

Decidable Reasoning About Time in Finite-Domain Situation Calculus Theories

Till Hofmann 

RWTH Aachen University, Germany

University of Oxford, UK

Stefan Schupp 

TU Wien, Austria

Gerhard Lakemeyer 

RWTH Aachen University, Germany

Abstract

Representing time is crucial for cyber-physical systems and has been studied extensively in the situation calculus. The most commonly used approach represents time by adding a real-valued function $\text{time}(a)$ that attaches a time point to each action and consequently to each situation. We show that in this approach, checking whether there is a reachable situation that satisfies a given formula is undecidable, even when the domain contains only finitely many objects. We present an alternative approach based on well-established results from timed automata theory by introducing clocks as real-valued fluents with restricted successor state axioms and comparison operators. With this restriction, we can show that the reachability problem for finite-domain basic action theories is decidable. Finally, we apply our results to Golog program realization by presenting a decidable procedure for determining an action sequence that is a successful execution of a given program.

2012 ACM Subject Classification Theory of computation $\rightarrow$ Logic and verification

Keywords and phrases Situation Calculus, Golog, timed automata, clocks, dense time, reachability, decidability, region abstraction, program realization

Digital Object Identifier 10.4230/OASICS.TIME.2026.11

Related Version *Extended Version*: <http://arxiv.org/abs/2402.03164> [19]

Funding This research was supported by the EU ICT-48 2020 project TAILOR (No 952215). Travel funding to present this paper was provided by the EU project euroFMX (No 101299128).

1 Introduction

Time is a ubiquitous concept in various applications, especially in cases where digital systems interact with the real world such as applications in robotics or generally in cyber-physical systems. Early approaches model discrete time, which allows modeling and reasoning about system states at discrete points in time. This approach gives rise to the problem of choosing an appropriate granularity (resolution) of time to facilitate useful reasoning. The introduction of models which incorporate continuous and dense notions of time circumvents this problem and allows for more expressive modeling and analysis approaches. Dense time has been under extensive investigation in the formal methods community, e.g., in the analysis of hybrid systems which started with the introduction of timed automata (TAs) [3].

The situation calculus (SC) is a well-known first-order formalism for reasoning about dynamically changing worlds, where each change in the world is caused by an *action*. Among others, it permits a representation of continuous time by attaching a time of occurrence to each action and hence also each situation [26, 28]. In this paper, we focus on *reachability*, i.e., checking whether there is a reachable situation where some formula ϕ holds. We show that the standard representation of time already makes the reachability problem undecidable,



© Till Hofmann, Stefan Schupp, and Gerhard Lakemeyer;
licensed under Creative Commons License CC-BY 4.0

33rd International Symposium on Temporal Representation and Reasoning (TIME 2026).

Editors: AndreA Orlandini and Sophie Pinchinat; Article No. 11; pp. 11:1–11:22

OpenAccess Series in Informatics



OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

even when the domain contains only finitely many objects. To recover decidability, we adapt the core techniques of timed automata theory to the situation calculus: we introduce clocks as real-valued fluents that may only be reset or advanced uniformly and compared against natural numbers, and we show that region abstraction, adapted to situations as a time-abstract bisimulation, yields a finite abstraction of the otherwise infinite tree of situations. This transfer lets us decide reachability for clocked basic action theories over finite domains. We then extend this result to GOLOG programs and investigate *program realization*. Given a GOLOG program, we present a decidable method for determining a corresponding action sequence that results in a final program configuration.

Reachability and program realization in this restricted setting are closely related to recent work on synthesis for GOLOG programs. Reactive synthesis has been studied for the propositional situation calculus [10] and for first-order agent programs against uncontrollable environments with LTL_f goals [18]. Our realization procedure complements these: rather than synthesizing a strategy against an adversarial environment, we focus on the fully controllable setting, but extend the formalism with continuous time.

This paper is organized as follows: We discuss related work in Section 2 and present the foundations of the SC in Section 3. We discuss reachability in Section 4, where we show undecidability for the existing approaches and then show a decidable approach in Section 5. In Section 6, we apply these results to program realization and conclude in Section 7.

2 Related Work

Already McCarthy [21] proposed a fluent function *time* that gives the value of time in a situation. Elaborating on this idea, Gelfond et al. [16] described an extension where time is a fluent with values from the integers or real numbers and where each action has a duration. Miller [22] attached time to each action and allowed overlapping actions with durations, along with a partial ordering of actions that allows describing two concurrent sequences of actions. Pinto and Reiter [26] modeled durative actions with instantaneous *start* and *end* actions and continuous time to describe actions and events. By distinguishing actual and possible situations, they define a total ordering on situations that allows expressing linear-time temporal properties within the SC. Finzi and Pirri [15] combine the SC with temporal constraint reasoning, among others supporting *processes* that have start and end actions and explicit start times. Several approaches [22, 28] allow modeling continuous processes, where a fluent continuously changes its value between two situations. The *hybrid situation calculus* [5] combines the SC with hybrid systems by embedding *hybrid automata* [2].

The *event calculus* [20, 24] uses an explicit representation of discrete or continuous time, where *events* occur at certain points on the timeline. In contrast to the SC, formulas do not refer to situations, but to points in time, and it uses circumscription to solve the frame problem. A comparison of the two formalisms can be found in [30]. As there is no action programming language such as GOLOG in the event calculus, we focus on the SC.

Timed automata (TAs) [3] extend finite automata with continuous time. While reachability is decidable for the original model, several extensions render it undecidable [4]. For TAs with *diagonal* constraints of the form $x - y \bowtie c$ (in contrast to $x \bowtie c$), reachability remains decidable [3], while it is undecidable with *additive constraints* ($x + y \bowtie c$) [7]. Allowing clock resets to non-zero values also affects decidability: with diagonal-free constraints, resets of clocks such as $x := x + 1$ preserve decidability, while diagonal constraints render reachability undecidable. Updates of the form $x := x - 1$ always yield undecidability [6]. As decidability for timed automata is well-studied, it is reasonable to adopt this model of time for the SC.

3 The Situation Calculus

The situation calculus (SC) [21, 29] is a sorted predicate logic for reasoning about dynamically changing worlds. All changes in the world are caused by *actions*, which are terms in the language. A *situation* is a possible history of the world, where the distinguished function symbol $\text{do}(\alpha, \sigma)$ describes the resulting situation of doing action α in situation σ and the constant S_0 denotes the initial situation where no actions have yet occurred. Besides actions and situations, the sort *objects* is used for terms describing objects of the world. *Fluents* are relational or functional symbols that describe possibly changing properties of the world and that have a situation as their last argument. Relational and functional symbols that remain unchanged between situations are called *rigid* and do not have a situation argument. We denote relational symbols with upper-case letters, e.g., $\text{IsFull}(p, s)$, and functional symbols with lower-case letters, e.g., $\text{counter}(s)$. Apart from the situation argument, all arguments of a fluent are of type object. We assume that there is a finite set of fluent symbols $\mathcal{F}$ and a finite number of action types. In contrast to the usual definition, we also assume a *finite set of objects* $\mathcal{O}$, for which we require the unique name assumption and domain closure. For an action type A , we write $|A|$ for the number of arguments of A . In the following, s denotes a situation variable, σ a ground situation term, a an action variable, α a ground action term, A an action type, p and o object variables, ρ a ground object term, and $\vec{x}$ a tuple of variables $x_1, \dots, x_k$ of the corresponding sort (similarly for terms). We also write $\phi_1 \supset \phi_2$ for $\neg\phi_1 \vee \phi_2$ and $\phi_1 \equiv \phi_2$ for $\phi_1 \supset \phi_2 \wedge \phi_2 \supset \phi_1$, where $\equiv$ has lowest operator precedence and free variables are assumed to be universally quantified from the outside, e.g., $F(\vec{o}, s) \equiv \phi_1 \vee \phi_2$ stands for $\forall \vec{o}, s. (F(\vec{o}, s) \equiv (\phi_1 \vee \phi_2))$. Finally, we write $\text{do}([\alpha_1; \dots; \alpha_k], S_0)$ for $\text{do}(\alpha_k, \text{do}(\dots, \text{do}(\alpha_1, S_0)))$. A formula is *uniform in s* if it does not mention any situation terms other than s and does not mention Poss (see [29] for a formal definition). A *situation-suppressed formula* is a formula where the situation argument of each fluent is omitted. For a formula ϕ and a situation term σ , we write $\phi[\sigma]$ for the formula obtained by replacing each occurrence of s with σ . Similarly, for a situation-suppressed formula ϕ , $\phi[\sigma]$ is like ϕ with every fluent augmented by the situation argument σ .

The SC has been extended to represent time by adding an additional real-valued sort *time* with the standard interpretation of the real numbers and their operations (addition, subtraction, etc.) and relations ($<$, $\leq$, etc.) and augmenting each action term $A(\vec{o}, t)$ with an additional time argument t , which specifies the time point when the action occurs [26, 29]. An additional function symbol $\text{time}(\alpha)$ is used to describe the starting time of the action situation, i.e., $\text{time}(A(\vec{o}, t)) := t$. This allows axiomatizing the starting time of a situation with $\text{start}(\text{do}(a, s)) = \text{time}(a)$ and *processes* with a duration that is defined by its start and end action [15]. With real numbers as an additional sort and independent of whether actions have an execution timepoint, we may also allow *real-valued fluent functions*. As we will see in Section 4, having either durative processes or real-valued fluent functions without any constraints renders the reachability problem undecidable.

Basic Action Theories

A basic action theory (BAT) [27] is an axiomatization of a domain and describes all available actions with precondition axioms and successor state axioms. In addition to the domain-specific axioms, a BAT also contains the foundational axioms of the SC. Similar to [13], we include domain closure axioms for action types and objects. Here, we axiomatize a finite set of objects.

► **Definition 1.** A basic action theory (BAT) is a set of sentences $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_{\text{poss}} \cup \mathcal{D}_{\text{ssa}} \cup \mathcal{D}_{\text{ca}} \cup \mathcal{D}_{\text{co}} \cup \Sigma$ where:

- $\mathcal{D}_0$ is a description of the initial situation with a set of sentences uniform in S_0 . Here, we assume complete information, i.e., for every atomic (fluent or rigid) formula ϕ uniform in S_0 , either $\mathcal{D} \models \phi$ or $\mathcal{D} \models \neg\phi$.
- $\mathcal{D}_{\text{poss}}$ are precondition axioms of the form $\text{Poss}(A(\vec{o}), s) \equiv \phi_A(\vec{o}, s)$, one per action type $A \in \mathcal{A}$ and where $\phi_A(\vec{o}, s)$ is a formula uniform in s that describes when the action $A(\vec{o})$ can be performed.
- $\mathcal{D}_{\text{ssa}}$ are successor state axioms (SSAs) describing action effects. For each relational fluent $F \in \mathcal{F}$, $\mathcal{D}_{\text{ssa}}$ contains an axiom of the form $F(\vec{o}, \text{do}(a, s)) \equiv \phi_F(\vec{o}, a, s)$, where $\phi_F(\vec{o}, a, s)$ is a formula uniform in s that describes how the fluent $F(\vec{o})$ changes if some action a is performed. For each functional fluent $f \in \mathcal{F}$, $\mathcal{D}_{\text{ssa}}$ contains an axiom of the form $f(\vec{o}, \text{do}(a, s)) = y \equiv \phi_f(\vec{o}, y, a, s)$, where $\phi_f(\vec{o}, y, a, s)$ is a formula uniform in s describing how the fluent $f(\vec{o})$ changes if an action a is performed.
- $\mathcal{D}_{\text{ca}}$ are domain closure and unique name axioms (UNAs) on actions $\mathcal{A}$, e.g.,

$$\begin{aligned} & \forall a (\exists \vec{o}_1 (a = A_1(\vec{o}_1)) \vee \dots \vee \exists \vec{o}_n (a = A_n(\vec{o}_n))) \\ & \forall \vec{o}_i \vec{o}_j (A_i(\vec{o}_i) \neq A_j(\vec{o}_j)) \text{ for all } i \neq j \\ & \forall \vec{o}_1 \vec{o}'_1 (A_1(\vec{o}_1) = A_1(\vec{o}'_1) \supset \vec{o}_1 = \vec{o}'_1) \wedge \dots \wedge \forall \vec{o}_n \vec{o}'_n (A_n(\vec{o}_n) = A_n(\vec{o}'_n) \supset \vec{o}_n = \vec{o}'_n) \end{aligned}$$

- $\mathcal{D}_{\text{co}}$ are domain closure and UNAs on object constants $\mathcal{O}$, e.g., $\forall o (o = \rho_1 \vee o = \rho_2 \vee \dots) \wedge \rho_1 \neq \rho_2 \wedge \dots$.
- Σ : foundational, domain-independent axioms, possibly including time and start from above and including the relation $\sqsubset$ that provides an ordering relation on situations.

We call $\mathcal{D}$ a *timed BAT* if it includes axioms for time, start, etc. from above and we say $\mathcal{D}$ is *real-valued* if it includes a functional fluent of sort *time*. Note that these are two independent properties: We may have occurrence times for actions without having any real-valued fluents and we may use real-valued fluents without assigning any time values to actions. In the following, we assume that situation terms only mention action types from $\mathcal{A}$ and object terms from $\mathcal{O}$. Following [29], a situation s is *executable*, written $\text{Exec}(s)$, if every action performed to reach s is possible in the situation where it occurred: $\text{Exec}(s) := \text{do}(a, s') \sqsubseteq s \supset \text{Poss}(a, s')$.

4 Reachability

Generally speaking, reachability asks whether there is a reachable situation where some property holds. Formally, given a BAT $\mathcal{D}$ and a formula ϕ , ϕ is *reachable* if there is an executable situation where ϕ holds, i.e., if $\mathcal{D} \models \exists s \text{Exec}(s) \wedge \phi[s]$. Many problems such as planning or verification of safety properties can be formulated as a reachability problem. It is therefore desirable to have a sound and complete method to decide reachability. However, depending on the expressivity of the underlying framework, the reachability problem may become undecidable. It is easy to see that for a given BAT over a finite set of objects that does not make use of time, the reachability problem is decidable. On the other hand, if we include fluents of sort *time* and allow arbitrary arithmetic operations on those fluents, then the problem is no longer decidable:¹

¹ Full proofs can be found in Appendix A.

► **Theorem 2.** *The reachability problem for a real-valued BAT is undecidable.*

Proof Idea. By reduction of the halting problem for 2-counter machines (2CMs), which is undecidable [23]. Given a 2CM $\mathcal{M}$, we can construct a BAT $\mathcal{D}_{\mathcal{M}}$ such that $\mathcal{D}_{\mathcal{M}} \models \exists s. \text{Exec}(s) \wedge \text{Halt}(s)$ iff $\mathcal{M}$ halts. ◀

Intuitively, the problem is that we may use time fluents as counters for 2CMs. These counters may be unbounded because time fluents are not bounded. A natural restriction is thus adding an upper bound b to all fluents of sort *time*: Given a bound $b \in \mathbb{Q}$ and a BAT $\mathcal{D}$ with $\mathcal{F}_{\text{time}}$ denoting all functional fluents of sort time, we express that all fluents are bounded by b with $\text{Bounded}_b(s) := \forall s'. s' \sqsubseteq s \supset \bigwedge_{f \in \mathcal{F}_{\text{time}}} \forall \vec{o}. f(\vec{o}, s') \leq b$. We can now define the *time-bounded reachability problem*: Given a property ϕ and a bound $b \in \mathbb{Q}$, does $\mathcal{D} \models \exists s \text{Exec}(s) \wedge \text{Bounded}_b(s) \wedge \phi[s]$ hold? Unfortunately, this restriction is not sufficient to accomplish decidability:

► **Theorem 3.** *The time-bounded reachability problem for a real-valued BAT and a time bound $b \geq 2$ is undecidable.*

Proof Idea. Again by reduction of the halting problem for 2CMs. A counter value u is encoded by a fluent value 2^{1-u} . A counter is decremented by doubling and incremented by halving the corresponding fluent and tested for zero by comparing the corresponding fluent to 2. ◀

These results suggest that we need to restrict the arithmetic operators on time fluents such that time may not change its value arbitrarily, but is monotonically increasing. However, this is not sufficient; even if we do not allow real-valued fluents but instead only consider actions with time arguments and processes with start times [15], the reachability problem remains undecidable:

► **Theorem 4.** *The reachability problem for timed BATs without real-valued functional fluents is undecidable.*

Proof Idea. By reduction of the language emptiness problem for TAs with 4 clocks $c_1, \dots, c_4$ extended with additive clock constraints of the form $c_1 + c_2 = 1$, which is undecidable [7]. For a TA $\mathcal{A}$, we can construct a BAT and a formula ϕ such that $\mathcal{D} \models \exists s. \text{Exec}(s) \wedge \phi[s]$ iff the language of $\mathcal{A}$ is non-empty. The key idea is to model each clock as a separate process, where a clock reset starts the process, defining the clock value of every clock in every situation. ◀

Together, Theorems 2 to 4 rule out the natural ways of modeling dense time in the SC while retaining decidability. Reachability is undecidable if fluents range over the reals and may be manipulated arithmetically (Theorem 2); bounding the range of these fluents does not help (Theorem 3); and removing real-valued functional fluents altogether, such that time only occurs through timed actions, does not retain decidability either (Theorem 4). In each case, the problem is not continuous time itself, but allowing time values to be combined arithmetically. This motivates the model of Section 5, which keeps dense time but restricts the admissible operations by only allowing resets, comparisons against natural numbers, and uniform progression of time.

5 Clocks in the Situation Calculus

In this section, we propose an alternative approach for modeling time in the SC that maintains decidability. We adopt methods from timed automata theory by defining *clocked BATs* and show that the reachability problem for clocked BATs is decidable. In particular, we introduce

clocks as real-valued functional fluents with restricted SSAs that guarantee uniform passing of time and that allow an action to reset a clock to zero. Furthermore, we require that the BAT uses a distinguished action *wait* that causes time to pass. Apart from *wait*, all actions are instantaneous, i.e., they do not have a time argument and there is no delay between the previous and the current action. This allows using clocks both in action preconditions and in successor state axioms. We restrict formulas mentioning clocks to comparisons to natural numbers:

► **Definition 5.** We distinguish the following types of formulas:

1. A formula ϕ is a clock comparison if it is of the form $f(\vec{o}, s) \bowtie v$ or $v \bowtie v'$, where $f \in \mathcal{F}$ is some functional fluent, $v, v' \in \mathbb{N}$, and $\bowtie \in \{<, \leq, =, \geq, >\}$.
2. A formula ϕ is called clocked if every atomic sub-formula mentioning a term of sort *time* is a clock comparison.
3. A formula ϕ is called time-independent if it does not mention any term of sort *time*.

We call a clock comparison $\mathcal{K}$ -bounded if it only compares against constants $v, v' \leq \mathcal{K}$. A clocked formula is $\mathcal{K}$ -bounded if each occurring clock comparison is $\mathcal{K}$ -bounded. As we will see, these restrictions allow us to apply *regionalization*, which defines an equivalence relation on clock values. Based on regionalization, we will define a *time-abstract bisimulation*, resulting in a *finite* abstraction of the otherwise infinite tree of situations, which permits a decidable procedure for the reachability problem. We start by introducing clocks to BATs:

► **Definition 6.** A BAT $\mathcal{D}$ is called a clocked BAT if:

1. Every functional fluent f is real-valued, i.e., of sort *time*;
2. For each functional fluent f , $\mathcal{D}_0$ contains an axiom $f(\vec{o}, S_0) = 0$;
3. For each functional fluent f , $\mathcal{D}_{ssa}$ contains an axiom of the form

$$f(\vec{o}, \text{do}(a, s)) = y \equiv \exists t. a = \text{wait}(t) \wedge y = f(\vec{o}, s) + t \\ \vee \neg \exists t. a = \text{wait}(t) \wedge (\phi_f(\vec{o}, a, s) \wedge y = 0 \vee \neg \phi_f(\vec{o}, a, s) \wedge y = f(\vec{o}, s))$$

where $\phi_f(\vec{o}, a, s)$ is a time-independent formula uniform in s describing if the action a resets the value of $f(\vec{o})$;

4. For every relational fluent R , $\mathcal{D}_{ssa}$ contains an axiom of the form $R(\vec{o}, \text{do}(a, s)) \equiv \phi_R(\vec{o}, a, s)$ where $\phi_R(\vec{o}, a, s)$ is a clocked formula uniform in s .
5. Every precondition axiom in $\mathcal{D}_{poss}$ is of the form $\text{Poss}(A(\vec{o}), s) \equiv \phi_A(\vec{o}, s)$ such that $\phi_A(\vec{o}, s)$ is a clocked formula uniform in s ;
6. Waiting for some non-negative amount of time is always possible: $\text{Poss}(\text{wait}(t), s) \equiv t \geq 0$;

We call a clocked BAT $\mathcal{D}$ $\mathcal{K}$ -bounded if each clocked formula occurring in $\mathcal{D}$ is $\mathcal{K}$ -bounded. Note that $\mathcal{D}$ is always $\mathcal{K}$ -bounded for some $\mathcal{K} \in \mathbb{N}$, because $\mathcal{D}$ consists of finitely many axioms. For the sake of simplicity, we assume that all functional fluents are of domain *time* and are initialized to zero. While this is not strictly necessary, it simplifies the treatment of functional fluents later on. The restricted form of the SSAs of functional fluents enforces a behavior similar to clocks in timed automata: An action *wait*(t) increments each clock by the value $t \in \mathbb{R}_{\geq 0}$; all other actions either leave the value unchanged or reset it to zero.² Hence, we will also call these fluents *clocks*. We write $\mathcal{C}$ for the set of all ground situation-suppressed clock terms, we denote clock variables with $c, c_1, \dots$ and (possibly situation-suppressed) ground

² This separation allows us to use clock comparisons in precondition axioms and SSAs. If we allowed time to pass with every action, then the precondition axiom would refer to the time of the *previous* situation, as time is only incremented after the action has occurred.

clock terms with $\omega, \omega_1, \dots$. Finally, we also restrict the usage of clocks in all axioms of a clocked BAT: Both precondition and successor state axioms may only use clocked formulas, which restrict any operations on clocks to clock comparisons. As we will see later on, this is an essential restriction for allowing regionalization. Nevertheless, it is easy to show that clocked BATs are expressive enough to represent TAs. We use a variant of the coffee delivery robot [29] as a running example:

► **Example 7.** We model a coffee delivery service that has the durative actions *brew* and *pour* and objects *Pot*, *Mug₁*, and *Mug₂*. The robot needs to brew coffee and then pour it into the mugs. The fluents *Brew* and *Pour* indicate that the robot is currently brewing (respectively pouring) coffee. After brewing, the pot *IsFull* and after pouring into a mug, the mug *IsFull*. The clock c_{brew} tracks the duration of *brew*, while c_{glob} tracks global time. Brewing takes at least 1 time unit. If the robot brews coffee for at least 2 time units, the coffee is *Strong*.

- Initially, the robot is doing nothing, nothing is filled, the coffee is not strong, and all clocks are initialized to zero. Also, only *Mug₁* should be filled with strong coffee:

$$\neg Brew(p, S_0), \neg Pour(p, m, S_0), \neg IsFull(o, S_0), \neg Strong(o, S_0), \\ c_{brew}(p, S_0) = 0, c_{glob}(S_0) = 0, WantStrong(o) \equiv o = Mug_1$$

- The robot may start a durative action if it is not doing anything and end it if it has started it before. Brewing takes at least 1 time unit. In order to pour the coffee, there must be coffee in the pot:

$$\begin{aligned} Poss(sBrew(p), s) &\equiv \neg Brew(p, s) \wedge \neg \exists m. Pour(p, m, s) \\ Poss(eBrew(p), s) &\equiv Brew(p, s) \wedge c_{brew}(p, s) \geq 1 \\ Poss(sPour(p, m), s) &\equiv IsFull(p, s) \wedge \neg \exists p', m'. Pour(p', m', s) \wedge \neg Brew(p, s) \\ Poss(ePour(p, m), s) &\equiv Pour(p, m, s) \\ Poss(wait(t), s) &\equiv t \geq 0 \end{aligned}$$

- The SSAs state that *brew* fills a pot with coffee and makes it strong if it took at least 2 time units:

$$\begin{aligned} IsFull(o, do(a, s)) &\equiv a = eBrew(o) \vee \exists p. a = ePour(p, o) \vee IsFull(o, s) \\ Strong(o, do(a, s)) &\equiv a = eBrew(o) \wedge c_{brew}(o, s) \geq 2 \\ &\quad \vee \exists p. a = ePour(p, o) \wedge Strong(p, s) \vee Strong(o, s) \\ Brew(p, do(a, s)) &\equiv a = sBrew(p) \vee Brew(p, s) \wedge a \neq eBrew(p) \\ Pour(p, m, do(a, s)) &\equiv a = sPour(p, m) \vee Pour(p, m, s) \wedge a \neq ePour(p, m) \\ c_{brew}(p, do(a, s)) = y &\equiv \exists t(a = wait(t) \wedge y = c_{brew}(p, s) + t) \vee \left(\neg \exists t. a = wait(t) \right. \\ &\quad \left. \wedge (a = sBrew(p) \wedge y = 0 \vee a \neq sBrew(p) \wedge y = c_{brew}(p, s)) \right) \\ c_{glob}(do(a, s)) = y &\equiv \exists t. a = wait(t) \wedge y = c_{glob}(s) + t \\ &\quad \vee \neg \exists t. a = wait(t) \wedge y = c_{glob}(s) \end{aligned}$$

- There are no other objects or actions:

$$\begin{aligned} \forall o(o = Pot \vee o = Mug_1 \vee \dots) \wedge Mug_1 \neq Pot \wedge \dots \\ \forall a(\exists o. a = sBrew(o) \vee \dots \vee \exists t. a = wait(t)) \\ \wedge \forall o_1, o_2. sBrew(o_1) \neq eBrew(o_2) \wedge \dots \\ \wedge \forall o_1, o_2. sBrew(o_1) = sBrew(o_2) \supset o_1 = o_2 \wedge \dots \end{aligned}$$

5.1 Regression

A fundamental reasoning task in the SC is *projection*, i.e., determining whether a sentence ϕ is true in a future situation. A common method to solve projection is *regression* [27], which reduces the formula ϕ to an equivalent formula $\mathcal{R}[\phi]$ which only refers to the initial situation. Given a BAT, regression substitutes each fluent mentioned in ϕ by the right-hand side of the corresponding SSA until every fluent only mentions S_0 . Regression is sound and complete and thus may be used for reducing queries about a future situation to the initial situation. Here, we use a simplified variant of regression, where we require that all action terms are ground:

► **Definition 8.** A formula ϕ is regressiveable iff

1. ϕ is first order (i.e., the second-order axioms Σ are not regressiveable).
2. Every term of sort situation mentioned in ϕ has the form $\text{do}([\alpha_1, \dots, \alpha_n], S_0)$, where $\alpha_1, \dots, \alpha_n$ are ground terms of sort action.
3. Every atom mentioning a clock f is of the form $f(\vec{o}, s) \bowtie \tau$, where $\vec{o}$ are terms of sort object and $\tau \in \mathbb{R}$.³
4. For every atom of the form $\text{Poss}(a, s)$ mentioned in ϕ , a has the form $A(\vec{o})$ for some action type A .
5. ϕ does not quantify over situations.
6. ϕ does not mention the symbol $\sqsubset$, nor does it mention any equality atom $s = s'$ for situation terms s, s' .

We extend the standard regression operator $\mathcal{R}$ with a regression rule for clock comparisons:

► **Definition 9.** Let $\phi = f(\vec{\rho}, \text{do}(\alpha, \sigma)) \bowtie r$ where $f \in \mathcal{F}$ is a clock and $r \in \mathbb{R}$. By Definition 6, there is an SSA of the following form:

$$\begin{aligned} f(\vec{o}, \text{do}(a, s)) = y &\equiv \exists t. a = \text{wait}(t) \wedge y = f(\vec{o}, s) + t \\ &\vee \neg \exists t. a = \text{wait}(t) \wedge (\phi_f(\vec{o}, a, s) \wedge y = 0 \vee \neg \phi_f(\vec{o}, a, s) \wedge y = f(\vec{o}, s)) \end{aligned}$$

Then $\mathcal{R}[\phi]$ is defined as follows:

1. If $\alpha = \text{wait}(t)$ for some $t \in \mathbb{R}_{\geq 0}$, then $\mathcal{R}[\phi] := \mathcal{R}[f(\vec{\rho}, \sigma) \bowtie r']$ with $r' = r - t$.
2. Otherwise: $\mathcal{R}[\phi] := \mathcal{R}[\phi_f(\vec{\rho}, \alpha, \sigma) \wedge 0 \bowtie r \vee \neg \phi_f(\vec{\rho}, \alpha, \sigma) \wedge f(\vec{\rho}, \sigma) \bowtie r]$

We obtain a variant of the regression theorem:

► **Theorem 10.** Let ϕ be a regressiveable formula. Then $\mathcal{R}[\phi]$ is uniform in S_0 and $\mathcal{D} \models \phi$ iff $\mathcal{D}_0 \cup \mathcal{D}_c \models \mathcal{R}[\phi]$.

With this variant of regression, deciding whether a BAT entails a regressiveable clocked formula is decidable:

► **Theorem 11.** Let ϕ be a regressiveable clocked formula and $\mathcal{D}$ a clocked BAT (over finite objects and with complete information). Then $\mathcal{D} \models \phi$ is decidable.

Proof. As regression is sound and complete, it is sufficient to consider $\mathcal{D}_0 \cup \mathcal{D}_c \models \mathcal{R}[\phi]$, which can be reduced to entailment in propositional logic: First, as $\mathcal{O}$ is finite, we can eliminate each quantifier in $\mathcal{D}$ and ϕ by substitution, e.g., by substituting $\exists o. \phi'$ with $\bigvee_{o \in \mathcal{O}} \phi'$. Second, note that regressing a clock comparison $f(\vec{\rho}, \sigma) \bowtie r$ results in a formula $f(\vec{\rho}, S_0) \bowtie r'$, where

³ In particular, we disallow quantifying over terms of sort *time*.

$r' \in \mathbb{R}$. As all clock values are initialized to 0, we can directly replace each such comparison by $\top$ or $\perp$, e.g., $f(\vec{\rho}, S_0) > 5$ is simplified to $\perp$. Finally, with the unique name assumption, we can interpret predicate symbols as propositional variables, thereby obtaining a propositional set of sentences $\mathcal{D}'_0$ and a propositional formula ϕ' such that $\mathcal{D}'_0 \models \phi'$ iff $\mathcal{D}_0 \cup \mathcal{D}_c \models \phi$. $\blacktriangleleft$

5.2 Regionalization

To show decidability of the reachability problem for clocked BATs, we use an approach based on abstraction for timed automata inspired by [3, 25]. We construct a finite abstraction of the otherwise infinite tree of situations of a clocked BAT $\mathcal{D}$ using a time-abstract bisimulation relation over situations:

► **Definition 12** (Time-abstract Bisimulation). *An equivalence relation $R \subseteq \mathcal{S} \times \mathcal{S}$ is a $\mathcal{K}$ -relative time-abstract bisimulation on $\mathcal{S}$ if $(\sigma_1, \sigma_2) \in R$ implies*

1. *For every $\mathcal{K}$ -bounded clocked formula ϕ uniform in s : $\mathcal{D} \models \phi[\sigma_1]$ iff $\mathcal{D} \models \phi[\sigma_2]$.*
2. *For every action type $A \neq \text{wait}$ and ground arguments $\vec{\rho}$, $(\text{do}(A(\vec{\rho}), \sigma_1), \text{do}(A(\vec{\rho}), \sigma_2)) \in R$.*
3. *For every $\tau_1 \in \mathbb{R}_{\geq 0}$, there is a $\tau_2 \in \mathbb{R}_{\geq 0}$ such that $(\text{do}(\text{wait}(\tau_1), \sigma_1), \text{do}(\text{wait}(\tau_2), \sigma_2)) \in R$.*

Intuitively, two situations are bisimilar if they represent the same state of the world (i.e., they satisfy the same $\mathcal{K}$ -bounded clocked formulas) and if applying the same action in both situations results again in bisimilar situations.

In the following, we construct such a bisimulation by applying regionalization to situations. Generally, a *clock valuation over clocks C* is a function $\nu : C \rightarrow \mathbb{R}_{\geq 0}$. For situations, we can define clock valuations as follows:

► **Definition 13** (Clock Valuation). *Let $\mathcal{D}$ be a clocked BAT with (situation-suppressed) clocks $\mathcal{C}$ and σ a situation. The clock valuation $\nu_\sigma^\mathcal{D}$ of σ is the clock valuation over $\mathcal{C}$ with $\nu_\sigma^\mathcal{D}(\omega) = \tau$ iff $\mathcal{D} \models \omega[\sigma] = \tau$ for a ground clock term ω .*

We will omit the superscript $\mathcal{D}$ if $\mathcal{D}$ is clear from context. It is sometimes convenient to write a clock valuation ν as a set of pairs $\{(c, v) \mid c \in C, v = \nu(c)\} \subseteq C \times \mathbb{R}_{\geq 0}$. We also write $\nu + \tau$ for $\{(c, v + \tau) \mid c \in C, v = \nu(c)\}$ where all clocks are incremented by τ . We can show that every situation has a unique clock valuation:

► **Lemma 14.** *If $\mathcal{D}$ is a clocked BAT and σ a situation, then ν_σ always exists and is unique.*

We use *regionalization* to obtain an abstraction of clock valuations. We briefly outline the intuition behind this idea: Given a constant $\mathcal{K} \in \mathbb{N}$ that is the maximal constant of sort *time* mentioned in $\mathcal{D}$ and in the reachability query ϕ , we can collect all clock values with a value larger than $\mathcal{K}$ in a single class $\top$. Furthermore, as clocks are only compared to natural numbers, for a single clock it suffices to observe only integer parts and whether fractional parts differ from zero to derive a finite set of equivalence classes for clock valuations $v \in [0, \mathcal{K}]$. In the presence of multiple clocks, the order between the fractional parts needs to be taken into account as well [1]. We define the *fractional part* $\text{fract}(v)$ of $v \in [0, \mathcal{K}] \cup \{\top\}$ as $v - \lfloor v \rfloor$ if $v \in [0, \mathcal{K}]$ and zero otherwise. This leads to the definition of clock regions:

► **Definition 15** (Clock Regions). *Given a maximal constant $\mathcal{K}$, let $V = [0, \mathcal{K}] \cup \{\top\}$. We define the region equivalence as the equivalence relation $\sim_{\mathcal{K}}$ on V such that $u \sim_{\mathcal{K}} v$ if*

- $u = v = \top$, or
- $u, v \neq \top$, $\lceil u \rceil = \lceil v \rceil$, and $\lfloor u \rfloor = \lfloor v \rfloor$.

A region is an equivalence class of $\sim_K$. We extend region equivalence to clock valuations. Let ν, ν' be two clock valuations over $\mathcal{C}$, then ν and ν' are region-equivalent, written $\nu \cong_K \nu'$ iff (1) for every $c \in \mathcal{C}$, $\nu(c) \sim_K \nu'(c)$, and (2) for every $c, c' \in \mathcal{C}$, $\text{fract}(\nu(c)) \leq \text{fract}(\nu'(c'))$ iff $\text{fract}(\nu'(c)) \leq \text{fract}(\nu(c'))$.

Note that for a fixed K , there is only a finite number of regions. Based on region equivalence wrt. K , we can now define an equivalence relation on situations:

- **Definition 16.** Let $\approx_{DK} \subseteq \mathcal{S} \times \mathcal{S}$ be an equivalence relation such that $\sigma_1 \approx_{DK} \sigma_2$ iff
1. for every fluent $R \in \mathcal{F}$, $\mathcal{D} \models R(\vec{\rho}, \sigma_1)$ iff $\mathcal{D} \models R(\vec{\rho}, \sigma_2)$,
 2. there is a bijection $f : \nu_{\sigma_1} \rightarrow \nu_{\sigma_2}$ such that
 - a. if $f(c_1, v_1) = (c_2, v_2)$, then $c_1 = c_2$ and $v_1 \sim_K v_2$
 - b. if $f(c_1, v_1) = (c_2, v_2)$ and $f(c'_1, v'_1) = (c'_2, v'_2)$, then $\text{fract}(v_1) \leq \text{fract}(v'_1)$ iff $\text{fract}(v_2) \leq \text{fract}(v'_2)$.

Two situations are equivalent if they satisfy the same fluents and agree on the clock regions. For a situation σ , we write $[\sigma]_{\approx_{DK}}$ for the equivalence class of σ wrt. $\approx_{DK}$. We denote the set of equivalence classes resulting from $\approx_{DK}$ as $\mathcal{S}/\approx_{DK}$. Equivalence directly extends to clocked formulas:

- **Lemma 17.** Let $\sigma_1 \approx_{DK} \sigma_2$ and let ϕ be a K -bounded clocked formula uniform in s . Then $\mathcal{D} \models \phi[\sigma_1]$ iff $\mathcal{D} \models \phi[\sigma_2]$.

We use a well-known result from [3] which states that it is always possible to maintain equivalence wrt. $\cong_K$ between clock valuations for a given time increment:

- **Proposition 18.** Let ν_1 and ν_2 be two clock valuations over a set of clocks $\mathcal{C}$ such that $\nu_1 \cong_K \nu_2$. Then for all $\tau \in \mathbb{R}_{\geq 0}$ there exists a $\tau' \in \mathbb{R}_{\geq 0}$ such that $\nu_1 + \tau \cong_K \nu_2 + \tau'$.

Therefore, for two situations with $\sigma_1 \approx_{DK} \sigma_2$ and any action $\text{wait}(\tau_1)$, it is always possible to find a τ_2 such that $\text{do}(\text{wait}(\tau_1), \sigma_1) \approx_{DK} \text{do}(\text{wait}(\tau_2), \sigma_2)$. Based on this, we can show that $\approx_{DK}$ satisfies all criteria of Definition 12:

- **Theorem 19.** $\approx_{DK}$ is a time-abstract bisimulation.

Hence, it suffices to argue about the resulting equivalence classes to determine entailment with respect to a given $\mathcal{D}$. This enables us to construct a time-abstract transition system as a finite abstraction for the otherwise infinite tree of situations, where states collect $\approx_{DK}$ -equivalent situations connected by executable actions from $\mathcal{D}$:

- **Definition 20 (Time-abstract Transition System).** Let $\mathcal{D}$ be a clocked BAT. We define the TS $\mathcal{T}_{\mathcal{D}} = (Q, q_0, \rightarrow)$ as follows:

- The set of states Q consists of equivalence classes of $\approx_{DK}$, i.e., $Q \subseteq \mathcal{S}/\approx_{DK}$;
- The initial state is the equivalence class $q_0 = [S_0]_{\approx_{DK}}$;
- $[\sigma]_{\approx_{DK}} \rightarrow [\sigma']_{\approx_{DK}}$ if σ' is an executable successor situation of σ , i.e., if $\sigma' = \text{do}(\alpha, \sigma)$ and $\mathcal{D} \models \text{Exec}(\sigma')$.

It is easy to see that it is sufficient to consider $\mathcal{T}_{\mathcal{D}}$ for checking reachable situations:

- **Lemma 21.** $[\sigma]_{\approx_{DK}}$ is reachable in $\mathcal{T}_{\mathcal{D}}$ iff there is some $\sigma' \in [\sigma]_{\approx_{DK}}$ with $\mathcal{D} \models \text{Exec}(\sigma')$.

In contrast to the tree of situations, the time-abstract transition system $\mathcal{T}_{\mathcal{D}}$ is finite:

- **Lemma 22.** The time-abstract transition system $\mathcal{T}_{\mathcal{D}}$ has finitely many states.

Hence, the time-abstract transition system $\mathcal{T}_{\mathcal{D}}$ is useful for deciding reachability: every reachable situation is also reachable in $\mathcal{T}_{\mathcal{D}}$, which is a finite transition system and can be explored exhaustively. It remains to be shown that we can actually construct $\mathcal{T}_{\mathcal{D}}$.

■ **Algorithm 1** Computation of the time-abstract transition system $\mathcal{T}_{\mathcal{D}}$ for a clocked BAT $\mathcal{D}$.

```

 $Open \leftarrow \{S_0\}; V \leftarrow \{[S_0]_{\approx_{\mathcal{D}\mathcal{K}}}\}; E \leftarrow \emptyset$ 
while  $Open \neq \emptyset$  do
   $\sigma \leftarrow \text{POP}(Open)$ 
   $Acts \leftarrow \{A(\vec{\rho}) \mid A \in \mathcal{A} \setminus \{wait\}, \vec{\rho} \in \mathcal{O}^{|\mathcal{A}|}\} \cup \{wait(\tau) \mid \tau \in \text{TSuccs}(\sigma, \mathcal{K})\}$ 
  for all  $\alpha \in Acts$  do
    if  $\mathcal{D}_0 \cup \mathcal{D}_c \models \mathcal{R}[\text{Poss}(\alpha, \sigma)]$  then
      if  $[\text{do}(\alpha, \sigma)]_{\approx_{\mathcal{D}\mathcal{K}}} \notin V$  then
         $V \leftarrow V \cup \{[\text{do}(\alpha, \sigma)]_{\approx_{\mathcal{D}\mathcal{K}}}\}$ 
         $Open \leftarrow Open \cup \{\text{do}(\alpha, \sigma)\}$ 
       $E \leftarrow E \cup \{([\sigma]_{\approx_{\mathcal{D}\mathcal{K}}}, [\text{do}(\alpha, \sigma)]_{\approx_{\mathcal{D}\mathcal{K}}})\}$ 
return  $(V, [S_0]_{\approx_{\mathcal{D}\mathcal{K}}}, E)$ 

```

■ **Algorithm 2** Time successor computation for a situation σ .

```

function  $\text{TSuccs}(\sigma, K)$ 
   $\nu \leftarrow \nu_{\sigma}; last \leftarrow 0; Succs \leftarrow \{last\}$ 
  while  $\exists(c, v) \in \nu : v \leq K$  do
     $\mu \leftarrow \max_{(c, v) \in \nu} \text{fract}(v)$ 
    if  $\exists(c, v) \in \nu : \text{fract}(v) = 0$  then  $incr \leftarrow \frac{1-\mu}{2}$ 
    else  $incr \leftarrow 1 - \mu$ 
     $last \leftarrow last + incr$ 
     $Succs \leftarrow Succs \cup \{last\}$ 
     $\nu \leftarrow \nu + incr$ 
  return  $Succs$ 

```

5.3 Decidability

We now describe an algorithm for constructing $\mathcal{T}_{\mathcal{D}}$, which is shown in Algorithm 1. Starting with the initial state $[S_0]_{\approx_{\mathcal{D}\mathcal{K}}}$, the algorithm iteratively adds a state $[\sigma]_{\approx_{\mathcal{D}\mathcal{K}}}$ to $\mathcal{T}_{\mathcal{D}}$ whenever σ is reachable by a possible single action from an existing state, i.e., if $\sigma = \text{do}(\alpha, \sigma')$ and $\mathcal{D} \models \text{Poss}(\alpha, \sigma')$ for some state $[\sigma']_{\approx_{\mathcal{D}\mathcal{K}}}$ of $\mathcal{T}_{\mathcal{D}}$. It does so in a breadth-first search manner by keeping a list of *open* states $Open$, which consists of states that need to be expanded. The algorithm computes all possible successors by enumerating all possible ground action terms. For the action *wait*, it includes a canonical timepoint τ as a representative for the equivalence class $[\nu_{\sigma} + \tau]_{\approx_{\mathcal{K}}}$. These canonical time successors $\text{TSuccs}(\sigma, \mathcal{K})$ are computed using Algorithm 2 based on a well-known method [25], which satisfies the following property:

► **Proposition 23.** *Let ν be a clock valuation, $\tau \in \mathbb{R}_{\geq 0}$, and $Succs$ the time successors computed by Algorithm 2. Then there is a $\tau' \in Succs$ such that $\nu + \tau \cong_{\mathcal{K}} \nu + \tau'$.*

The reachability problem is indeed decidable:

► **Lemma 24.** *Let σ_1 and σ_2 be two situations. Then it is decidable whether $\sigma_1 \approx_{\mathcal{D}\mathcal{K}} \sigma_2$.*

► **Lemma 25.** *For a clocked BAT $\mathcal{D}$, Algorithm 1 terminates and returns the time-abstract transition system $\mathcal{T}_{\mathcal{D}}$.*

► **Theorem 26.** *Reachability for clocked BATs is decidable.*

Proof. By Lemma 21, it is sufficient to consider the time-abstract transition system $\mathcal{T}_{\mathcal{D}}$. By Lemma 25, Algorithm 1 terminates and computes $\mathcal{T}_{\mathcal{D}}$. We now show how to use $\mathcal{T}_{\mathcal{D}}$ for deciding the reachability problem of some property ϕ with maximal constant $\leq \mathcal{K}$: By

construction, every state of $\mathcal{T}_{\mathcal{D}}$ is an equivalence class wrt. $\approx_{\mathcal{D}\mathcal{K}}$ of executable situations, of which there are finitely many. By Lemma 17, it is sufficient to consider one situation $\sigma^* \in [\sigma]_{\approx_{\mathcal{D}\mathcal{K}}}$ for each such state. Furthermore, by Theorem 10, we can check whether $\mathcal{D} \models \phi[\sigma^*]$ by regression. If such a state exists, return “reachable”, otherwise return “not reachable”. ◀

► **Example 27.** We can check the following reachability queries:

$$\begin{aligned}\phi_1 &:= \forall o. IsFull(o) \\ \phi_2 &:= \forall o. Strong(o) \equiv WantStrong(o) \\ \phi_3 &:= c_{glob}() \leq 1 \wedge Strong(Mug_1)\end{aligned}$$

A situation satisfying ϕ_1 is reachable, e.g., with

$$\sigma_1 = \text{do}([sBrew(Pot); wait(1); eBrew(Pot); sPour(Pot, Mug_1); ePour(Pot, Mug_1); \\ sPour(Pot, Mug_2); ePour(Pot, Mug_2)], S_0)$$

For ϕ_2 , a different action sequence is required, as the pot may not be filled with strong and weak coffee at the same time. Instead, we brew coffee directly in a mug:

$$\sigma_2 = \text{do}([sBrew(Pot); sBrew(Mug_1); wait(1); eBrew(Pot); wait(2); eBrew(Mug_1); \\ sPour(Pot, Mug_2); wait(1); ePour(Pot, Mug_2)], S_0)$$

Finally, there is no reachable situation that satisfies ϕ_3 , as it would require providing strong coffee after one time unit, while it takes at least two time units to prepare strong coffee.

6 Realizing Golog Programs

In this section, we show how we can utilize reachability for clocked BATs to decide realizability for a GOLOG program Δ . The goal is to check whether the non-determinisms in Δ can be resolved and action timepoints can be determined such that the resulting action sequence is executable and results in a final configuration of the program. Similar to the above, we create a transition system $\mathcal{T}_{\Delta}$ from a given program Δ and check whether a final program state is reachable in $\mathcal{T}_{\Delta}$.

6.1 Golog Transition Semantics

We consider programs in CONGOLOG [11] based on clocked BATs. A *program* is a pair $\Delta = (\mathcal{D}, \delta)$, where $\mathcal{D}$ is a BAT and δ is a program expression built from the following grammar:

$$\delta ::= \alpha \mid \phi? \mid \delta; \delta \mid \delta | \delta \mid \pi x. \delta \mid \delta^* \mid \delta || \delta$$

Program expressions consist of actions α , tests $\phi?$, sequence of actions $\delta_1; \delta_2$, non-deterministic branching $\delta_1 | \delta_2$, non-deterministic choice of argument $\pi x. \delta$, non-deterministic iteration δ^* , and concurrent execution $\delta_1 || \delta_2$. We require that each action term is time-suppressed (i.e., $wait()$ has no argument) and each test $\phi?$ tests a situation-suppressed clocked formula ϕ . We write $\text{nil} := \text{TRUE?}$ for the empty program that always succeeds. As usual, conditionals and loops can be defined as macros, i.e., **if** ϕ **do** δ_1 **else** δ_2 **fi** $:= [\phi?; \delta_1][\neg\phi?; \delta_2]$ and **while** ϕ **do** δ **done** $:= [\phi?; \delta]^*; \neg\phi?$. We call $\Delta = (\mathcal{D}, \delta)$ a *clocked program* if $\mathcal{D}$ is a

clocked BAT and we assume that $\mathcal{K} \in \mathbb{N}$ is the maximal constant mentioned in $\mathcal{D}$ and δ . We adapt the GOLOG transition semantics from CONGOLOG such that each transition step follows a single primitive action, corresponding to *synchronized conditionals* known from INDIGOLOG [12]. We also extend the relation Trans to include the action such that $\text{Trans}(\delta, s, a, \delta', s')$ is true if there is a single-step transition with action a in the program δ starting in situation s and resulting in situation s' with remaining program δ' . Finally, we assume that the program δ does not fix the time of occurrence for *wait* and instead the time is determined in the transition semantics. Therefore, *wait* actions in δ do not have a time argument t and the timepoint is instead fixed in the transition rule for a *wait* action.⁴ We also write $\text{Trans}^*(\delta, s, \vec{a}, \delta', s')$ for the transitive closure of Trans such that $\vec{a}$ is the sequence of actions resulting from following the transitions according to Trans :

$$\begin{aligned}
\text{Trans}(a, s, a, \delta', s') &\equiv \text{Poss}(a, s) \wedge \delta' = \text{nil} \wedge s' = \text{do}(a, s) \\
\text{Trans}(\text{wait}(), s, a, \delta', s') &\equiv \exists t. a = \text{wait}(t) \wedge \delta' = \text{nil} \wedge s' = \text{do}(a, s) \\
\text{Trans}(\phi?, s, a, \delta', s') &\equiv \text{FALSE} \\
\text{Trans}(\delta_1; \delta_2, s, a, \delta', s') &\equiv \exists \gamma. \delta' = (\gamma; \delta_2) \wedge \text{Trans}(\delta_1, s, a, \gamma, s') \\
&\quad \vee \text{Final}(\delta_1, s) \wedge \text{Trans}(\delta_2, s, a, \delta', s') \\
\text{Trans}(\delta_1 | \delta_2, s, a, \delta', s') &\equiv \text{Trans}(\delta_1, s, a, \delta', s') \vee \text{Trans}(\delta_2, s, a, \delta', s') \\
\text{Trans}(\pi x. \delta, s, a, \delta', s') &\equiv \exists v. \text{Trans}(\delta_v^x, s, a, \delta', s') \\
\text{Trans}(\delta^*, s, a, \delta', s') &\equiv \exists \gamma. (\delta' = \gamma; \delta^*) \wedge \text{Trans}(\delta, s, a, \gamma, s') \\
\text{Trans}(\delta_1 \| \delta_2, s, a, \delta', s') &\equiv \exists \gamma. \delta' = (\gamma \| \delta_2) \wedge \text{Trans}(\delta_1, s, a, \gamma, s') \\
&\quad \vee \exists \gamma. \delta' = (\delta_1 \| \gamma) \wedge \text{Trans}(\delta_2, s, a, \gamma, s')
\end{aligned}$$

$\text{Final}(\delta, s)$ is true if program δ may terminate in situation s :

$$\begin{aligned}
\text{Final}(a, s) &\equiv \text{FALSE} & \text{Final}(\phi?, s) &\equiv \phi[s] \\
\text{Final}(\delta_1; \delta_2, s) &\equiv \text{Final}(\delta_1, s) \wedge \text{Final}(\delta_2, s) \\
\text{Final}(\delta_1 | \delta_2, s) &\equiv \text{Final}(\delta_1, s) \vee \text{Final}(\delta_2, s) \\
\text{Final}(\pi x. \delta, s) &\equiv \exists v. \text{Final}(\delta_v^x, s) & \text{Final}(\delta^*, s) &\equiv \text{TRUE} \\
\text{Final}(\delta_1 \| \delta_2, s) &\equiv \text{Final}(\delta_1, s) \wedge \text{Final}(\delta_2, s)
\end{aligned}$$

► **Example 28.** We consider the following program δ :

$$\begin{aligned}
&(s\text{Brew}(\text{Pot}); \exists m. (\text{WantStrong}(m)) \equiv c_{\text{brew}}(\text{Pot}) \geq 2?; e\text{Brew}(\text{Pot}); \\
&\quad \text{while } (\exists m. \neg \text{IsFull}(m)) \text{ do } \pi m. s\text{Pour}(\text{Pot}, m); e\text{Pour}(\text{Pot}, m); \text{done}) \parallel \text{wait}()^*
\end{aligned}$$

The program first brews coffee. If there is a mug that should be filled with strong coffee, it waits until $c_{\text{brew}}(\text{Pot}) \geq 2$, otherwise it stops brewing earlier. Afterwards, it fills all mugs.

6.2 Program Realization

We describe how to construct a transition system $\mathcal{T}_\Delta$ to determine a realization of a given program Δ similar to $\mathcal{T}_\mathcal{D}$ from above. We first formally define program realization:

⁴ We may still enforce a certain time delay by using clock formulas in tests $\phi?$, as demonstrated in Example 28.

■ **Algorithm 3** Construction of the time-abstract transition system $\mathcal{T}_\Delta$.

```

Open ← {(S0, δ0)}; V ← ∅; E ← ∅
while Open ≠ ∅ do
  (σ, δ) ← POP(Open)
  Acts ← {A(ρ̄) | A ∈ A \ {wait}, ρ̄ ∈ O|A|} ∪ {wait(τ) | τ ∈ TSUCCS(σ, K)}
  for all α ∈ Acts do
    if D ⊨ Trans(δ, σ, α, δ', σ') then
      if ([σ']≈DK, δ') ∉ V then
        V ← V ∪ {([σ']≈DK, δ')}
        Open ← Open ∪ {(σ', δ')}
      E ← E ∪ {([σ]≈DK, δ), ([σ']≈DK, δ')}
  return (V, ([S0]≈DK, δ0), E)

```

► **Definition 29** (Program Realization). Let $\mathcal{D}$ be a BAT and δ a program. An action sequence $\vec{\alpha} = \langle \alpha_1, \dots, \alpha_n \rangle$ is a realization of δ wrt. $\mathcal{D}$ if $\mathcal{D} \models \text{Trans}^*(\delta, S_0, \vec{\alpha}, \delta', \sigma') \wedge \text{Final}(\delta', \sigma')$. The corresponding decision problem is to decide whether such a realization exists.

First, we can show that the equivalence relation $\approx_{\mathcal{D}K}$ can also be applied to the transition semantics, i.e., bisimilar situations cannot be distinguished by possible program transitions:

- **Lemma 30.** Let σ_1, σ_2 be two situations with $\sigma_1 \approx_{\mathcal{D}K} \sigma_2$ and $\mathcal{D} \models \text{Trans}(\delta, \sigma_1, \alpha, \delta^*, \sigma'_1)$.
1. If $\alpha = \text{wait}(\tau_1)$ for some $\tau_1 \in \mathbb{R}_{\geq 0}$, then there is $\tau_2 \in \text{TSUCCS}(\sigma_2, K)$ such that $\text{Trans}(\delta, \sigma_2, \text{wait}(\tau_2), \delta^*, \sigma'_2)$ and $\sigma'_1 \approx_{\mathcal{D}K} \sigma'_2$.
 2. If $\alpha = A(\bar{\rho})$ for some action type $A \neq \text{wait}$, then $\text{Trans}(\delta, \sigma_2, \alpha, \delta^*, \sigma'_2)$ and $\sigma'_1 \approx_{\mathcal{D}K} \sigma'_2$.

This allows us to use time-abstract bisimulations for determining realizability. We adapt Algorithm 1 such that states also contain the remaining program. The resulting algorithm is shown in Algorithm 3. The algorithm is correct, i.e., the transition system contains a final program configuration iff a realization exists:

► **Lemma 31.** Let $\Delta = (\mathcal{D}, \delta)$ be a program over a clocked BAT $\mathcal{D}$ and $\mathcal{T}_\Delta$ the transition system returned by Algorithm 3. Then $\mathcal{T}_\Delta$ contains a state $([\sigma']_{\approx_{\mathcal{D}K}}, \delta')$ reachable from $([S_0]_{\approx_{\mathcal{D}K}}, \delta)$ with $\mathcal{D} \models \text{Final}(\delta', \sigma')$ iff there is a realization of program Δ .

Moreover, the transition system is finite:

► **Lemma 32.** $\mathcal{T}_\Delta$ has finitely many states.

Proof. As $\mathcal{O}$ and $\mathcal{A}$ are finite, each state only has finitely many successors. Furthermore, as there are only finitely many fluents $\mathcal{F}$ and objects $\mathcal{O}$, $\approx_{\mathcal{D}K}$ has finitely many equivalence classes. Finally, there are finitely many reachable sub-programs from δ [8]. Therefore, the time-abstract transition system $\mathcal{T}_\Delta$ has finitely many states. ◀

Combining the results, we obtain:

► **Theorem 33.** The realization problem for a clocked program $\Delta = (\mathcal{D}, \delta)$ is decidable.

Proof. By Lemma 31, it is sufficient to check whether there is a reachable state in $\mathcal{T}_\Delta$ that corresponds to a final program configuration. By Lemma 32, $\mathcal{T}_\Delta$ is finite and so explicitly constructing it is feasible. Finally, checking whether $\mathcal{D} \models \text{Trans}(\delta, \sigma, \alpha, \delta', \sigma')$ reduces to checking entailment of clocked formulas uniform in σ (in particular, checking $\mathcal{D} \models \text{Poss}(\alpha, \sigma)$ and $\mathcal{D} \models \phi[\sigma]$ for tests ϕ). By Theorem 11, this is decidable. ◀

We conclude by presenting a realization of the example:

► **Example 34.** The following sequence is a realization of the program from Example 28:

```
sBrew(Pot); wait(2); eBrew(Pot); sPour(Pot, Mug1);
ePour(Pot, Mug1); sPour(Pot, Mug2); ePour(Pot, Mug2);
```

However, the program does not fill every mug with the correct kind of coffee. In fact, let $\delta' := \delta; \forall m(\text{Strong}(m) \equiv \text{WantStrong}(m))?$ be the program δ augmented with a check whether every mug was filled with the right kind of coffee. Then, no realization of δ' exists: As the program brews coffee only once, it fills all mugs with the same coffee.

7 Conclusion

We investigated how continuous time can be modeled in the situation calculus (SC), focusing on *reachability*, which asks whether there is a reachable situation that satisfies some given formula ϕ , and *program realization*, the task to determine an action sequence corresponding to a successful execution of a given GOLOG program. For these problems, the standard approaches for modeling time in the SC are unsuitable, because reachability (and thus also realizability) is undecidable. We presented an alternative approach based on *clocks*, i.e., real-valued fluents with restricted successor state axioms, and described sound and complete algorithms that explicitly construct a finite transition system, showing that both problems are decidable.

Our approach is restricted to basic action theories with a finite set of objects. Since quantifiers can be eliminated by grounding, our setting is essentially that of the propositional situation calculus [10], modulo the real-valued time of *wait*. For future work, it may be interesting to extend this approach towards more expressive formalisms that retain decidability, e.g., *bounded action theories* [14], or the two-variable fragment of the SC [17], which has been used for verifying GOLOG programs over local-effect [9] and acyclic [31] BATs, and for synthesizing GOLOG policies against uncontrollable environments with LTL_f goals [18]. Extending our notion of time to the synthesis setting is a promising direction.

References

- 1 Rajeev Alur. Timed automata. In Nicolas Halbwachs and Doron Peled, editors, *Computer Aided Verification (CAV)*, Lecture Notes in Computer Science, pages 8–22, Berlin, Heidelberg, 1999. Springer. doi:10.1007/3-540-48683-6_3.
- 2 Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science, pages 209–229, Berlin, Heidelberg, 1993. Springer. doi:10.1007/3-540-57318-6_30.
- 3 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, April 1994. doi:10.1016/0304-3975(94)90010-8.
- 4 Rajeev Alur and P. Madhusudan. Decision Problems for Timed Automata: A Survey. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems*, Lecture Notes in Computer Science, pages 1–24. Springer, Berlin, Heidelberg, 2004. doi:10.1007/978-3-540-30080-9_1.
- 5 Vitaliy Batusov, Giuseppe De Giacomo, and Mikhail Soutchanski. Hybrid temporal situation calculus. In Marie-Jean Meurs and Frank Rudzicz, editors, *Advances in Artificial Intelligence*, Lecture Notes in Computer Science, pages 173–185, Cham, 2019. Springer. doi:10.1007/978-3-030-18305-9_14.

- 6 Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2):291–345, August 2004. doi:10.1016/j.tcs.2004.04.003.
- 7 Béatrice Bérard and Catherine Dufourd. Timed automata and additive clock constraints. *Information Processing Letters*, 75(1):1–7, July 2000. doi:10.1016/S0020-0190(00)00075-2.
- 8 Jens Claßen and Gerhard Lakemeyer. A logic for non-terminating Golog programs. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 589–599. AAAI Press, 2008. URL: <https://aaai.org/papers/KR08-058-a-logic-for-non-terminating-golog-programs/>.
- 9 Jens Claßen, Martin Liebenberg, Gerhard Lakemeyer, and Benjamin Zarriß. Exploring the boundaries of decidable verification of non-terminating Golog programs. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI)*, pages 1012–1019. AAAI Press, 2014. doi:10.1609/AAAI.V28I1.8875.
- 10 Giuseppe De Giacomo, Yves Lespérance, Matteo Mancanelli, and Gianmarco Parretti. Reactive synthesis for Golog specifications in the propositional situation calculus. In *Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 2026.
- 11 Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121:109–169, 2000. doi:10.1016/S0004-3702(00)00031-X.
- 12 Giuseppe De Giacomo, Yves Lespérance, Hector J. Levesque, and Sebastian Sardina. IndiGolog: A high-level programming language for embedded reasoning agents. In *Multi-Agent Programming*. Springer, 2009. doi:10.1007/978-0-387-89299-3.
- 13 Giuseppe De Giacomo, Yves Lespérance, and Fabio Patrizi. Bounded situation calculus action theories and decidable verification. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*, May 2012.
- 14 Giuseppe De Giacomo, Yves Lespérance, and Fabio Patrizi. Bounded situation calculus action theories. *Artificial Intelligence*, 237:172–203, August 2016. doi:10.1016/j.artint.2016.04.006.
- 15 Alberto Finzi and Fiora Pirri. Representing flexible temporal behaviors in the situation calculus. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 436–441, 2005. URL: <http://ijcai.org/Proceedings/05/Papers/1562.pdf>.
- 16 Michael Gelfond, Vladimir Lifschitz, and Arkady Rabinov. What Are the Limitations of the Situation Calculus? In *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pages 167–179. Springer Netherlands, Dordrecht, 1991. doi:10.1007/978-94-011-3488-0_8.
- 17 Yilan Gu and Mikhail Soutchanski. A description logic based situation calculus. *Annals of Mathematics and Artificial Intelligence*, 58(1):3–83, February 2010. doi:10.1007/s10472-010-9176-z.
- 18 Till Hofmann and Jens Claßen. LTLf synthesis on first-order agent programs in nondeterministic environments. In *Proceedings of the 39th AAAI Conference on Artificial Intelligence (AAAI)*, volume 39, pages 14976–14986, 2025. doi:10.1609/aaai.v39i14.33642.
- 19 Till Hofmann, Stefan Schupp, and Gerhard Lakemeyer. Decidable reasoning about time in finite-domain situation calculus theories, 2026. doi:10.48550/arXiv.2402.03164.
- 20 Robert Kowalski and Marek Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, March 1986. doi:10.1007/BF03037383.
- 21 John McCarthy. Situations, actions, and causal laws. Technical Report, Stanford University, 1963.
- 22 Rob Miller. A case study in reasoning about actions and continuous change. In *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI)*, page 21, 1996.
- 23 Marvin Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Englewood Cliffs, N.J, 1967.

- 24 Erik T. Mueller. Event Calculus. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Foundations of Artificial Intelligence*, volume 3 of *Handbook of Knowledge Representation*, pages 671–708. Elsevier, January 2008. doi:10.1016/S1574-6526(07)03017-9.
- 25 Joël Ouaknine and James Worrell. On the decidability and complexity of Metric Temporal Logic over finite words. *Logical Methods in Computer Science*, 3(1):1–27, February 2007. Publisher: Episciences.org. doi:10.2168/LMCS-3(1:8)2007.
- 26 Javier Pinto and Raymond Reiter. Reasoning about time in the situation calculus. *Annals of Mathematics and Artificial Intelligence*, 14(2-4):251–268, 1995. doi:10.1007/BF01530822.
- 27 Fiora Pirri and Ray Reiter. Some contributions to the metatheory of the situation calculus. *Journal of the ACM*, 46(3):325–361, 1999. doi:10.1145/316542.316545.
- 28 Raymond Reiter. Natural Actions, Concurrency and Continuous Time in the Situation Calculus. In *Proceedings of the 5th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 2–13, San Francisco, CA, 1996. Morgan Kaufmann Publishers Inc.
- 29 Raymond Reiter. *Knowledge in action: logical foundations for specifying and implementing dynamical systems*. MIT Press, 2001.
- 30 Kristof Van Belleghem, Marc Denecker, and Danny De Schreye. On the relation between situation calculus and event calculus. *The Journal of Logic Programming*, 31(1):3–37, April 1997. doi:10.1016/S0743-1066(96)00123-9.
- 31 Benjamin Zarri  and Jens Cla en. Decidable verification of Golog programs over non-local effect actions. In Dale Schuurmans and Michael Wellman, editors, *Proceedings of the thirtieth AAAI conference on artificial intelligence (AAAI 2016)*, pages 1109–1115. AAAI Press, 2016. doi:10.1609/AAAI.V30I1.10109.

A Proofs

► **Theorem 2.** *The reachability problem for a real-valued BAT is undecidable.*

Proof. Let $Incrs = \{(p, c, q)_i\}_i$ be the finite set of increment instructions, where p is the instruction label, $c \in \{1, 2\}$ is the index of the counter to be incremented, and q is the next instruction label. Similarly, let $Decrs = \{(p, c, q, r)_i\}_i$ be the finite set of decrement instructions, where p is the instruction label, $c \in \{1, 2\}$ is the index of the counter to be decremented, q is the jump instruction if the condition is true, and r is the jump instruction otherwise. We define a BAT $\mathcal{D}_M$ corresponding to a 2CM M as follows: We use the unary relational fluent $Next(i)$ to describe the next instruction i , two nullary functional fluents f_1, f_2 for counter values (we omit parentheses and write f_1 instead of $f_1()$), and the nullary relational fluent $Halt$ which is true iff M halted. We define the BAT $\mathcal{D}_M$ as follows. Initially, both counting fluents are set to zero and the next instruction is s_0 : $\mathcal{D}_0 = \{f_1 = 0, f_2 = 0, Next(i) \equiv i = s_0\}$. An action is possible if it is the next instruction: $Poss(i) \equiv Next(i)$. The SSA for each counting fluent $f_i \in \{f_1, f_2\}$ works as follows: If the last instruction was an increment instruction from $Incrs$ and it incremented c_i , then the fluent f_i is incremented as well. Similarly, if the last instruction was a decrement instruction and the counter value was larger than zero, then the fluent is decremented, otherwise it remains unchanged: $f_i(\text{do}(a, s)) = n \equiv \bigvee_{(p,c,q) \in Incrs} a = p \wedge c = i \wedge n = f_i + 1 \vee \bigvee_{(p,c,q,r) \in Decrs} a = p \wedge c = i \wedge (f_i > 0 \wedge n = f_i - 1 \vee f_i = 0 \wedge n = 0) \vee f_i = n \wedge \bigwedge_{(p,c,q) \in Incrs} (a \neq p \vee c \neq i) \wedge \bigwedge_{(p,c,q,r) \in Decrs} (a \neq p \vee c \neq i)$. The SSA for $Next$ works as follows: If the last instruction was an increment instruction that sets the next instruction to q , then set $Next$ accordingly. Similarly, if the last instruction was a decrement instruction, then branch on the value of the corresponding counter (i.e., whether $f_i > 0$) and set the next instruction accordingly: $Next(i) \equiv \bigvee_{(p,c,q) \in Incrs} a = p \wedge i = q \vee \bigvee_{(p,c,q,r) \in Decrs} a = p \wedge ((c = 1 \wedge f_1 > 0 \vee c = 2 \wedge f_2 > 0) \wedge i = q \vee (c = 1 \wedge f_1 = 0 \vee c = 2 \wedge f_2 = 0) \wedge i = r)$.

By construction, $\mathcal{D}_M \models \exists s \text{ Exec}(s) \wedge Halt(s)$ iff the 2CM reaches **HALT**. ◀

► **Theorem 3.** *The time-bounded reachability problem for a real-valued BAT and a time bound $b \geq 2$ is undecidable.*

Proof. The construction is the same as in Theorem 2, except that counting fluents work differently: A counter value u is encoded by a fluent value 2^{1-u} . Hence, every increment of f_i is replaced by an update to $\frac{f_i}{2}$ and every decrement of f_i is replaced by an update to $2f_i$. Furthermore, a comparison $f_i = 0$ is replaced by $f_i = 2$ and $f_i > 0$ is replaced by $f_i < 2$. Hence, the initial situation is $\mathcal{D}_0 = \{f_1 = 2, f_2 = 2, \text{Next}(i) \equiv i = s_0\}$. The precondition axiom is: $\text{Poss}(i) \equiv \text{Next}(i)$. The SSA for each counting fluent $f_i \in \{f_1, f_2\}$ is defined as: $f_i(\text{do}(a, s)) = n \equiv \bigvee_{(p,c,q) \in \text{Incrs}} a = p \wedge c = i \wedge n = \frac{f_i}{2} \vee \bigvee_{(p,c,q,r) \in \text{Decrs}} a = p \wedge c = i \wedge (f_i < 2 \wedge n = 2f_i \vee f_i = 2 \wedge n = 2) \vee f_i = n \wedge \bigwedge_{(p,c,q) \in \text{Incrs}} (a \neq p \vee c \neq i) \wedge \bigwedge_{(p,c,q,r) \in \text{Decrs}} (a \neq p \vee c \neq i)$. The SSA for Next is defined as: $\text{Next}(i) \equiv \bigvee_{(p,c,q) \in \text{Incrs}} a = p \wedge i = q \vee \bigvee_{(p,c,q,r) \in \text{Decrs}} a = p \wedge ((c = 1 \wedge f_1 < 2 \vee c = 2 \wedge f_2 < 2) \wedge i = q \vee (c = 1 \wedge f_1 = 2 \vee c = 2 \wedge f_2 = 2) \wedge i = r)$

By construction, $\mathcal{D} \models \exists s \text{Exec}(s) \wedge \text{Bounded}_2(s) \wedge \text{Halt}(s)$ iff the 2CM reaches the instruction **HALT**. ◀

► **Theorem 4.** *The reachability problem for timed BATs without real-valued functional fluents is undecidable.*

Proof. By reduction of the language emptiness problem for TAs with 4 clocks extended with additive clock constraints of the form $c_1 + c_2 = 1$, which is undecidable [7]. Let $\mathcal{A} = (L, l_0, L_F, \Sigma_{\mathcal{A}}, X, E)$ be such a TA. For an edge $e = (l, a, g, Y, l')$, we write l_e, g_e, Y_e, l'_e for its components, and for each clock $x \in X$, we write $R_x := \{e \in E \mid x \in Y_e\}$ for the set of edges resetting x . We define a BAT $\mathcal{D}_{\mathcal{A}}$ corresponding to $\mathcal{A}$ as follows. For each edge $e \in E$, there is one action type $E_e(t)$ with occurrence time t . For each location $l \in L$, there is a relational fluent $\text{Loc}_l(s)$. Finally, for each clock $x \in X$, there is a process $\text{Reset}_x(t, s)$, where t is an argument of sort time. Let $\text{start}(S_0) = 0$. Initially, $\mathcal{A}$ is in location l_0 , and every clock s was last reset at time 0: $\text{Loc}_{l_0}(S_0), \neg \text{Loc}_l(S_0)$ for each $l \neq l_0$, $\text{Reset}_x(t, S_0) \equiv t = 0$ for each $x \in X$. For a clock constraint g and variable t of sort time, the formula $\mu_g(t, s)$ is defined by

- $\mu_{x \bowtie r}(t, s) := \exists t_x. \text{Reset}_x(t_x, s) \wedge t - t_x \bowtie r$
- $\mu_{x+y=1}(t, s) := \exists t_x, t_y. \text{Reset}_x(t_x, s) \wedge \text{Reset}_y(t_y, s) \wedge (t - t_x) + (t - t_y) = 1$
- $\mu_{g_1 \wedge g_2} := \mu_{g_1}(t, s) \wedge \mu_{g_2}(t, s)$

An edge action is possible iff time does not decrease, $\mathcal{A}$ is in the source location, and the guard holds at the occurrence time: $\text{Poss}(E_e(t), s) \equiv t \geq \text{start}(s) \wedge \text{Loc}_{l_e}(s) \wedge \mu_{g_e}(t, s)$. The SSA for Loc_l states that $\mathcal{A}$ is in location l after an action iff that action is an edge with target l : $\text{Loc}_l(\text{do}(a, s)) \equiv \bigvee_{e \in E, l'_e = l} \exists t. a = E_e(t)$. The SSA for Reset_x states that the last reset time of x is the occurrence time of the current action if that action resets x , and is unchanged otherwise: $\text{Reset}_x(t, \text{do}(a, s)) \equiv \bigvee_{e \in R_x} a = E_e(t) \vee \text{Reset}_x(t, s) \wedge \bigwedge_{e \in R_x} \forall t'. a \neq E_e(t')$. By construction, $\mathcal{D}_{\mathcal{A}} \models \exists s. \text{Exec}(s) \wedge \bigvee_{l \in L_F} \text{Loc}_l(s)$ iff there is an accepting timed word of $\mathcal{A}$. ◀

► **Theorem 10.** *Let ϕ be a regressable formula. Then $\mathcal{R}[\phi]$ is uniform in S_0 and $\mathcal{D} \models \phi$ iff $\mathcal{D}_0 \cup \mathcal{D}_c \models \mathcal{R}[\phi]$.*

Proof. By induction on the maximal length of all situation terms in ϕ and structural sub-induction on ϕ .

Base case. Let $\sigma = S_0$. Then $\mathcal{R}[\phi] = \phi$ and $\mathcal{D} \models \phi$ iff $\mathcal{D}_0 \cup \mathcal{D}_c \models \phi$ as ϕ is uniform in S_0 .

Induction step. Let $\sigma = \text{do}(\alpha, \sigma')$. Here, we only consider the case of formulas of the form $\phi = f(\vec{\rho}, \sigma) \bowtie r$ and refer to [27] for the other cases. If $\alpha = \text{wait}(t)$ for some $t \in \mathbb{R}_{\geq 0}$, then $\mathcal{R}[f(\vec{\rho}, \sigma) \bowtie r] = \mathcal{R}[f(\vec{\rho}, \sigma') \bowtie r']$ with $r' = r - t$. By induction, $\mathcal{D}_0 \cup \mathcal{D}_c \models \mathcal{R}[\phi_f(\vec{\rho}, \alpha, \sigma) \bowtie r']$ iff $\mathcal{D} \models \phi_f(\vec{\rho}, \alpha, \sigma) \bowtie r'$. Otherwise (if α is not *wait*), by the definition

of the SSA for f , $\mathcal{D} \models f(\vec{\rho}, \text{do}(\alpha, \sigma)) = y \equiv \phi_f(\vec{\rho}, \alpha, \sigma) \wedge y = 0 \vee \neg \phi_f(\vec{\rho}, \alpha, \sigma) \wedge y = f(\vec{\rho}, \sigma)$ and so $\mathcal{D} \models f(\vec{\rho}, \text{do}(\alpha, \sigma)) \bowtie r \equiv \phi_f(\vec{\rho}, \alpha, \sigma) \wedge 0 \bowtie r \vee \neg \phi_f(\vec{\rho}, \alpha, \sigma) \wedge f(\vec{\rho}, \sigma) \bowtie r$. By induction, this holds iff $\mathcal{D}_0 \cup \mathcal{D}_c \models \mathcal{R}[\phi_f(\vec{\rho}, \alpha, \sigma) \wedge 0 \bowtie r \vee \neg \phi_f(\vec{\rho}, \alpha, \sigma) \wedge f(\vec{\rho}, \sigma) \bowtie r]$ and so $\mathcal{D} \models f(\vec{\rho}, \text{do}(\alpha, \sigma)) \bowtie r$ iff $\mathcal{D}_0 \cup \mathcal{D}_c \models \mathcal{R}[f(\vec{\rho}, \text{do}(\alpha, \sigma)) \bowtie r]$. $\blacktriangleleft$

► **Lemma 14.** *If $\mathcal{D}$ is a clocked BAT and σ a situation, then ν_σ always exists and is unique.*

Proof. By definition of clocked BATs, each clock is initialized to 0. Furthermore, once the initial value is fixed, the SSAs uniquely define each clock value in all successor situations. $\blacktriangleleft$

► **Lemma 17.** *Let $\sigma_1 \approx_{\mathcal{DK}} \sigma_2$ and let ϕ be a $\mathcal{K}$ -bounded clocked formula uniform in s . Then $\mathcal{D} \models \phi[\sigma_1]$ iff $\mathcal{D} \models \phi[\sigma_2]$.*

Proof. Let $f : \nu_{\sigma_1} \rightarrow \nu_{\sigma_2}$ be a bijection witnessing $\sigma_1 \approx_{\mathcal{DK}} \sigma_2$. We show that $\mathcal{D} \models \phi[\sigma_1]$ iff $\mathcal{D} \models \phi[\sigma_2]$ by structural induction on ϕ .

- Let $\phi = R(\vec{\rho}, s)$ for some fluent predicate R . By Definition 12, it directly follows that $\mathcal{D} \models \phi[\sigma_1]$ iff $\mathcal{D} \models \phi[\sigma_2]$.
- Let $\phi = \omega \bowtie \tau$ for some ground situation-suppressed clock term $\omega \in \mathcal{C}$ and $\tau \in \mathbb{N}$. Let $\nu_{\sigma_1}(\omega) = v_1$ and $\nu_{\sigma_2}(\omega) = v_2$. As f is a witness for $\sigma_1 \approx_{\mathcal{DK}} \sigma_2$, it follows that $f(\omega, v_1) = (\omega, v_2)$ and $v_1 \sim_{\mathcal{K}} v_2$. By Definition 15, $v_1 \bowtie \tau$ iff $v_2 \bowtie \tau$ and so $\mathcal{D} \models \omega[\sigma_1] \bowtie \tau$ iff $\mathcal{D} \models \omega[\sigma_2] \bowtie \tau$.
- The cases for Boolean connectives and quantification follow by induction. $\blacktriangleleft$

► **Theorem 19.** *$\approx_{\mathcal{DK}}$ is a time-abstract bisimulation.*

Proof. We show that $\approx_{\mathcal{DK}}$ satisfies the criteria from Definition 12. Let $\sigma_1 \approx_{\mathcal{DK}} \sigma_2$ such that $f : \nu_{\sigma_1} \rightarrow \nu_{\sigma_2}$ is a bijection witnessing $\sigma_1 \approx_{\mathcal{DK}} \sigma_2$.

1. Let ϕ be a clocked formula uniform in s . It directly follows from Lemma 17 that $\mathcal{D} \models \phi[\sigma_1]$ iff $\mathcal{D} \models \phi[\sigma_2]$.
2. Let $A \neq \text{wait}$ be some action type, $\vec{\rho}$ a ground tuple of arguments. Let $\sigma'_1 = \text{do}(A(\vec{\rho}), \sigma_1)$ and $\sigma'_2 = \text{do}(A(\vec{\rho}), \sigma_2)$. We show that $\sigma'_1 \approx_{\mathcal{DK}} \sigma'_2$:
 - a. Let $R \in \mathcal{F}$ be a relational fluent and $\vec{\rho}'$ a ground tuple of arguments. We show that $\mathcal{D} \models R(\vec{\rho}', \sigma'_1)$ iff $\mathcal{D} \models R(\vec{\rho}', \sigma'_2)$. There is an SSA for R of the form $R(\vec{\sigma}, \text{do}(a, s)) \equiv \phi_R(\vec{\sigma}, a, s)$, where $\phi_R(\vec{\sigma}, a, s)$ is a clocked formula uniform in s . As $\sigma_1 \approx_{\mathcal{DK}} \sigma_2$, it directly follows from Lemma 17 that $\mathcal{D} \models \phi_R(\vec{\rho}', A(\vec{\rho}), \sigma_1)$ iff $\mathcal{D} \models \phi_R(\vec{\rho}', A(\vec{\rho}), \sigma_2)$ and therefore $\mathcal{D} \models R(\vec{\rho}', \sigma'_1)$ iff $\mathcal{D} \models R(\vec{\rho}', \sigma'_2)$.
 - b. We construct a bijection $f' : \nu_{\sigma'_1} \rightarrow \nu_{\sigma'_2}$ witnessing $\approx_{\mathcal{DK}}$. Let c be a functional fluent and $\omega = c(\vec{\rho}')$. There is an SSA of the form $c(\vec{\sigma}, \text{do}(a, s)) = y \equiv \exists t. a = \text{wait}(t) \wedge y = c(\vec{\sigma}, s) + t \vee \neg \exists t. a = \text{wait}(t) \wedge (\phi_c(\vec{\sigma}, a, s) \wedge y = 0 \vee \neg \phi_c(\vec{\sigma}, a, s) \wedge y = c(\vec{\sigma}, s))$. Thus, $\omega[\sigma'_1] = 0$ if $\mathcal{D} \models \phi_c(\vec{\rho}, \alpha, \sigma_1)$ and $\omega[\sigma'_1] = \omega[\sigma_1]$ otherwise, similarly for $\omega[\sigma'_2]$. Now, as $\phi_c(\vec{\rho}, a, s)$ is a clocked formula uniform in s , by Lemma 17, $\mathcal{D} \models \phi_c(\vec{\rho}, \alpha, \sigma_1)$ iff $\mathcal{D} \models \phi_c(\vec{\rho}, \alpha, \sigma_2)$. Hence, for each ω , we can set $f'(\omega, 0) = (\omega, 0)$ if $\mathcal{D} \models \phi_c(\vec{\rho}, \alpha, \sigma_1)$ and $f'(\omega, \nu_{\sigma_1}(\omega)) = (\omega, \nu_{\sigma_2}(\omega))$ otherwise.
3. Let $\tau_1 \in \mathbb{R}_{\geq 0}$ and $\sigma'_1 = \text{do}(\text{wait}(\tau_1), \sigma_1)$. As $\nu_{\sigma_1} \cong_{\mathcal{K}} \nu_{\sigma_2}$, it follows with Proposition 18 that there is some $\tau_2 \in \mathbb{R}_{\geq 0}$ such that $\nu_{\sigma_1} + \tau_1 \cong_{\mathcal{K}} \nu_{\sigma_2} + \tau_2$. Let $\sigma'_2 = \text{do}(\text{wait}(\tau_2), \sigma_2)$. We show that $\sigma'_1 \approx_{\mathcal{DK}} \sigma'_2$:
 - a. Let $R \in \mathcal{F}$ be a relational fluent and $\vec{\rho}'$ a ground tuple of arguments. We show that $\mathcal{D} \models R(\vec{\rho}', \sigma'_1)$ iff $\mathcal{D} \models R(\vec{\rho}', \sigma'_2)$. There is an SSA for R of the form $R(\vec{\sigma}, \text{do}(a, s)) \equiv \phi_R(\vec{\sigma}, a, s)$, where $\phi_R(\vec{\sigma}, a, s)$ is a clocked formula uniform in s . As $\sigma_1 \approx_{\mathcal{DK}} \sigma_2$, it directly follows from Lemma 17 that $\mathcal{D} \models \phi_R(\vec{\rho}', \text{wait}(\tau_1), \sigma_1)$ iff $\mathcal{D} \models \phi_R(\vec{\rho}', \text{wait}(\tau_2), \sigma_2)$ and therefore $\mathcal{D} \models R(\vec{\rho}', \sigma'_1)$ iff $\mathcal{D} \models R(\vec{\rho}', \sigma'_2)$.

- b. Let c be a functional fluent and $\omega = c(\vec{\rho}')$. By definition of the SSA for c , it follows that $\nu_{\sigma'_1}(\omega) = \nu_{\sigma_1}(\omega) + \tau_1$ and $\nu_{\sigma'_2}(\omega) = \nu_{\sigma_2}(\omega) + \tau_2$. We can set $f'(\omega, \nu_{\sigma'_1}(\omega)) = (\omega, \nu_{\sigma'_2}(\omega))$. As $\nu_{\sigma_1} + \tau_1 \cong_{\mathcal{K}} \nu_{\sigma_2} + \tau_2$, f' is a bijection witnessing $\approx_{\mathcal{DK}}$. $\blacktriangleleft$

► **Lemma 21.** $[\sigma]_{\approx_{\mathcal{DK}}}$ is reachable in $\mathcal{T}_{\mathcal{D}}$ iff there is some $\sigma' \in [\sigma]_{\approx_{\mathcal{DK}}}$ with $\mathcal{D} \models \text{Exec}(\sigma')$.

Proof.

⇐: By induction on the number n of do operators in σ .

Base case. Let $n = 0$. Therefore, $\sigma = S_0$. By definition, $[S_0]_{\approx_{\mathcal{DK}}}$ is reachable in $\mathcal{T}_{\mathcal{D}}$.

Induction step. Let $\sigma = \text{do}(\alpha, \sigma^*)$ such that $\mathcal{D} \models \text{Exec}(\sigma)$. By induction, $[\sigma^*]_{\approx_{\mathcal{DK}}}$ is reachable from $[S_0]_{\approx_{\mathcal{DK}}}$ in $\mathcal{T}_{\mathcal{D}}$. By definition of $\mathcal{T}_{\mathcal{D}}$, $[\sigma^*]_{\approx_{\mathcal{DK}}} \rightarrow [\sigma]_{\approx_{\mathcal{DK}}}$ is a transition in $\mathcal{T}_{\mathcal{D}}$ and so $[\sigma]_{\approx_{\mathcal{DK}}}$ is also reachable in $\mathcal{T}_{\mathcal{D}}$.

⇒: Follows immediately by the definition of $\mathcal{T}_{\mathcal{D}}$. $\blacktriangleleft$

► **Lemma 22.** The time-abstract transition system $\mathcal{T}_{\mathcal{D}}$ has finitely many states.

Proof. As there are only finitely many fluents $\mathcal{F}$, objects $\mathcal{O}$, and clock regions, $\approx_{\mathcal{DK}}$ has finitely many equivalence classes. Also, as $\mathcal{O}$ and $\mathcal{A}$ are finite, each state only has finitely many successors. Therefore, the time-abstract transition system $\mathcal{T}_{\mathcal{D}}$ has finitely many states. $\blacktriangleleft$

► **Lemma 24.** Let σ_1 and σ_2 be two situations. Then it is decidable whether $\sigma_1 \approx_{\mathcal{DK}} \sigma_2$.

Proof. For every fluent $R \in \mathcal{F}$ and parameters $\vec{\rho}$, $\mathcal{D} \models R(\vec{\rho}, \sigma_1)$ iff $\mathcal{D}_0 \cup \mathcal{D}_c \models \mathcal{R}[R(\vec{\rho}, \sigma_1)]$, similarly for σ_2 . Following Theorem 11, checking whether $\mathcal{D}_0 \cup \mathcal{D}_c \models \mathcal{R}[R(\vec{\rho}, \sigma)]$ is decidable. Furthermore, because $\mathcal{D}$ has finitely many objects and finitely many fluents, there are only finitely many fluent atoms that need to be checked. Finally, checking for the existence of a bijection $f : \nu_{\sigma_1} \rightarrow \nu_{\sigma_2}$ can be done by enumerating all possible bijections (of which there are finitely many) and checking whether all requirements are satisfied. $\blacktriangleleft$

► **Lemma 25.** For a clocked BAT $\mathcal{D}$, Algorithm 1 terminates and returns the time-abstract transition system $\mathcal{T}_{\mathcal{D}}$.

Proof. Let $\mathcal{T}_{\mathcal{D}}$ be the time-abstract TS of a BAT $\mathcal{D}$ and T the TS returned by Algorithm 1. We show for every path p starting in $[S_0]_{\approx_{\mathcal{DK}}}$ that p is a path in $\mathcal{T}_{\mathcal{D}}$ iff p is a path in T .

⇒: Let $p = [S_0]_{\approx_{\mathcal{DK}}} \rightarrow^* [\sigma]_{\approx_{\mathcal{DK}}}$ be a path in $\mathcal{T}_{\mathcal{D}}$. We show by induction on the length n of p that p is a path in T .

Base case. Let $n = 0$, i.e., p is the empty path, which is also a path in T .

Induction step. Let $p = [S_0]_{\approx_{\mathcal{DK}}} \rightarrow^* [\sigma^*]_{\approx_{\mathcal{DK}}} \rightarrow [\sigma]_{\approx_{\mathcal{DK}}}$. By induction, $[S_0]_{\approx_{\mathcal{DK}}} \rightarrow^* [\sigma^*]_{\approx_{\mathcal{DK}}}$ is a path in T . It remains to be shown that $[\sigma^*]_{\approx_{\mathcal{DK}}} \rightarrow [\sigma]_{\approx_{\mathcal{DK}}}$ is a transition in T . Clearly, $\sigma = \text{do}(\alpha, \sigma^*)$ for some α . We distinguish two cases:

1. Assume $\alpha = A(\vec{\rho})$ for some action type $A \in \mathcal{A} \setminus \{\text{wait}\}$. α is added to Acts in line 4. By definition of $\mathcal{T}_{\mathcal{D}}$, $\mathcal{D} \models \text{Exec}(\sigma)$ and so $\mathcal{D} \models \text{Poss}(\alpha, \sigma^*)$. Hence, $[\sigma]_{\approx_{\mathcal{DK}}} \in V$ (either it is added in line 8 or it was in V before) and $([\sigma^*]_{\approx_{\mathcal{DK}}}, [\sigma]_{\approx_{\mathcal{DK}}})$ is added to E in line 10.
2. Otherwise, $\alpha = \text{wait}(\tau)$ for some $\tau \in \mathbb{R}_{\geq 0}$. By Proposition 23, there is a $\tau' \in \text{Succs}$ such that $\nu_{\sigma^*} + \tau \cong_{\mathcal{K}} \nu_{\sigma^*} + \tau'$ and so $\alpha' = \text{wait}(\tau')$ is added to Acts in line 4. Also, $\text{do}(\alpha, \sigma^*) \approx_{\mathcal{DK}} \text{do}(\alpha', \sigma^*)$, and so $[\text{do}(\alpha, \sigma^*)]_{\approx_{\mathcal{DK}}} = [\text{do}(\alpha', \sigma^*)]_{\approx_{\mathcal{DK}}} = [\sigma]_{\approx_{\mathcal{DK}}}$. Furthermore, as the precondition axiom of wait is $\text{Poss}(\text{wait}(t), s) \equiv t \geq 0$, it is clear that $\mathcal{D}_0 \cup \mathcal{D}_c \models \mathcal{R}[\text{Poss}(\alpha', \sigma^*)]$. Therefore, $[\sigma]_{\approx_{\mathcal{DK}}} \in V$ (either it is added in line 8 or it was already in V before) and $([\sigma^*]_{\approx_{\mathcal{DK}}}, [\sigma]_{\approx_{\mathcal{DK}}})$ is added to E in line 10.

⇐: Let $p = [S_0]_{\approx_{\mathcal{DK}}} \rightarrow^* [\sigma]_{\approx_{\mathcal{DK}}}$ be a path in T . We show by induction on the length $n = |p|$ of p that (1) $\mathcal{D} \models \text{Exec}(\sigma)$ and (2) p is a path in $\mathcal{T}_{\mathcal{D}}$.

Base case. Let $n = 0$, i.e., p is the empty path. Clearly, the empty path is also a path in $\mathcal{T}_D$. Also, by definition, $D \models \text{Exec}(S_0)$.

Induction step. Let $p = [S_0]_{\approx_{\mathcal{DK}}} \rightarrow^* [\sigma^*]_{\approx_{\mathcal{DK}}} \rightarrow [\sigma]_{\approx_{\mathcal{DK}}}$. By induction, $D \models \text{Exec}(\sigma^*)$ and $[S_0]_{\approx_{\mathcal{DK}}} \rightarrow^* [\sigma^*]_{\approx_{\mathcal{DK}}}$ is a path in $\mathcal{T}_D$. It remains to be shown that $D \models \text{Exec}(\sigma)$ and $[\sigma^*]_{\approx_{\mathcal{DK}}} \rightarrow [\sigma]_{\approx_{\mathcal{DK}}}$ is a transition in $\mathcal{T}_D$. Clearly, $\sigma = \text{do}(\alpha, \sigma^*)$ for some ground action term α . From the definition of Algorithm 1, it directly follows that $D \models \text{Poss}(\alpha, \sigma^*)$ and so $D \models \text{Exec}(\sigma)$. As $D \models \text{Exec}(\sigma)$, by definition of $\mathcal{T}_D$, $[\sigma^*]_{\approx_{\mathcal{DK}}} \rightarrow [\sigma]_{\approx_{\mathcal{DK}}}$ is a transition in $\mathcal{T}_D$.

We now show that the algorithm terminates. By Theorem 11 checking whether $D_0 \cup D_c \models \mathcal{R}[\text{Poss}(\alpha, \sigma)]$ is decidable. Also, by Lemma 24, checking for each new situation $\text{do}(\alpha, \sigma)$ whether there is an existing state $[\sigma]_{\approx_{\mathcal{DK}}} \in V$ (line 7) is decidable. Furthermore, by Lemma 22, $\mathcal{T}_D$ is finite, and so the loop eventually terminates. $\blacktriangleleft$

► **Lemma 30.** *Let σ_1, σ_2 be two situations with $\sigma_1 \approx_{\mathcal{DK}} \sigma_2$ and $D \models \text{Trans}(\delta, \sigma_1, \alpha, \delta^*, \sigma'_1)$.*

1. *If $\alpha = \text{wait}(\tau_1)$ for some $\tau_1 \in \mathbb{R}_{\geq 0}$, then there is $\tau_2 \in \text{TSUCCS}(\sigma_2, K)$ such that $\text{Trans}(\delta, \sigma_2, \text{wait}(\tau_2), \delta^*, \sigma'_2)$ and $\sigma'_1 \approx_{\mathcal{DK}} \sigma'_2$.*
2. *If $\alpha = A(\vec{\rho})$ for some action type $A \neq \text{wait}$, then $\text{Trans}(\delta, \sigma_2, \alpha, \delta^*, \sigma'_2)$ and $\sigma'_1 \approx_{\mathcal{DK}} \sigma'_2$.*

Proof. First, note that by Lemma 17, for every clocked formula ϕ uniform in s , $D \models \phi[\sigma_1]$ iff $D \models \phi[\sigma_2]$ and so it directly follows that $D \models \text{Final}(\delta, \sigma_1)$ iff $D \models \text{Final}(\delta, \sigma_2)$.

1. Let $\alpha = \text{wait}(\tau_1)$ for some $\tau_1 \in \mathbb{R}_{\geq 0}$. By Proposition 23, there is a $\tau_2 \in \text{TSUCCS}(\sigma_2, K)$ such that $\nu_{\sigma_1} + \tau_1 \cong_{\mathcal{K}} \nu_{\sigma_2} + \tau_2$. Let $\sigma'_2 = \text{do}(\text{wait}(\tau_2), \sigma_2)$. With $D \models \text{Poss}(\text{wait}(t), s) \equiv \top$, it follows that $\text{Trans}(\delta, \sigma_2, \text{wait}(\tau_2), \delta^*, \sigma'_2)$. It remains to be shown that $\sigma'_1 \approx_{\mathcal{DK}} \sigma'_2$. Analogously to Theorem 19:
 - a. Let $R \in \mathcal{F}$ be a relational fluent and $\vec{\rho}'$ a ground tuple of arguments. We show that $D \models R(\vec{\rho}', \sigma'_1)$ iff $D \models R(\vec{\rho}', \sigma'_2)$. There is an SSA for R of the form $R(\vec{o}, \text{do}(a, s)) \equiv \phi_R(\vec{o}, a, s)$, where $\phi_R(\vec{o}, a, s)$ is a clocked formula uniform in s . As $\sigma_1 \approx_{\mathcal{DK}} \sigma_2$, it directly follows from Lemma 17 that $D \models \phi_R(\vec{\rho}', \text{wait}(\tau_1), \sigma_1)$ iff $D \models \phi_R(\vec{\rho}', \text{wait}(\tau_2), \sigma_2)$ and therefore $D \models R(\vec{\rho}', \sigma'_1)$ iff $D \models R(\vec{\rho}', \sigma'_2)$.
 - b. Let c be a func. fluent and $\omega = c(\vec{\rho}')$. By definition of the SSA for c , it follows that $\nu_{\sigma'_1}(\omega) = \nu_{\sigma_1}(\omega) + \tau_1$ and $\nu_{\sigma'_2}(\omega) = \nu_{\sigma_2}(\omega) + \tau_2$. Thus, we can set $f'(\omega, \nu_{\sigma'_1}(\omega)) = (\omega, \nu_{\sigma'_2}(\omega))$. As $\nu_{\sigma_1} + \tau_1 \cong_{\mathcal{K}} \nu_{\sigma_2} + \tau_2$, it follows that f' is a bijection witnessing $\approx_{\mathcal{DK}}$.
2. Let $\alpha = A(\vec{\rho})$ for some action type $A \neq \text{wait}$ and so $\sigma'_1 = \text{do}(\alpha, \sigma_1)$. There is a precondition axiom of the form $\text{Poss}(A(\vec{o}), s) \equiv \phi_A(\vec{o}, s)$ where $\phi_A(\vec{o}, s)$ is a clocked formula uniform in s . As $\text{Trans}(\delta, \sigma_1, \alpha, \delta^*, \sigma'_1)$, it is clear that $D \models \phi_A(\vec{\rho}, \sigma_1)$ and with $\sigma_1 \approx_{\mathcal{DK}} \sigma_2$, also $D \models \phi_A(\vec{\rho}, \sigma_2)$. Therefore, $D \models \text{Poss}(A(\vec{\rho}), \sigma_2)$ and so $D \models \text{Trans}(\delta, \sigma_2, \alpha, \delta^*, \sigma'_2)$ with $\sigma'_2 = \text{do}(\alpha, \sigma_2)$. It remains to be shown that $\sigma'_1 \approx_{\mathcal{DK}} \sigma'_2$. Analogously to Theorem 19:
 - a. Let $R \in \mathcal{F}$ be a relational fluent and $\vec{\rho}'$ a ground tuple of arguments. We show that $D \models R(\vec{\rho}', \sigma'_1)$ iff $D \models R(\vec{\rho}', \sigma'_2)$. There is an SSA for R of the form $R(\vec{o}, \text{do}(a, s)) \equiv \phi_R(\vec{o}, a, s)$, where $\phi_R(\vec{o}, a, s)$ is a clocked formula uniform in s . As $\sigma_1 \approx_{\mathcal{DK}} \sigma_2$, it directly follows from Lemma 17 that $D \models \phi_R(\vec{\rho}', A(\vec{\rho}), \sigma_1)$ iff $D \models \phi_R(\vec{\rho}', A(\vec{\rho}), \sigma_2)$ and therefore $D \models R(\vec{\rho}', \sigma'_1)$ iff $D \models R(\vec{\rho}', \sigma'_2)$.
 - b. We construct a bijection $f' : \nu_{\sigma'_1} \rightarrow \nu_{\sigma'_2}$ witnessing $\approx_{\mathcal{DK}}$. Let c be a functional fluent and $\omega = c(\vec{\rho}')$. There is an SSA of the form $c(\vec{o}, \text{do}(a, s)) = y \equiv \exists t. a = \text{wait}(t) \wedge y = c(\vec{o}, s) + t \vee \neg \exists t. a = \text{wait}(t) \wedge (\phi_c(\vec{o}, a, s) \wedge y = 0 \vee \neg \phi_c(\vec{o}, a, s) \wedge y = c(\vec{o}, s))$. Thus, $\omega[\sigma'_1] = 0$ if $D \models \phi_c(\vec{\rho}, \alpha, \sigma_1)$ and $\omega[\sigma'_1] = \omega[\sigma_1]$ otherwise, similarly for $\omega[\sigma'_2]$. Now, as $\phi_c(\vec{\rho}, a, s)$ is a clocked formula uniform in s , by Lemma 17, $D \models \phi_c(\vec{\rho}, \alpha, \sigma_1)$ iff $D \models \phi_c(\vec{\rho}, \alpha, \sigma_2)$. Hence, for each ω , we can set $f'(\omega, 0) = (\omega, 0)$ if $D \models \phi_c(\vec{\rho}, \alpha, \sigma_1)$ and $f'(\omega, \nu_{\sigma_1}(\omega)) = (\omega, \nu_{\sigma_2}(\omega))$ otherwise. $\blacktriangleleft$

► **Lemma 31.** *Let $\Delta = (\mathcal{D}, \delta)$ be a program over a clocked BAT $\mathcal{D}$ and $\mathcal{T}_\Delta$ the transition system returned by Algorithm 3. Then $\mathcal{T}_\Delta$ contains a state $([\sigma']_{\approx_{\mathcal{DK}}}, \delta')$ reachable from $([S_0]_{\approx_{\mathcal{DK}}}, \delta)$ with $\mathcal{D} \models \text{Final}(\delta', \sigma')$ iff there is a realization of program Δ .*

Proof. We first show that $([\sigma]_{\approx_{\mathcal{DK}}}, \delta')$ is reachable in $\mathcal{T}_\Delta$ iff $\mathcal{D} \models \text{Trans}^*(\delta, S_0, \vec{\alpha}, \delta', \sigma)$.

⇒: Let $p = ([S_0]_{\approx_{\mathcal{DK}}}, \delta) \rightarrow^* ([\sigma']_{\approx_{\mathcal{DK}}}, \delta')$ be a path in $\mathcal{T}_\Delta$. We show by induction on the length n of p that $\mathcal{D} \models \text{Trans}^*(\delta, S_0, \vec{\alpha}, \delta', \sigma')$ for some $\vec{\alpha}$ with $|\vec{\alpha}| = n$:

Base case. Let $n = 0$ and therefore $\sigma' = S_0$ and $\delta' = \delta$. Clearly, $\mathcal{D} \models \text{Trans}^*(\delta, S_0, \langle \rangle, \delta, S_0)$.

Induction step. Let $p = ([S_0]_{\approx_{\mathcal{DK}}}, \delta) \rightarrow^* ([\sigma^*]_{\approx_{\mathcal{DK}}}, \delta^*) \rightarrow ([\sigma']_{\approx_{\mathcal{DK}}}, \delta')$. By induction, $\mathcal{D} \models \text{Trans}^*(\delta, S_0, \vec{\alpha}, \delta^*, \sigma^*)$ for some $\vec{\alpha}$ with $|\vec{\alpha}| = n$. By construction of $\mathcal{T}_\Delta$, $\mathcal{D} \models \text{Trans}(\delta^*, \sigma^*, \alpha, \delta', \sigma')$ for some α and therefore also $\mathcal{D} \models \text{Trans}^*(\delta, S_0, \vec{\alpha} \cdot \alpha, \delta', \sigma')$.

⇐: Let $\mathcal{D} \models \text{Trans}^*(\delta, S_0, \vec{\alpha}, \delta', \sigma')$. We show by induction on the length n of $\vec{\alpha}$ that there is a path $p = ([S_0]_{\approx_{\mathcal{DK}}}, \delta) \rightarrow^* ([\sigma']_{\approx_{\mathcal{DK}}}, \delta')$ in $\mathcal{T}_\Delta$ with length n .

Base case. Let $n = 0$ and therefore $\sigma' = S_0$ and $\delta' = \delta$. Clearly, there is an empty path in $\mathcal{T}_\Delta$ starting in $([S_0]_{\approx_{\mathcal{DK}}}, \delta)$.

Induction step. Let $\mathcal{D} \models \text{Trans}^*(\delta, S_0, \vec{\alpha}, \delta^*, \sigma^*)$ for some $\vec{\alpha}$ with $|\vec{\alpha}| = n$ and let $\mathcal{D} \models \text{Trans}(\delta^*, \sigma^*, \alpha, \delta', \sigma')$. By induction, there is a path $p = ([S_0]_{\approx_{\mathcal{DK}}}, \delta) \rightarrow^* ([\sigma^*]_{\approx_{\mathcal{DK}}}, \delta^*)$ with length n . By Lemma 30, there is a α' such that $\mathcal{D} \models \text{Trans}(\delta^*, \sigma^*, \alpha', \delta', \sigma'')$ and $\sigma'' \approx_{\mathcal{DK}} \sigma'$. Hence, $([\sigma^*]_{\approx_{\mathcal{DK}}}, \delta^*) \rightarrow ([\sigma']_{\approx_{\mathcal{DK}}}, \delta')$ is a transition in $\mathcal{T}_\Delta$.

Now, as $([\sigma]_{\approx_{\mathcal{DK}}}, \delta')$ is reachable in $\mathcal{T}_\Delta$ iff $\mathcal{D} \models \text{Trans}^*(\delta, S_0, \vec{\alpha}, \delta', \sigma)$ for some $\vec{\alpha}$, it follows with Lemma 17 that $\mathcal{D} \models \text{Final}(\delta', \sigma)$ iff $\vec{\alpha}$ is a realization of Δ . ◀