstraction (TPA) is a set of predicates $\mathcal{P}$ that parametrizes the abstraction function α. The TI analysis, if used together with counterexample-guided abstraction refinement, requires (in addition to testing well-foundedness) the ability to compute a suitable approximation to the abstraction function α.

An example of reasoning based on abstraction: The generic class $\mathcal{P}_{SCT}$ captures the expressivity of size-change graphs. Inspection of $\mathcal{P}_{SCT}$ reveals that comparisons can be made only between a current variable value and a next variable value. Thus it is impossible to express relations between two current values, as would be required to model tests in the program being analysed.

Acknowledgements. This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). The second author thanks the Alexander von Humboldt-Stiftung for supporting a stimulating half-year stay at the Institut für Informatik at the University of Freiburg.

References

1. James Avery. Size-change termination and bound analysis. In *FLOPS 2006*, pages 192–207. Springer, 2006.
2. Thomas Ball, Rupak Majumdar, Todd D. Millstein, and Sriram K. Rajamani. Automatic predicate abstraction of C programs. In *PLDI*, pages 203–213. ACM, 2001.
3. Thomas Ball, Andreas Podelski, and Sriram K. Rajamani. Relative completeness of abstraction refinement for software model checking. In *TACAS*, pages 158–172. Springer, 2002.
4. Thomas Ball and Sriram K. Rajamani. The SLAM project: debugging system software via static analysis. In *POPL*, pages 1–3. ACM, 2002.
5. Amir M. Ben-Amram. Size-change termination with difference constraints. *ACM Trans. Program. Lang. Syst.*, 30(3):1–31, 2008.
6. Amir M. Ben-Amram. A complexity tradeoff in ranking-function termination proofs. *Acta Inf.*, 46(1):57–72, 2009.

[6] By principle, software model checking, if based on predicate abstraction (or any other abstraction of a program to a finite-state system), is unable to prove termination of programs with executions of unbounded length.

7. Amir M. Ben-Amram. Size-change termination, monotonicity constraints and ranking functions. *Logical Methods in Computer Science*, 6, 2010.
8. Amir M. Ben-Amram and Michael Codish. A SAT-based approach to size change termination with global ranking functions. In *TACAS*, pages 218–232. Springer, 2008.
9. Amir M. Ben-Amram and Chin Soon Lee. Program termination analysis in polynomial time. *ACM Trans. Program. Lang. Syst.*, 29(1), 2007.
10. Amir M. Ben-Amram and Chin Soon Lee. Ranking functions for size-change termination II. *Logical Methods in Computer Science*, 5(2), 2009.
11. Josh Berdine, Aziem Chawdhary, Byron Cook, Dino Distefano, and Peter W. O'Hearn. Variance analyses from invariance analyses. In *POPL*, pages 211–224. ACM, 2007.
12. Michael Codish, Vitaly Lagoon, and Peter J. Stuckey. Testing for termination with monotonicity constraints. In *ICLP*, pages 326–340. Springer, 2005.
13. Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Termination proofs for systems code. In *PLDI*, pages 415–426. ACM, 2006.
14. Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Summarization for termination: no return! *Formal Methods in System Design*, 35(3):369–387, 2009.
15. Patrick Cousot. Partial completeness of abstract fixpoint checking. In *SARA*, pages 1–25. Springer, 2000.
16. Arne J. Glenstrup and Neil D. Jones. Termination analysis and specialization-point insertion in offline partial evaluation. *ACM Trans. Program. Lang. Syst.*, 27(6):1147–1215, 2005.
17. Alexey Gotsman, Byron Cook, Matthew J. Parkinson, and Viktor Vafeiadis. Proving that non-blocking algorithms don't block. In *POPL*, pages 16–28. ACM, 2009.
18. Matthias Heizmann, Neil D. Jones, and Andreas Podelski. Size-change termination and transition invariants (online version). http://swt.informatik.uni-freiburg.de/staff/heizmann/SCTandTI.pdf, 2010.
19. Neil D. Jones. *Computability and Complexity from a Programming Perspective*. Foundations of Computing. MIT Press, Boston, London, 1 edition, 1997.
20. Neil D. Jones and Nina Bohr. Call-by-value termination in the untyped lambda-calculus. *Logical Methods in Computer Science*, 4(1), 2008.
21. Daniel Kroening, Natasha Sharygina, Aliaksei Tsitovich, and Christoph Wintersteiger. Termination analysis with compositional transition invariants. In *CAV*, pages 89–103. Springer, 2010.
22. Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The size-change principle for program termination. In *POPL*, pages 81–92. ACM, 2001.
23. Zohar Manna and A. Pnueli. *Temporal verification of reactive systems: safety*. Springer, 1995.
24. T. Æ. Mogensen, D. A. Schmidt, and I. H. Sudborough, editors. *The Essence of Computation; Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, volume 2566 of *Lecture Notes in Computer Science*. Springer, 2002.
25. Andreas Podelski and Andrey Rybalchenko. A complete method for the synthesis of linear ranking functions. In *VMCAI*, pages 239–251. Springer, 2004.
26. Andreas Podelski and Andrey Rybalchenko. Transition invariants. In *LICS*, pages 32–41. IEEE Computer Society, 2004.
27. Andreas Podelski and Andrey Rybalchenko. Transition predicate abstraction and fair termination. In *POPL*, pages 132–144. ACM, 2005.
28. Damien Sereni. Termination analysis and call graph construction for higher-order functional programs. In *ICFP*, pages 71–84. ACM, 2007.

29. Damien Sereni and Neil D. Jones. Termination analysis of higher-order functional programs. In *APLAS*, pages 281–297. Springer, 2005.
30. Stephan Swiderski, Michael Parting, Jürgen Giesl, Carsten Fuhs, and Peter Schneider-Kamp. Termination analysis by dependency pairs and inductive theorem proving. In *CADE*, pages 322–338. Springer, 2009.