Verifying Behavioral UML Systems via CEGAR*

Yael Meller[1], Orna Grumberg[1], and Karen Yorav[2]

[1]CS Department, Technion, Israel, [2]IBM Research, Haifa, Israel
{ymeller,orna}@cs.technion.ac.il yorav@il.ibm.com

Abstract. This work presents a novel approach for applying abstraction and refinement in the verification of behavioral UML models.
The *Unified Modeling Language* (UML) is a widely accepted modeling language for embedded and safety critical systems. As such the correct behavior of systems represented as UML models is crucial. *Model checking* is a successful automated verification technique for checking whether a system satisfies a desired property. Nevertheless, its applicability is often impeded by its high time and memory requirements. A successful approach to avoiding this limitation is *CounterExample-Guided Abstraction-Refinement (CEGAR)*. We propose a CEGAR-like approach for UML systems. We present a model-to-model transformation that generates an *abstract UML system* from a given concrete one, and formally prove that our transformation creates an *over-approximation*.
The abstract system is often much smaller, thus model checking is easier. Because the abstraction creates an over-approximation we are guaranteed that if the abstract model satisfies the property then so does the concrete one. If not, we check whether the resulting abstract counterexample is *spurious*. In case it is, we automatically *refine* the abstract system, in order to obtain a more precise abstraction.

1 Introduction

This work presents a novel approach for applying abstraction and refinement for the verification of behavioral UML models.

The *Unified Modeling Language* (UML) [2] is a widely accepted modeling language that can be used to visualize, specify, and construct systems. It provides means to represent a system in terms of classes and their relationships, and to describe the systems' internal structure and behavior. UML has been accepted as a standard object-oriented modeling language by the Object Management Group (OMG) [11]. It is becoming the dominant modeling language for embedded and safety critical systems. As such, the correct behavior of systems represented as UML models is crucial and verification techniques applicable to such models are required.

Model checking [6] is a successful automated verification technique for checking whether a given system satisfies a desired property. It traverses *all* of the system behaviors, and either confirms that the system is correct w.r.t. the checked

* This is an extended version including full proofs of [17]

property, or provides a *counterexample (CEX)* that demonstrates an erroneous behavior. Model checking is widely recognized as an important approach to increasing reliability of hardware and software systems and is vastly used in industry.

Unfortunately, the applicability of model checking is impeded by its high time and memory requirements. One of the most successful approaches for fighting these problems is *abstraction*, where some of the system details are hidden. This results in an *over-approximated* system that has *more behaviors and less states* than the concrete (original) system. The abstract system has the feature that if a property holds on the abstract system, then it also holds on the concrete system. However, if the property does not hold, then nothing can be concluded of the concrete system. The *CounterExample-Guided Abstraction Refinement (CEGAR)* approach [5] provides an automatic and iterative framework for abstraction and refinement, where the refinement is based on a spurious CEX. When model checking returns an abstract CEX, a matching concrete CEX is searched. If there exists one, then a real bug on the concrete system is found. Otherwise, the CEX is *spurious* and a refinement is needed. In the refinement stage, more details are added to the abstract system, in order to eliminate the spurious CEX.

In this paper we focus on behavioral systems that rely on *UML state machines*. UML state machines are a standard graphical language for modeling the behavior of event-driven software components. We propose a CEGAR-like framework for verifying such systems. We present a model-to-model transformation that generates an *abstract system* from a given concrete one. Our transformation is done on the UML level, thus resulting in a *new UML behavioral system* which is an *over-approximation* of the original system. We adapt the CEGAR approach to our UML framework, and apply refinement if needed. Our refinement is also performed as a model-to-model transformation. It is important to note that by defining abstraction and refinement in terms of model-to-model transformations, we avoid the translation to lower level representation (such as Kripke structures). This is highly beneficial to the user, since both the property, the abstraction, and the abstract CEX are given at the UML level and are therefore more meaningful.

Our abstraction is obtained by abstracting some (or all) of the state machines in the concrete system. When abstracting a state machine, we over-approximate its *interface behavior* w.r.t. the rest of the system. In the context of behavioral UML systems, an interface includes the events generated/consumed and the (non-private) variables. We thus abstract part of the system's variables, and maintain an *abstract view* of the events generated by the abstracted state machines. In particular, the abstract state machines may change the number and order of the generated events. Further, abstracted variables are assigned the "don't-know" value. Our abstraction does not necessarily replace an *entire* state machine. Rather, it enables abstracting *different parts* of a state machine whose behavior is irrelevant to the checked property. Our abstraction construction is presented in section 4.

We show that the abstract system is an over-approximation by proving that for every concrete system computation there exists an abstract system computation that "behaves similarly". This is formally defined and proven in section 5. To formalize the notion of *system computation*, we present in section 3 formal semantics for behavioral UML systems that rely on state machines. Works such as [7, 10, 15] also give formal semantics to state machines, however they all differ from our semantics: e.g. [7] defines the semantics on flat state machines and present a translation from hierarchical to flat state machines, whereas we maintain the hierarchical structure of the state machines. [10] define the semantics of a *single* state machine. Thus it neither addresses the semantics of the full system, nor the communication between state machines. [15] addresses the communication of state machines, however their notion of run-to-completion step does not enable context switches during a run-to-completion step. Our formal semantics is defined for a *system*, possibly multi-threaded, where the atomicity level is a transition execution (formally defined later).

Our CEGAR framework is suitable for verifying LTL_x, which is the Linear-time Temporal Logic (LTL)[22] without the next-time operator. Also, we assume the existence of a model checker for behavioral UML systems. Extensive work has been done in the last years in providing such model checkers by translating the system into an input language of some model checker. [3, 4] present translation of state machines to SMV. Several works [18, 14, 21, 1, 8] translate state machines to PROMELA, which is the input language of the model checker SPIN. A verification environment for UML behavioral models was developed in the context of the European research project OMEGA [20], and works such as [25, 19] apply different methods for model checking these models. [12, 16] translate a UML behavioral model into C code, and apply bounded model checking via CBMC. As mentioned before, we add the special value "don't-know" to the domain of the variables. This results in a 3-valued semantics for UML systems, as shown in section 4. To model check abstract systems we need a 3-valued model checker. Extending a model checker to support the 3-valued semantics (e.g., [27, 13]) is straightforward.

Many works such as [26, 28, 24, 9, 23] address *semantic refinement* of state machines, which is adding details to a partially defined state machine while preserving behavior of the original (abstracted) model. Though we also address an abstraction-refinement relation between state machines, these works are very different from ours. These works look at manual refinement as part of the modeling process, whereas we are suggesting an automatic abstraction to improve scalability of the verification tool. Moreover, these works handle a single state machine level, where we consider a system which includes possibly many state machines that interact with each other. To the best of our knowledge, this is the first work that addresses the abstraction for a behavioral UML system at the UML level.

2 Preliminaries - UML Behavioral Systems

Behavioral UML systems include objects (instances of classes) that process events. Event processing is defined by state machines, which include complex features such as hierarchy, concurrency and communication. UML objects communicate by sending each other events (asynchronous messages) that are kept in *event queues* (EQs). Every object is associated with a single EQ, and several objects can be associated with the same EQ. In a multi-threaded system there are several EQs, one for each thread. Each thread executes a never-ending loop, taking an event from its EQ, and dispatching it to the target object. The target object makes a *run-to-completion (RTC) step*, where it processes the event and continues execution until it cannot continue anymore. Only when the target object finishes its RTC step, the thread dispatches the next event available in its EQ. Next we formally define state machines, UML systems, and the set of behaviors associated with them. The following definitions closely follow the UML2 standard.

2.1 UML State Machines

Definition 21 (States and Regions) *Let S denote a set of states partitioned into disjoint subsets according to two types: simple states S_{sim} and compound states S_{com}. Let R be a non-empty set of regions. We assume R contains the region TOP. Let $\Omega : S \cup R \to S \cup R \cup \{\epsilon\}$ be a function that associates regions to their containing states, and states to their containing regions. We assume the following constraints on Ω:*

- *For every $s \in S$, $\Omega(s) \in R$ (the container of a state is a region).*
- *For every $r \in R$ if $r = TOP$ then $\Omega(r) = \epsilon$, otherwise $\Omega(r) \in S$ (the container of a region is a state and TOP has no container).*
- *For every $r \in R$ s.t. $r \neq TOP$, $\Omega(r) \in S_{com}$ (only compound states contain regions)*
- *For every $r \in R$ there exists at least one $s \in S$ such that $\Omega(s) = r$*
- *The transitive closure of Ω is irreflexive*

The function Ω induces a partial order on $S \cup R$: $u \lhd u'$ denotes that u' *contains* u.

We say that two different regions $r_1, r_2 \in R$ are *orthogonal*, denoted $ORTH(r_1, r_2)$, if they are contained in the same state. Formally, $ORTH(r_1, r_2) = true$ iff $r_1 \neq r_2$ and $\Omega(r_1) = \Omega(r_2)$.

From here on we assume a fixed set V of variables over finite domains. We use Σ to denote the set of all possible valuations for the variables in V, and σ or σ_i to denote specific assignments. We use $\mathcal{B}$ to denote the set of Boolean expressions over V. We also assume a fixed set of environment events EV_{env} and a fixed set of system events EV_{sys}, and we denote $EV = EV_{env} \cup EV_{sys}$. An event e is a pair $(type(e), trgt(e))$, where $type(e)$ denotes the event name (or type), and $trgt(e)$ denotes the state machine to which the event was sent (formally defined later).

Definition 22 (Actions) *An action is a sequence of statements in some programming language. A simple statement is either an assignment "$x = e$" over variables in V, or "$GEN(e)$", which is the generation of an event from EV_{sys}. skip represents an empty sequence of statements. A compound statement is a sequence of statements, "$a_1; a_2$" or a branching statement "if b then a_1 else a_2", for actions a_1 and a_2 and $b \in \mathcal{B}$.*

Given an action *act*, we denote by $modif(act)$ the set of variables that may be modified on *act*. Formally, $x \in modif(act)$ if statement "$x = e$" is part of *act*. By abuse of $modif(t)$ denotes the set of variables that may be modified on $act(t)$.

Definition 23 (State Machines) *A* state machine *is a tuple* $(S, R, \Omega, init, Trans, L, \mathcal{H})$ *such that:*

- *S, R, and Ω are the sets of states and regions and the Ω function, as defined above.*
- *$init \subseteq S$ are initial states, such that there is exactly one initial state in each region.*
- *$TR \subseteq S \times S$ is the set of transitions. Each transition t connects a single source state $src(t)$ with a single target state $trgt(t)$.*
- *$L : TR \rightarrow EV \times \mathcal{B} \times Actions$ is a function that labels each transition with a trigger (an event from EV), a guard, and an action. Since none of these components are mandatory we assume $\epsilon \in EV$ representing no trigger, $true \in \mathcal{B}$ representing an empty guard, and $skip \in Actions$ representing no action. We use $trig(t)$, $grd(t)$, and $act(t)$ to refer to the trigger, guard, and action of t respectively.*
- *$\mathcal{H} \subseteq R$ is the history marker, marking those regions that have history (these would have a history pseudostate in them in the graphical representation).*

Transitions t where $trig(t) = \epsilon$ and $grd(t) = true$ are referred to as *null transitions*.

Figure 1 describes the state machine of class DB. States are denoted as squares. Regions are graphically represented only if they are orthogonal. Orthogonal regions are denoted by a dashed line. For example, state $Working$ contains two orthogonal regions. A transition t is denoted with $tr[g]/a$ where $tr = trig(t)$, $g = grd(t)$ and $a = act(t)$. If $tr = \epsilon$, $g = true$ or $a = skip$ they are omitted from the representation.

Definition 24 (State Machine Configurations) *Let $SM = (S, R, \Omega, init, Trans, L, \mathcal{H})$ be a state machine. An* SM-configuration *is a tuple (ω, ρ, H) such that:*

- *$\omega \subseteq S$ is the set of currently active states. ω has the property that for every $s \in \omega$ and for every $r \in R$ such that $\Omega(r) = s$ there exists a* single *$s' \in S$ such that $\Omega(s') = r$ and $s' \in \omega$.*
- *$\rho \in type(EV) \cup \{\epsilon\}$ holds an event currently* dispatched *(formally defined later) to the state machine and not yet consumed (and ϵ if there is no event to be consumed).*

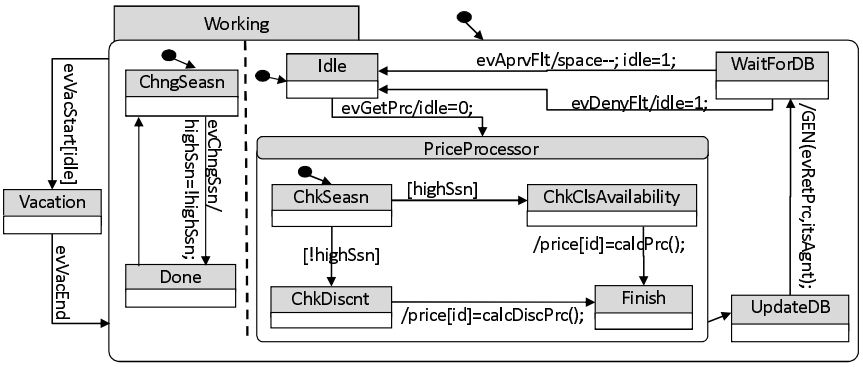

Fig. 1. DB State Machine

– $H : R \to S$ *is the history information. It records the last active state in each region marked with history ($r \in \mathcal{H}$), or the initial state if either the region has not yet been visited or the region is not marked with history.*

From here on, we assume the following restrictions on SM:

1. An action includes at most one "$GEN(e)$". In addition, an action that includes "$GEN(e)$" is a non-branching sequence of statements. If either one of these restrictions does not hold, then SM can be preprocessed s.t. the transition is replaces with a series of states and transitions, each executing part of the original action.
2. SM does not include complex UML syntactic features: cross-hierarchy transitions, fork, join, entry and exit actions. It is straightforward to eliminate these features, at the expense of additional states, transitions and variables. Note that the hierarchical structure of the state machines is maintained, thus avoiding the exponential blow-up incurred by flatenning.

2.2 Systems

Next we define UML systems and their behavior. UML2 places no restrictions on the implementation of the event queue and neither do we. We use a finite sequence $q = (e_1, ..., e_l)$ of events $e_i \in EV$ to represent the contents of an event queue at a particular point in time (thus the set of all possible values for an event queue is EV^*). We assume the functions $pop(q)$, $top(q)$, and $push(q, e)$ are defined in the usual way.

Definition 25 (System) *A* system *is a tuple* $(SM_1, ..., SM_n, Q_1, ..., Q_m, thread, V)$ *s.t.* $SM_1, ..., SM_n$ *are state machines,* $Q_1, ..., Q_m$ *($m \leq n$) are event queues (one for each thread),* $thread : \{1, ..., n\} \to \{1, ..., m\}$ *assigns each state machine to a thread, and* V *is a collection of variables over finite domains.*

Note that in the original UML model variables are partitioned into private attributes, public attributes, and global variables. These definitions govern the constraints on which variables each state machine may read or write to. For the semantic model we bundle all variables together into a single vector V and assume that all accesses are legal.

Definition 26 (System Configuration) *Let $\Gamma = (SM_1, ..., SM_n, Q_1, ..., Q_m, thread, V)$ be a system. A* system-configuration *is a tuple $(c_1, ..., c_n, q_1, ..., q_m, id_1, ..., id_m, \sigma)$ such that:*

- *c_i is an SM-configuration of SM_i*
- *q_j is the contents of Q_j*
- *$id_j \in \{0, ..., n\}$ is the id of the state machine associated with thread j that is currently executing a run-to-completion step. $id_j = 0$ means that all the state machines of thread j are* inactive.
- *σ is an assignment giving each variable in V a value from its legal domain.*

From now on we fix a given system $\Gamma = (SM_1, ..., SM_n, Q_1, ..., Q_m, thread, V)$. We use lower case c for SM-configurations and capital C for Γ-configurations. We use k as a superscript to range over steps in time, making c_i^k the SM-configuration of SM_i at time k. For every $e \in EV$, we define $trgt(e) \in \{0, ..., n\}$ to give the index of the state machine that is the target of e. $trgt(e) = 0$ means the event is sent to the environment of Γ.

Next we define computations of a system. In principle, a computation is a series of transitions fired according to certain constraints and following the run-to-completion semantics per-thread. The main difference between our definition and the majority of formal semantic theories suggested for UML state machines is that we differentiate between the extraction of an event from the event queue and the state machine transition that is fired as a result of this event being dispatched.

In order to define computations we require few more definitions.

Definition 27 (Enabled Transition) *A transition t of a state machine SM_i is* enabled *in a configuration $C = (c_1, ..., c_n, q_1, ..., q_m, id_1, ..., id_m, \sigma)$ (where $c_i = (\omega_i, \rho_i, H_i)$), denoted $enabled(t, C)$, if the following conditions hold:*

- *$src(t) \in \omega_i$ (the source state of t is active)*
- *$trig(t) = \rho_i$ (the trigger is the currently dispatched event, or no trigger on the transition if $\rho_i = \epsilon$)*
- *$\sigma \models grd(t)$ (the guard of the transition is satisfied under the current assignment to variables)*
- *For every $t' \in Trans_i$ such that $src(t') \in \omega_i$ and $src(t') \lhd src(t)$: $trig(t') \neq \rho_i$ or $\sigma \not\models grd(t')$ (a transition is enabled if all transitions from states contained in $src(t)$ are not enabled)*

We say that a state machine configuration c_i is *stable* in a configuration $C = (c_1, ..., c_n, q_1, ..., q_m, id_1, ..., id_m, \sigma)$ if there are no enabled transitions in SM_i.

Definition 28 (Transition Execution on state machine) *When a transition t of a state machine SM_i in state machine configuration $c_i = (\omega_i, \rho_i, H_i)$ is executed, SM_i moves to a new state machine configuration $c'_i = (\omega'_i, \rho'_i, H'_i)$, marked $dest(c_i, t)$, which is defined as follows:*

- $\omega'_i = (\omega_i \backslash \{s \in \omega_i | s = src(t) \vee s \lhd src(t)\}) \cup \{s \in S | s = trgt(t) \vee (s \lhd trgt(t) \wedge s = H_i(\Omega(s)) \wedge \forall s' \in S : s \lhd s' \lhd trgt(t) \rightarrow s' = H_i(\Omega(s')))\}$ *(ω'_i is obtained by removing from ω_i states contained in $src(t)$ and then adding states contained in $trgt(t)$, based on the history).*
- $\rho'_i = \epsilon$ *(an event is consumed once)*
- *For every region $r \in R_i$ where $r \in \mathcal{H}_i$: If there exists $s \in S_i$ s.t. $s \in \omega'_i$ and $\Omega(s) = r$ then $H'_i(r) = s$ (if region r is an active region that has history marker, then we update the history according to the current state). Otherwise, $H'_i(r) = H_i(r)$.*

Let C be a Γ-configuration, SM_i be a state machine in Γ, and let $s_1, s_2 \in S_i$ and $t, t_1, ..., t_y \in TR_i$. We will further use the following notations:

- $Qpush(t, (q_1, ..., q_m)) = (q'_1, ..., q'_m)$ denotes the effect of executing transition t on the different queues of the system; if for some event e, $GEN(e) \in act(t)$, then executing t pushes e to the relevant event queue (to $Q_{thread(trgt(e))}$). The rest of the event queues remain unchanged.
- $act(t)(\sigma, C) = \sigma'$ represents the effect of executing the assignments in $act(t)$ on the valuation σ of C, which results in a new assignment, σ'.
- $ORTH(s_1, s_2)$ is *true* if the states are contained in orthogonal regions, and $false$ otherwise. Formally, $ORTH(s_1, s_2) = true$ iff $\exists r_1, r_2 \in R_i$ s.t. $ORTH(r_1, r_2)$ and for $k \in \{1, 2\}$: $s_k \lhd r_k$.
- $ORTH(t_1, ..., t_y)$ is *true* iff $t_1, ..., t_y$ are pairwise orthogonal. I.e., for every $k, l \in \{1, ..., y\}$ s.t. $k \neq l$: $ORTH(src(t_k), src(t_l))$.
- $maxORTH((t_1, ..., t_y), C)$ is *true* iff $(t_1, ..., t_y)$ is a *maximal set* of *enabled* orthogonal transitions. Formally $maxORTH((t_1, ..., t_q), C) = true$ iff (1) for every $i \in \{1, ..., q\}$, $enabled(t_i, C)$, and (2) $ORTH(t_1, ..., t_y)$, and (3) there is no $t \in Trans_i$ such that $enabled(t, C)$ and $ORTH(t, t_1, ..., t_q)$. Note that for some configuration C and state machine SM_i there can be several *different* sets transitions for which $maxORTH$ is *true*.

Definition 29 (Transition Execution on system) *Let $C = (c_1, ..., c_n, q_1, ..., q_m, id_1, ..., id_m, \sigma)$ be a configuration on Γ, and let $t_1, ..., t_q \in Trans_i$ (possibly $q = 1$) be a set of transitions. $apply((t_1, ..., t_q), C) = C'$ represents the effect of executing t_1 of C followed by t_2 on the result etc. until executing t_y, which results in configuration $C' = (c_1, ..., c'_i, ..., c_n, q'_1, ..., q'_m, id_1, ..., id_m, \sigma')$ defined as follows:*

- $c'_i = dest(...dest(dest(dest(c_i, t_1), t_2), t_3)..., t_q)$
- $\sigma' = act(t_q)(...act(t_3)(act(t_2)(act(t_1)(\sigma, C), C), C)..., C)$
- $q'_1, ..., q'_m = Qpush(t_q, (...Qpush(t_3, (Qpush(t_2, (Qpush(t_1, (q_1, ..., q_m))))))...))$

3 System Computations

Definition 31 (System Computations) *A* computation *of a system Γ is a maximal sequence $C^0, step^0, C^1, step^1, ...$ such that: (1) each C^k is a Γ-configuration, (2) each step $C^k \xrightarrow{step^k} C^{k+1}$ can be generated by one of the inference rules detailed below, and (3) each $step^k$ is a pair $(thid^k, t^k)$ where $thid^k \in \{1, ..., m\}$ represents the id of the thread executing the step (t^k is described in the inference rules).*

We now define the set of inference rules describing $C \xrightarrow{step} C'$. We specify only the parts of C' that change w.r.t. C due to *step*.

Initialization In the initial configuration all event queues are empty, and the state machines are in their initial state and are inactive. Formally: C^0 is the initial system configuration, such that for every j: $q_j^0 = \phi$ and $id_j^0 = 0$. c_i^0 is the initial configuration of SM_i ($\rho_i^0 = \epsilon$ and $\omega_i^0 = \{s \in S_i | s \in init_i \wedge \forall s' \in S_i.s \lhd s' \rightarrow s' \in init_i\}$),

Dispatch An event can be dispatched from thread j's event queue only if the processing of the previous event on thread j has terminated (i.e. the run-to-completion step ended) and the queue is not empty. A dispatch step pops the event out of the thread's queue and places it in the target object's ρ element. It also updates the corresponding id_j^{k+1} with the index of the state machine that is the target of the event. Formally:

$$DISP(j,e): \quad \frac{id_j = 0 \quad q_j \neq \phi \quad top(q_j) = e \quad trgt(e) = l}{id'_j = l \quad q'_j = pop(q_j) \quad c'_l = (\omega_l, type(e), H_l)}$$

Transition A transition can be fired if it is enabled and the state machine containing it is currently executing a RTC step. If the state machine's ρ element is not empty then the fired transition has ρ as its trigger. After firing the transition ρ is set to ϵ (so that an event cannot be consumed twice). There is a single case where more than one transition can be fired together. It is the case where transitions are in orthogonal regions and several transitions simultaneously consume the event (the state machine's ρ element is not empty). UML2 defines a simultaneous execution of the transitions in this case. Since it is not clear how to define simultaneous execution of actions, we define an interleaved execution of these transitions. Only after all transitions have executed, the next step is enabled. Note that the transitions are executed according to their order in the $TRANS$ step (t_1 executed first, followed by t_2 etc.). However, since the step itself can be defined with any order of transitions, then if from a given configuration $step = TRANS(j, (t_1, ..., t_q))$ is possible, then also $step = TRANS(j, (t'_1, ..., t'_q))$ is possible for any permutation $(t'_1, ..., t'_q)$ of $(t_1, ..., t_q)$. Formally:

$$TRANS(j,(t_1,...,t_y)): \quad \frac{\begin{array}{c} id_j = l > 0 \quad t_1,...,t_y \in TR_l \\ \rho_l \neq \epsilon \rightarrow (maxORTH((t_1,...,t_y),C) = true) \\ \rho_l = \epsilon \rightarrow (y = 1 \wedge enabled(t_1,C)) \end{array}}{C' = apply((t_1,...,t_y),C)}$$

EndRTC If the currently running state machine on thread j is stable, then the RTC step is complete, id_j is set to zero, and the ρ element of the state machine that finished the RTC is cleared. Formally:

$$EndRTC(j,\epsilon): \quad \frac{id_j = l > 0 \quad stable(c_l,C)}{id'_j = 0 \quad c'_l = (\omega_l,\epsilon,H_l)}$$

ENV The behavior of the environment is not precisely described in the UML standard. We assume the most general definition, where the evnironment may insert events into the event queues at any step. Formally:

$$ENV(j,e): \quad \frac{e \in EV_{env} \quad thread(trgt(e)) = j}{q'_j = push(q_j,e)}$$

Let π be a computation. A run-to-completion (RTC) step w.r.t. π on thread j is a maximal sequence of $TRANS$ steps of state machine i where $thread(i) = j$, s.t. the $TRANS$ steps appear between a $DISP$ step (initiating the RTC step) and a $EndRTC$ step (terminating the RTC step). Note that between each $DISP$ step and its following $EndRTC$ step on thread j, the currently active state machine remains the same (the value of id_j does not change).

Definition 32 (Run-to-Completion Steps) *Given a computation $\pi = C^0, step^0, C^1, step^1, ...,$ a* run-to-completion (RTC) step *is a maximal series of steps $\chi = step^{i_0}, step^{i_1}, ..., step^{i_d}$ where for every $r \in \{1,...,d\}$: $i_{r-1} < i_r$, and for some thread j the following holds:*

- *$step^{i_0} = DISP(j,e)$*
- *For every $r \in \{1,...,d-1\}$: $step^{i_r} = TRANS(j,(t_1,...,t_q))$*
- *$step^{i_d} = EndRTC(j,\epsilon)$*
-
- *For every $r \in \{i_0, i_0+1, ..., i_d\}$ s.t. $r \neq i_0, i_1, ..., i_d$: if $step^r$ is on thread j, then $step^r == ENV(j,e')$ (for some event e'). This item ensures the maximality of χ.*

4 Abstract State Machines

We now present the construction of abstract UML systems.

4.1 Abstracting a State Machine

The abstraction of a state machine $SM = (S, R, parent, init, Trans, L, \mathcal{H})$ is defined w.r.t. a disjoint *abstraction collection* $ABS = \{A^1, ..., A^g\} \subseteq 2^S$. Every A^β is referred to as an *abstraction set*. Every abstraction set A^β is a set of states (simple or composite) for which the following holds. (1) for every $s, s' \in A^\beta$: $\Omega(s) = \Omega(s')$, and (2) for every $s \in A^\beta$ and $s' \in A^\gamma$, if $\beta \neq \gamma$ then $s \not\lhd s'$ and $s' \not\lhd s$. The first requirement states that an abstraction set A^β includes states from a single region. The second requirement states that the abstraction sets are disjoint: no two states in different abstraction sets s.t. one state contains the other state. Intuitively, out abstraction replaces every A^β (and all states contained in A^β) with a *different construct* that ignores the details of A^β and maintains an over-approximated behavior of the events generated by A^β.

We add the value $don't{-}know$, denoted $\perp$, to the domain of all variables in V, where $\perp$ represents any value in the domain. The semantics of boolean operations is extended to 3-valued logic as follows: $\perp \wedge false = false$, $\perp \wedge true = \perp$ and $\neg\perp = \perp$. An expression is evaluated to $\perp$ if one of its arguments is $\perp$. For simplicity of presentation, we enable $trig(t)$ to be a *set of triggers*. I.e. $trig(t) = \{e_1, ..., e_q\} \cup \epsilon$, and $enabled(t, C) = true$ if *one of the events* from $trig(t)$ matches ρ.

Next, we define several notions that are concrete and are defined w.r.t. an abstraction set A^β:

- $S(A^\beta) = \{s \in S | \exists s' \in A^\beta.(s \lhd s')\}$ are the *abstracted states*.
- $R(A^\beta) = \{r \in R | \exists s \in A^\beta.(r \lhd s)\}$ are the *abstracted regions*.
- $TR(A^\beta) = \{t \in TR | src(t), trgt(t) \in S(A^\beta)\}$ are the *abstracted transitions*.
- $EV(A^\beta) = \{e \in EV | \exists t \in TR(A^\beta).(GEN(e) \in act(t))\}$.
- $Trig(A^\beta) = \{tr | \exists t \in TR(A^\beta).(trig(t) = tr)\} \setminus \{\epsilon\}$.
- $V(A^\beta) = \{v \in V | \exists t \in TR(A^\beta).(v \in modif(t))\}$.
- $GRDV(A^\beta) = \{v \in V | \exists t \in TR(A^\beta).(trig(t) = \epsilon \wedge v \in grd(t))\}$.

We require the following restrictions of A^β of SM_i:

1. For every $v \in GRDV(A^\beta)$, if v can be modified by several state machines in Γ, then all these state machines are assigned to the same thread. Formally: if $v \in modif(Trans_i) \cap modif(Trans_j)$ then $thread(i) = thread(j)$. This is needed for correctness of the construction (details in the proof of theorem 56.
2. For every $t \in TR(A^\beta)$, if $trgt(t) \neq \epsilon$ then $GEN(e) \notin act(t)$.
3. There are no loops without triggers within $S(A^\beta)$. Further, there are no self loops without a trigger on states containing $S(A^\beta)$. This is needed to enable static analysis described next.

In order to explain our abstraction we introduce the notion of an A-round on A^β, which is a maximal, possibly non-consecutive, sequence of steps from a computation π, s.t. all the steps are part of a single RTC, every step executes an abstracted transition, and the state machines remains in an abstracted state throughout the A-round. Formally,

Definition 41 (A-Round) *Let A^β be an abstraction set of state machine SM_i, and let $\tau = step^{i_0}, ..., step^{i_d}$ be a RTC step of system Γ on SM_i. An* A-round *is a maximal sub-sequence of τ, $\tau^A = step^{i_{j_1}}, ..., step^{i_{j_k}}$ s.t. the following holds:*

(a) For every $j \in \{j_1, j_2, ..., j_k\}$: $step^{i_j} = TRANS(j, (t_1, ..., t_y))$ (possibly $y = 1$), and there exists a transition $t \in \{t_1, ..., t_y\}$ s.t. $t \in TR(A^\beta)$ (the step executes an abstracted transition), and
(b) For every $\iota \in \{i_{j_1}, (i_{j_1}) + 1, ..., i_{j_k}\}$, $\omega_i^\iota \cap S(A^\beta) \neq \phi$ (the state machine remains in the abstracted states throughout τ^A.

Since there are no loops without triggers that include abstracted states, we can easily apply static analysis in order to determine the maximal number of evetns that can be generated by any single A-round. We denote this number by f.

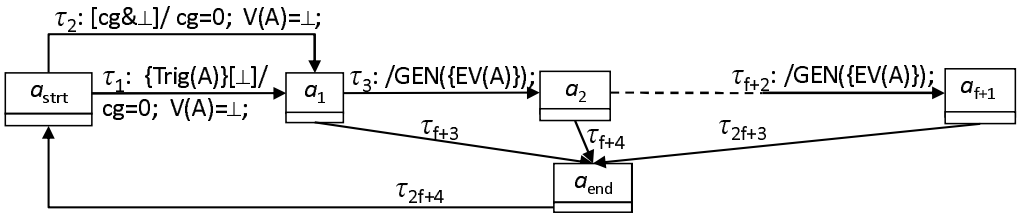

Fig. 2. $\Delta(A)$: The abstraction construct created for abstraction set A

Given an abstraction set $A \in ABS$, our abstraction replaces $S(A)$,$R(A)$ and $trans(A)$ with *a new construct*, referred to as $\Delta(A)$, demonstrated in Figure 2. $\Delta(A)$ includes an initial state a_{strt} and a final state a_{end}. Every A-round over states from $S(A)$ is represented by a computation from a_{strt} to a_{end}. $\Delta(A)$ includes computations that can generate any sequence of size 0 to f events from $EV(A)$. also, all the variables that *can* be modified in the A-round are given the value $\perp$.

An A-round whose first transition consumes an event, is represented by a computation that starts with transition τ_1 from a_{strt} to a_1, which can consume any single event from $Trig(A)$. The guard $\perp$ on τ_1 and τ_2 represents a non-deterministic choice between "true" or "false". If the first transition on an A-round does not consume an event, it will be represented by transition τ_2, which is not marked with a trigger. Since $\Delta(A)$ contains a loop of transitions without triggers we must ensure that all RTCs through $\Delta(A)$ are finite. We introduce a new Boolean variable cg. A trace on $\Delta(A)$ can be initiated without a trigger only if cg is 1. $\Delta(A)$ then sets cg to 0 on the transitions exiting a_{strt}.

When cg is set to 1 it signals that it is possible to execute an A-round that does not consume an event. Such a situation abstracts a concrete execution in which the RTC step that includes the A-round starts at a state that is not abstracted and continues within the abstraction. I.e., execution of a transition t

whose source is outside of $S(A)$ and whose target is a state s that either contains A or $s \in A$. The situation can also occur if an abstracted transition becomes enabled due to some variable change. I.e., execution of some transition t, which is either orthogonal to A or is in a different state machine, and t modifies a variable v and $v \in GRDV(A)$.

If by static analysis we can conclude that the first transition of every A-round *consumes an event*, then cg is redundant (and τ_2 can be removed). All the A-rounds are then represented by computations that start by traversing τ_1.

We now formally define our abstract state machines. Given $SM = (S, R, \Omega, init, TR, L)$ and an abstraction set $A \in ABS$, $SM(A) = (S^A, R^A, \Omega^A, init^A, TR^A, L^A)$ is the abstraction of SM w.r.t. A. We denote functions over the abstraction (src, $trgt$, $trig$, grd, and act) with a superscript A. The following definition enables us to define several different abstraction over a concrete state machine, by defining them one after the other. Given an abstraction collection $ABS = \{A^1, ..., A^g\}$, the abstraction of SM w.r.t. ABS is defined as $SM^A = (((SM(A^1))(A^2))....)(A^g)$.

- $S^A = (S \setminus S(A)) \cup \{a_{strt}, a_1, ..., a_{f+1}, a_{end}\}$
- $R^A = (R \setminus R(A))$
- For every $s \in (S^A \cap S) \cup R^A$: $\Omega^A(s) = \Omega(s)$.
  For every $s \in \{a_{strt}, a_1, ..., a_{f+1}, a_{end}\}$: $\Omega^A(s) = \Omega(s')$ for some $s' \in A$.
- $init^A = init \cap S^A$ or $init^A = (init \cap S^A) \cup \{a_{strt}\}$ if $\exists s \in A$ s.t. $s \in init$
- $TR^A = (TR \setminus TR(A)) \cup \{\tau_1, ..., \tau_{2f+5}\}$. The src^A, $trgt^A$, $trig^A$, grd^A and act^A functions are redefined as follows:
  Transitions $\tau_1, ..., \tau_{2f+5}$ are defined according to Fig. 2. Every transition $t \in TR \setminus TR(A)$ has a representation (*matching transition*) in $SM(A)$. Note that for every such transition, either $src(t)$ or $trgt(t)$ are not abstracted (are in $S \cap S^A$). If $src(t)$ or $trgt(t)$ are abstracted, then $src^A(t)$ or $trgt^A(t)$ respectively are in $\Delta(A)$. The handling of cg is added to the relevant actions, as discussed above. In the following we present only the the values of src^A, $trgt^A$, $trig^A$ grd^A and act^A that change in $SM(A)$ w.r.t. SM. For every $t \in TR \setminus TR(A)$:
  1. $trgt(t) \in S(A)$ (the target of t is abstracted): we define $trgt^A(t) = a_{strt}$. If there exists an abstracted transition from $trgt(t)$ whose trigger is ϵ then $act^A(t) = act(t); cg = 1$ (otherwise, $act^A(t) = act(t)$). This describes the case that the RTC can start outside the abstraction and continue within the abstraction.
  2. $src(t) \in S(A)$ (the source of t is abstracted): we define $src^A(t) = a_{strt}$, $act^A(t) = cg = 0; act(t)$ and $grd^A(t) = grd(t)\&\bot$. We add $\bot$ to the guard in order to ensure that executions of possibly enabled transitions from states containing the abstraction remain (possibly) enabled.
  3. Otherwise (neither $src(t)$ nor $trgt(t)$ are abstracted):
     Case a: $A \lhd trgt(t)$: If an execution of t results in a new ω that includes an abstracted state $s \in S(A)$, and there exists an abstracted transition from s whose trigger is ϵ. Then $act^A(t) = act(t); cg = 1$ (otherwise $act^A(t) = act(t)$).

Case b: $src(t)$ and a_{strt} are contained in orthogonal regions (t can be executed orthogonally to the abstraction): Then $act^A(t) = act(t)$ with the following modifications:
If $\exists v \in GRDV(A)$ s.t. $v \in modif(t)$ then $cg = 1$ is added to $act^A(t)$. In addition, if SM is in an abstracted state, then variables that *can* be modified by abstracted transitions should remain $\perp$. For that, every assignment $x = e$ in $act(t)$, if $x \in V(A)$ then $x = e$ is replaced with: "if $(isIn(A))$ $x = \perp$; else $x = e$;" in $act^A(t)$. The current state is checked using the macro $isIn(U)$, that checks whether a certain state from U is active.

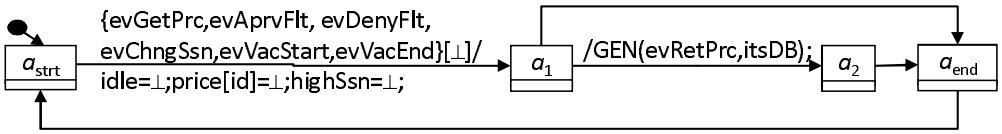

Fig. 3. Abstract DB State Machine

Fig. 3 shows the state machine created by abstracting the DB state machine (Fig. 1) with $A = \{Working, Vacation\}$. Note that in this state machine, by static analysis we can conclude that every A-round first consumes an event, and therefore we do not need the cg flag and transition τ_2. Also, on every A-round no more than one event can be generated, therefore $f = 1$.

4.2 Abstracting a System

Next we define an abstract system. This is a system in which some of the state machines are abstract. For SM_i and an abstraction collection $ABS_i = \{A^1, ..., A^{g_i}\}$, SM_i^A denotes the abstraction of SM_i w.r.t. ABS_i. We denote the cg variable in SM_i^A that was added when abstracting w.r.t. abstraction set A^β as cg_i^β.

Definition 42 *Let Γ and Γ' be two system, each with n state machines and m event queues. We say that Γ' is an* abstraction *of Γ, denoted Γ^A, if the following holds:*

1. *For $i \in \{1, ..., n\}$, $SM_i' = SM_i$ or $SM_i' = SM_i^A$*
2. *$thread = thread'$*
3. *$V' = V \cup \{cg_i^\beta | SM_i' = SM_i^A$ and $A^\beta \in ABS_i\}$*
4. *For every $i, j \in \{1, ..., n\}$ s.t. $i \neq j$, and for every $t \in TR_j'$: if there exists a variable $v \in GRDV(A^\beta)$ where $A^\beta \in ABS_i$, and $v \in modif(t)$ then $cg_i^\beta = 1$ is added to $act'(t)$ (in SM_j').*

Recall that setting cg_i^β to 1 on SM_i^A signals that it is possible to execute an A-round on SM_i that does not consume an event. Requirement (4) in Def. 42 handles the case where a guard of an abstracted transition of SM_i changes by a transition t of SM_j, by ensuring that cg_i^β is set to 1 on such transitions of TR'_j.

Adding the value $\perp$ to the domain of all variables in V affects the cases when a transition is enabled, and when a state machine is stable, since now $grd(t)(\sigma) \in \{true, false, \perp\}$. Intuitively, if $grd(t)(\sigma) = \perp$ then we assume it *can* be either $true$ or $false$. We thus consider both cases in the analysis. Therefore, $enabled(t, C) = true$ iff t *can be enabled* w.r.t. C and all transitions from states contained in $src(t)$ *can be not enabled*. Similarly, $stable(c_i, C)$ if c_i *can* be stable in C. I.e., for every $t \in TR_i$, s.t. $src(t) \in \omega_i$, either $trig(t) \neq \rho_i$ or $grd(t)(\sigma) \in \{false, \perp\}$.

We formally re-define when a transition is enabled and when a state machine is stable. For a boolean expression g, we denote by $g(\sigma)$ the value of g under the assignment σ to variables.

Definition 43 (Enabled Transition) *A transition t of a state machine SM_i is* enabled *in a configuration $C = (c_1, ..., c_n, q_1, ..., q_m, id_1, ..., id_m, \sigma)$ (where $c_i = (\omega_i, \rho_i, H_i)$) if the following conditions hold:*

- *$src(t) \in \omega_i$ (the source state of t is active)*
- *$trig(t) = \rho_i$ (the trigger is the currently dispatched event, or no trigger on the transition if $\rho_i = \epsilon$)*
- *$grd(t)(\sigma) \in \{true, \perp\}$ (the guard of the transition is evaluated to either true or $\perp$))*
- *For every $t' \in Trans_i$ such that $src(t') \in \omega_i$ and $src(t') \lhd src(t)$: $trig(t') \neq \rho_i$ or $grd(t')(\sigma) \in \{false, \perp\}$ (a transition is enabled if all transitions from states contained in $src(t)$ are either not enabled or enabled with value $\perp$ on the guard)*

We say that a state machine configuration c_i is *stable* in a configuration C if for every $t \in Trans_i$ s.t. $src(t) \in \omega_i$, $trig(t) \neq \rho_i$ or $grd(t)(\sigma) \in \{false, \perp\}$. Note that when enabling 3-valued semantics, a transition might be enabled, even though lower level transitions are enabled as well. This is not possible under the 2-valued semantics. Note also that in the 3-valued context it still holds that for a SM-configuration c_i, if there exists a transition $t \in TR_i$ s.t. $src(t) \in \omega_i$, $trig(t) = \epsilon$ and $grd(t) = true$, then c_i is not stable.

We will further use the following lemma, which captures the fact that when an event is dispatched on some thread j, then for all of the state machines associated with thread j: if the state machine is in an abstract state, then that state can only be a^k_{strt}.

Lemma 1. *Let $\pi = C^0, step^0, C^1, step^1, ...$ be some computation. For every $1 \leq j \leq m$, for every r s.t. $step^r = DISP(j, e)$, and for every l s.t. $thread(l) = j$: If $A^k \in ABS_l$ then $\{a^k_1, ..., a^k_{f+1}, a^k_{end}\} \cap \omega^r_l = \phi$.*

Proof. Assume that the above does not hold. This means that for some r, $step^r = DISP(j, e)$ and there exists l s.t. $thread(l) = j$, and one of $(\Delta(A^k) \setminus$

$\{a^k_{strt}\}) \in \omega^r_l$. We differentiate between two cases, the first case is when state machine l did not make any transition until reaching configuration C^r, and the second case is that state machine l did make some transition.

- If for every $r' < r$, if $step^{r'} = TRANS(j, (t_1, ..., t_q))$ then $id^{r'}_j \neq l$: This means that for every $s \in \omega^r_l$, $s \in init_l$ (current state is updated only on $TRANS$ steps). Since none of $\Delta(A^k) \setminus \{a^k_{strt}\}$ are in the initial configuration, then the lemma holds.
- If there exists some $r' < r$ for which $step^{r'} = TRANS(j, (t_1, ..., t_q))$ and $id^{r'}_j = l$: By definition of inference rules, there exists some $r' < r$ for which $step^{r'} = EndRTC(j, \epsilon)$ and $id^{r'}_j = l$. Since an $EndRTC$ cannot occur when there exists an enabled null transition, then if $\Delta(A^k) \cap \omega^{r'}_l \neq \phi$ then $a^k_{strt} \in \omega^{r'}_l$. Consider the greatest such r'. By definition of the inference rules, there cannot be a $TRANS$ steps on SM_l between $step^{r'}$ and $step^r$. Also, by the inference rules, only $TRANS$ steps on SM_l can change the current state of SM_l. We can then conclude that for every $r'' \in \{r', ..., r\}$: if $\Delta(A^k) \cap \omega^{r''}_l \neq \phi$ then $a^k_{strt} \in \omega^{r''}_l$.

□

The following lemma is needed later. It states that during an $EndRTC$ step, if the state machine that finished its RTC step is in an abstract state, then that state can only be a^k_{strt} (for $k \in \{1, ..., g\}$).

Lemma 2. *Let $\pi = C^0, step^0, C^1, step^1, ...$ be some computation. For every r s.t. $step^r = EndRTC(j, \epsilon)$, if $id^r_j = l$ then for every $k \in \{1, ..., g\}$: if $\Delta(A^k) \cap \omega^r_l \neq \phi$ then $a^k_{strt} \in \omega^r_l$.*

5 Correctness of The Abstraction

In this section we prove that Γ^A is an over-approximation of Γ by showing that every computation of Γ has a "matching" computation in Γ^A.

Definition 51 (Abstraction relation of SM-configuration) *Let $c = (\omega, \rho, H)$ and $c^A = (\omega^A, \rho^A, H^A)$ be SM-configurations of a state machine SM and its abstraction SM^A respectively. c^A* abstracts *c, denoted $c \preceq c^A$, if the following holds:*

- *$\rho = \rho^A$*
- *c, c^A agree on the joint states: $\omega \neq \omega^A$ iff $\omega \setminus \omega^A \subseteq S(A)$ and $\omega^A \setminus \omega \subseteq \Delta(A)$.*
- *H, H^A agree on their history: for every $r \in R^A$: $H(r) \neq H^A(r)$ iff for some $A^\beta \in ABS$, $H^A(r) \in A^\beta$ and $H(r) \in S(A^\beta)$.*

Definition 52 (Abstraction relation σ) *Let σ and σ' be variable assignments over V of Γ and V' of Γ^A respectively. We say that σ' abstracts σ, denoted $\sigma \preceq \sigma'$ if for every $v \in V$ either $\sigma(v) = \sigma'(v)$ or $\sigma'(v) = \perp$.*

Definition 53 (Abstraction relation of Γ-configuration) *Let C and C' be two system configurations of Γ and Γ^A respectively. We say that C'* abstracts *C, denoted $C \preceq C'$, if C and C' agree on the event queues and id elements, and the state machine configurations and σ' of Γ^A are abstraction of the matching elements in Γ:*

- *For $j \in \{1, ..., m\}$: $q_j = q'_j$ and $id_j = id'_j$*
- *For $i \in \{1, ..., n\}$: $c_i \preceq c'_i$*
- *$\sigma \preceq \sigma'$*

We will further need the following lemma, stating that when executing two matching transitions t and t_a from two computations C and C', where C' is an abstraction of C, then the resulting variable assignments are related.

Lemma 3. *Let C and C' be Γ-configurations of Γ and Γ^A respectively, such that $C \preceq C'$. For every $l \in \{1, ..., n\}$, for every $t \in Trans_l$ and $t_a \in Trans_l^A$: if t_a matches t then $act(t)(\sigma, C) \preceq act(t_a)(\sigma', C')$.*

Proof. To show that $act(t)(\sigma, C) \preceq act(t)(\sigma', C')$ we have to show that for every $v \in V$: either $act(t)(\sigma, C)(v) = act(t_a)(\sigma', C')(v)$ or $act(t_a)(\sigma', C')(v) = \perp$.

Since $C \preceq C'$, then for every variable $v \in V$, either $\sigma(v) = \sigma'(v)$ or $\sigma'(v) = \perp$. Note that by the definition of matching transitions, $modif(t) = modif(t_a) \cap V$. For every $v \in V$:

- If $v \notin modif(t)$ then $v \notin modif(t_a)$. Therefore $act(t)(\sigma, C)(v) = \sigma(v)$ and $act(t_a)(\sigma', C')(v) = \sigma'(v)$, and clearly the requirement holds.
- If $v \in modif(t)$: If $act(t_a)(\sigma', C')(v) \neq \perp$ then the value of v is determined by an evaluation of an expression over V for variables whose value is not $\perp$. These variables have the same value in σ, and the evaluating expression is the same. Therefore, $act(t)(\sigma, C)(v) = act(t_a)(\sigma', C')(v)$. Otherwise, $act(t_a)(\sigma', C')(v) = \perp$, and clearly the requirement holds.

□

We now define *stuttering computation inclusion*, which is an extension of stuttering-trace inclusion ([6]) to system computations. For simplicity of presentation, we assume from now on that computations are infinite. however, all the results presented hold for finite computations as well. Intuitively, there exists stuttering inclusion between π and π' if they can be partitioned into infinitely many finite intervals, s.t. *every* configurations in the kth interval of π' abstracts *every* configuration in the kth interval of π, and vice versa.

Definition 54 (Stuttering Computation Inclusion) *Let $\pi = C^0, step^0, C^1, step^1, ...$ and $\pi' = C'^0, step'^0, C'^1, step'^1, ...$ be two computations over Γ and Γ^A respectively. There exists a* stuttering computation inclusion *between π and π', denoted $\pi \preceq_s \pi'$, if there are two infinite sequences of integers $0 = i_0 < i_1 < i_2 < ...$ and $0 = i'_0 < i'_1 < i'_2 < ...$ such that for every $k \geq 0$:*
For every $j \in \{i_k, ..., (i_{k+1}) - 1\}$ and for every $j' \in \{i'_k, ..., (i'_{k+1}) - 1\}$: $C^j \preceq C'^{j'}$

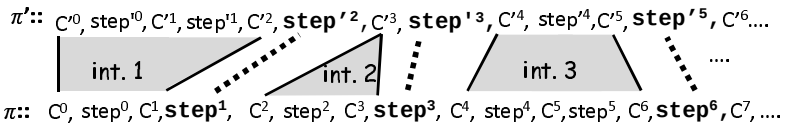

Fig. 4. Stuttering Computation Inclusion

Note that corresponding intervals in π and π' may have different lengths.

Fig. 4 illustrates two computations where $\pi \preceq_s \pi'$. Definition 53 implies that steps of type $DISP$, ENV and $EndRTC$ cannot be steps within an interval, due to the effect of these steps on Γ-configuration. For example, in Figure 4, $C^6 \preceq C'^5$. Assume $step^6 = EndRTC(j, \epsilon)$, then by the definition of $EndRTC$ step, the value of id_j changes from C^6 to C^7. Since Γ-configuration abstraction requires equality of the id elements, then clearly $C^7 \not\preceq C'^5$. Thus C^6 and C^7 cannot be in the same interval. For a similar reason, a step of type $DISP$, ENV or $EndRTC$ on π implies a step of the same type on π', and vice versa. Steps of type $TRANS$ that are either the first step in a RTC or a step that generates events are also steps that cannot be part of an interval, due to the effect of these steps on the ρ elements and the event queues.

The above is captured in the following lemma.

Lemma 4. *Let $\pi = C^0, step^0, C^1, step^1, ...$ and $\pi' = C'^0, step'^0, C'^1, step'^1, ...$ be two computations over Γ and Γ^A respectively, s.t. $\pi \preceq_s \pi'$. Let $0 = i_0 < i_1 < i_2 < ...$ and $0 = i'_0 < i'_1 < i'_2 < ...$ be two infinite sequences of positive integers describing the intervals of the stuttering inclusion. Then for every $k \geq 0$:*

- *$step^{i_k} = DISP(j, e)$ iff $step'^{i'_k} = DISP(j, e)$*
- *$step^{i_k} = ENV(j, e)$ iff $step'^{i'_k} = ENV(j, e)$*
- *$step^{i_k} = EndRTC(j, \epsilon)$ iff $step'^{i'_k} = EndRTC(j, \epsilon)$*
- *$step^{i_k} = TRANS(j, (t_1, ..., t_y))$ where $id_j^{i_k} = l$ and $\rho_l^{i_k} \neq \epsilon$ iff $step'^{i'_k} = TRANS(j, (t'_1, ..., t'_{y'}))$ where $id_j^{i'_k} = l$ and $\rho_l^{i'_k} \neq \epsilon$*
- *$step^{i_k} = TRANS(j, (t_1, ..., t_y))$ where $id_j^{i_k} = l$ and $GEN(e) \in act(t)$ for some $t \in \{t_1, ..., t_y\}$ iff $step'^{i'_k} = TRANS(j, (t'_1, ..., t'_{y'}))$ where $id_j^{i'_k} = l$ and $GEN(e) \in act(t')$ for some $t' \in \{t'_1, ..., t'_{y'}\}$*

An immediate result of the above is that an interval can be of size greater than one only if the steps in the interval are $TRANS$ steps that are neither a first step in a RTC nor a step generating an event. Recall that Definition 53 requires a correlation between the current states of the state machines. It can therefore be shown (for a similar reason as above) that if $step^i = TRANS(j, (t))$ is a step inside an interval, i.e. between two configurations in the same interval, then one the following holds: (1) If $step^i \in \pi$ then t is an abstracted transition, (2) If $step^i \in \pi'$ then $t \in \Delta(A)$.

We extend the notion of stuttering inclusion to systems, and say that there exists a *stuttering inclusion* between Γ and Γ^A, denoted $\Gamma \preceq_s \Gamma^A$, if for each

computation π of Γ from an initial configuration C_{init}, there exists a computation π' of Γ^A from an initial configuration C'_{init} s.t. $\pi \preceq_s \pi'$.

Every system Γ can be viewed as a Kripke structure K, where the K-states are the set of Γ-configurations, and there exists a K-transition (C, C') iff C' is reachable from C within a single *step*. Thus, every computation of Γ corresponds to a trace in K. Let Γ be a system, and let $A\psi$ be an LTL formula, where the atomic propositions are predicates over Γ. Then $\Gamma \models A\psi$ iff for every computation π of Γ from an initial configuration, $\pi \models \psi$. By preservation of LTL_x over stuttering traces we conclude:

Corollary 55 *Let Γ and Γ^A be two systems, s.t. $\Gamma \preceq_s \Gamma^A$, and let $A\psi$ be an LTL_x formula over joint elements of Γ and Γ^A. If $\Gamma^A \models A\psi$ then $\Gamma \models A\psi$.*

Due to the stuttering-inclusion, Γ^A preserves LTL_x and not LTL. It is important to note that since Γ itself is a multi-threaded system, properties of interest are most often defined without the next-time operator.

The following theorem captures the relation between Γ and Γ^A, stating that there *exists* stuttering inclusion between Γ and Γ^A.

Theorem 56 *If Γ^A is an abstraction of Γ then $\Gamma \preceq_s \Gamma^A$.*

Proof. For simplicity of the proof, we assume the following restrictions on the system Γ w.r.t. its abstraction Γ^A.

1. Transitions with trigger have *skip* in their action. Formally, for every $i \in \{1, ..., n\}$ and for every $t \in TR_i$, if $trig(t) \neq \epsilon$ then $act(t) = skip$.
2. There is no history on the states. We assume that the state machines are preprocessed and history is removed.

Assume a computation $\pi = C^0, step^0, C^1, step^1, ...$ on Γ s.t. C^0 is an initial configuration. We prove by induction on the number of steps in π that there exists a computation $\pi' = C'^0, step'^0, C'^1, step'^1, ...$ on Γ^A s.t. $\pi \preceq_s \pi'$.
Base: Given $C^0 = (c_1, ..., c_n, q_1, ..., q_m, id_1, ..., id_m, \sigma)$, the initial configuration of π. We define the following initial configuration for π': $C'^0 = (c'_1, ..., c'_n, q'_1, ..., q'_m, id'_1, ..., id'_m, \sigma')$ and show that it is an initial configuration on Γ^A.

- For every $1 \leq i \leq n$ c'_i is defined as follows. For every $s \in c_i$:
  - If $s \in S_i^A$ then $s \in c'_i$ (if state s from SM_i exists also in SM_i^A, then it is part of c'_i)
  - If $s \notin S_i^A$ and $s \in A^\beta$ (s is part of the abstracted states A^β) then $a^\beta_{strt} \in c'_i$
- For every $1 \leq j \leq m$ $q'_j = q_j = \phi$ ($q_j = \phi$ since C^0 is an initial configuration)
- For every $1 \leq j \leq m$ $id'_j = id_j = 0$ ($id_j = 0$ since C^0 is an initial configuration)
- For every $v \in V$, $\sigma'(v) = \sigma(v)$
- For every $cg_i^\beta \in V^A$:
  - If there exists $s \in c_i$ s.t. $s \in S(A^\beta)$ and there exists $t \in TR(A^\beta)$ s.t. $src(t) = s$, $trig(t) = \epsilon$ and $grd(t)(\sigma^0) = true$ then $cg_i^\beta = 1$.

- Otherwise, $cg_i^\beta = 0$.

The above captures the case that an abstracted transition on SM_i can be executed without consumption of an event, and without a modification of some variable effecting its guard. This situation can occur if the guard is $true$ under the initial configuration. In this case we initialize the matching cg_i^β to 1.

It is immediate to see that C'^0 is an initial configuration and also that $C^0 \preceq C'^0$

Step: We assume that for the first r steps of π: $C^0, step^0, C^1, step^1, ..., C^{r-1}, step^{r-1}, C^r$ $(r > 0)$ there exists a partial computation $\pi' = C'^0, step'^0, C'^1, step'^1, ..., C'^p$ over Γ^A s.t. there are two sequences of positive integers $0 = i_0 < i_1 < i_2 < ... < i_l = r$ and $0 = i'_0 < i'_1 < i'_2 < ... < i'_l = p$ and for every $0 \leq k < l$, $C^{i_k}, C^{(i_k)+1}, ..., C^{(i_{k+1})-1} \preceq C'^{i'_k}, C'^{(i'_k)+1}, ..., C'^{(i'_{k+1})-1}$, and also $C^r \preceq C'^p$.

We define the matching extension of π' based on $step^r$:

- $step^r = DISP(j, ev)$

  By definition, $id_j^r = 0$, $q_j^r \neq \phi$ and $top(q_j^r) = ev$. Since $C^r \preceq C'^p$, then $id_i^r = id_i'^p$ and $q_i^r = q_i'^p$ for every i. Therefore, $id_j'^p = 0$, $q_j'^p \neq \phi$ and $top(q_j'^p) = ev$ as well, and it is possible to make a step where $step'^p = DISP(j, ev)$ from C'^p.

  By definition of $DISP$ step, $C^{r+1} = (c_1^r, ..., c_n^r, q_1^r, ..., q_j^{r+1}, ..., q_m^r, id_1^r, ..., id_j^{r+1}, ...id_m^r, \sigma^r)$, $q_j^{r+1} = pop(q_j^r)$, $id_j^{r+1} = trgt(ev)$ and $\rho_{trgt(ev)}^{r+1} = type(ev)$.

  By definition of $DISP$ step, $C'^{p+1} = (c_1'^p, ..., c_n'^p, q_1'^p, ..., q_j'^{p+1}, ..., q_m'^p, id_1'^p, ..., id_j'^{p+1}, ...id_m'^p, \sigma'^p)$, $q_j'^{p+1} = pop(q_j'^p)$, $id_j'^{p+1} = trgt(ev)$ and $\rho_{trgt(ev)}'^{p+1} = type(ev)$.

  Since $C^r \preceq C'^p$, it is clear that $C^{r+1} \preceq C'^{p+1}$ as well.

- $step^r = EndRTC(j, \epsilon)$

  Assume $id_j^r = id_j'^p = l > 0$. Since $stable(c_l^r, C^r)$, then for every $t \in Trans_l$ s.t. $src(t) \in \omega_l^r$ either $trig(t) \neq \rho_l^r$ or $grd(t)(\sigma^r) = false$.

  For every $s \in \omega_l'^p$:

  1. If $s \in \{a_1^k, ..., a_{f+1}^k, a_{end}^k\}$: If $\rho_l'^p \neq \epsilon$, then by definition of the semantics there exists $step'^{p'} = DISP(j, ev)$ where $id_j'^{p'} = l$ (only $DISP$ steps can change the ρ element to non-ϵ). Also, for every $p' < p'' < p$, if $step'^{p''} = TRANS(j, (t_1, ..., t_q))$ then $id_j'^{p''} \neq l$ (otherwise, the $TRANS$ step would have set the ρ element to ϵ. This means that $s \in \omega_l'^{p'}$ (since only $TRANS$ steps can change the current state of a state machine). However, it is not possible that $s \in \omega_l'^{p'}$, since there exists a null transition from s (by definition of $\Delta(A)$). Therefore, we can conclude that $\rho_l'^p = \epsilon$.

     For $s \in \{a_1^k, ..., a_{f+1}^k\}$ $(s = a_{end}^k)$: There exists an enabled transition t' s.t. $trgt(t') = a_{end}^k$ (a_{strt}^k) (this is a null transition). We define $step'^p = TRANS(j, (t'))$. Clearly, if $C_l^r \preceq C_l'^p$, then also $C_l^r \preceq C_l'^{p+1}$. Note that we match C'^{p+1} to C^r and not to C^{r+1}. We prove stuttering simulation, and this step of π' is part of the matching interval, continuing in the handling of a_{end}^k (a_{strt}^k).

2. if $s = a^k_{strt}$ then for every t' s.t. $src(t') = a^k_{strt}$, by construction of SM^A_i, $grd(t')(\sigma) \in \{\bot, false\}$ (for any σ), since $\bot$ is included in $grd(t)$.
3. Otherwise, since $c^r_l \preceq c'^p_l$, then $s \in \omega^r_l$. By definition of SM^A_l, for every transition $t \in TR_l$ s.t. $src(t) = s$ there exists a transition $t_a \in TR^A_l$ s.t. $src(t_a) = s$, $trig(t) = trig(t_a)$ and $grd(t) = grd(t_a)$. Thus, if $trig(t) \neq \rho^r_l$, then $trig(t_a) \neq \rho'^p_l$, and if $grd(t)(\sigma^r) = false$ then $grd(t_a)(\sigma'^p) \in \{false, \bot\}$.

The above means that for cases (2) and (3), for every $t_a \in TR^A_l$ s.t. $src(t_a) \in \omega'^p_l$ either $trig(t_a) \neq \rho'^p_l$ or $grd(t_a)(\sigma'^p) \in \{false, \bot\}$. Therefore, an $EndRTC$ step is possible s.t. $step'^p = EndRTC(j, \epsilon)$, and clearly $C^{r+1} \preceq C'^{p+1}$.

– $step^r = ENV(j, ev)$.
Since the environment is always enabled, then an ENV step s.t. $step'^p = ENV(j, ev)$ is possible from C'^p, and clearly $C^{r+1} \preceq C'^{p+1}$.

– $step^r = TRANS(j, \{t_1, ..., t_q\})$.
Assume $id^r_j = id'^p_j = l$. By definition of the semantics, $\rho^r_l = \rho'^p_l \neq \epsilon$. Thus, there exists $r' < r$ s.t. $step^{r'} = DISP(j, ev)$ and $id^{r'}_j = l$ (only $DISP$ steps can set ρ to a value not ϵ) and there are no $TRANS$ or $EndRTC$ steps on thread j between $step^{r'}$ and $step^r$ (since these steps set the value of ρ to ϵ). From lemma 4 we know that there exists a matching step $step'^{p'} = DISP(j, ev)$ where $id'^{p'}_j = l$. From lemma 4 we also know that there are no $TRANS$ or $EndRTC$ steps on thread j between $step'^{p'}$ and $step'^p$ (if there was such a $TRANS$ step, then it had to be the first step in the RTC, in which case it had to have a matching step on π between $step^{r'}$ and $step^r$).
From lemma 1 we know that none of $a^k_1, ...a^k_{f+1}, a^k_{end}$ are in $\omega'^{p'}_l$. Since only $TRANS$ steps on thread j can change the current state of state machine SM^A_l, then we can conclude that none of $a^k_1, ...a^k_{f+1}, a^k_{end}$ are in ω'^p_l.
For every $i \in \{1, ..., q\}$ we match transition t_i with a transition $t^a_i \in Trans^A_l$.
Assume $s = src(t_i)$.
1. If $src(t_i) \in \omega'^p_l$ and for every $A^k \in ABS_l$, $a^k_{strt} \not\lhd s$: For every $s' \in \omega'^p_l$ s.t. $s' \lhd s$, and for every $t'_a \in Trans^A_l$ s.t. $src(t'_a) = s'$, since s does not contain any abstracted state then $s' \in \omega^r_l$. Consider the transition $t' \in Trans_l$ that matches t'_a: $trig(t') = trig(t'_a)$ and $grd(t') = grd(t'_a)$. Since $enabled(t_i, C^r) = true$ and $src(t') \lhd src(t_i)$ then $enabled(t', C^r) = false$. Thus either $trig(t') \neq \rho^r_l$ or $grd(t')(\sigma^r_l) = false$. Since $\rho'^p_l = \rho^r_l$ then $trig(t'_a) \neq \rho'^p_l$ as well. Since $\sigma^r \preceq \sigma'^p$ then $grd(t'_a)(\sigma'^p) \in \{false, \bot\}$. Therefore, we conclude that $enabled(t'_a, C'^p) \in \{false, \bot\}$.
By definition of the abstraction, there exists $t^a_i \in Trans^A_l$ that matches t_i. Thus, $src(t^a_i) = s$, $trig(t^a_i) = trig(t_i)$ and $grd(t^a_i) = grd(t_i)$. For similar reasons, since $trig(t_i) = \rho^r_l$ and $grd(t_i) = true$ then $trig(t^a_i) = \rho'^p_l$ and $grd(t^a_i) \in \{\bot, true\}$.
Therefore, since $enabled(t_i, C^r) = true$ then $enabled(t^a_i, C'^p) \in \{\bot, true\}$. Since $trig(t_i) \neq \epsilon$, then $act(t_i) = skip$. By definition of the abstraction $act(t^a_i) = skip$ as well. Therefore, since $\sigma^r \preceq \sigma'^p$ then $act(t_i)(\sigma^r, C^r) \preceq act(t^a_i)(\sigma'^p, C'^p)$

2. If $s \in \omega_l'^p$ and for some $A^k \in ABS_l$, $a^k_{strt} \lhd s$: By definition of the abstraction, there exists a transition $t^a_i \in Trans^A_l$ that matches t_i. Thus, $src(t^a_i) = src(t_i)$, $trig(t^a_i) = trig(t_i)$ and $grd(t^a_i) = grd(t_i)$.
   For every transition $t'^a_i \in Trans^A_l$ s.t. $src(t'^a_i) = s' \in \omega_l'^p$ and $s' \lhd s$:
   (a) If $s' = a^k_{strt}$ then by definition of the abstraction structure $grd(t'^a_i) = \bot$ or $grd(t'^a_i) = grd\&\bot$. Therefore $enabled(t'^a_i, C'^p) = \bot$.
   (b) If $s' \neq a^k_{strt}$ then by definition of the simulation, $s' \in \omega^r_l$, and by the definition of the abstraction there exists $t'_i \in Trans_l$ that matches t'^a_i. Thus, $src(t'_i) = src(t'^a_i)$, $trig(t'_i) = trig(t'^a_i)$ and $grd(t'_i) = grd(t'^a_i)$. Since $enabled(t'_i, C^r) = false$ (otherwise t_i is not enabled), and since $C^r \preceq C'^p$ then $enabled(t'^a_i, C^p) \in \{false, \bot\}$ as well.
   From the above we conclude that $enabled(t^a_i, C'^p) \in \{\bot, true\}$.
   Since $trig(t_i) \neq \epsilon$, then $act(t_i) = skip$. By definition of the abstraction $modif(t^a_i) \subseteq \{cg^1_l, ..., cg^g_l\}$. Therefore, if $\sigma \preceq \sigma'$ then $act(t_i)(\sigma, C^r) \preceq act(t^a_i)(\sigma', C'^p)$
3. If $s \notin \omega_l'^p$: Notice that if $s \notin \omega_l'^p$, and since $c^r_l \preceq c'^p_l$, then $s \notin S^A_l$. This means that the transition t_i taken is from an abstracted state s.
   (a) if $s \in S_a(A^k)$ and $trgt(t_i) \in S^A_l$: This means that the transition taken is from an abstracted state $s \in S_a(A^k)$ (for $A^k \in ABS_l$) and the target is not an abstracted state.
   Since $enabled(t_i, C^r) = true$ and $\rho^r_l \neq \epsilon$, then $trig(t_i) \neq \epsilon$. By definition of the abstraction there exists a matching transition $t^a_i \in Trans^A_l$. Thus, $src(t^a_i) = a^k_{strt}$, $trig(t^a_i) = trig(t_i)$, $grd(t^a_i) = grd(t_i)\&\bot$, and $trgt(t^a_i) = trgt(t_i)$. Since $C^r \preceq C'^p$, then if $enabled(t_i, C^r) = true$ then $enabled(t^a_i, C'^p) = \bot$.
   Since $trig(t_i) \neq \epsilon$, then $act(t_i) = skip$. By definition of the abstraction $modif(t^a_i) = \{cg^k_l\}$. Therefore, if $\sigma \preceq \sigma'$ then $act(t_i)(\sigma, C^r) \preceq act(t^a_i)(\sigma', C'^p)$
   (b) if $s \in S_a(A^k)$, $trgt(t_i) \in S_a(A^{k'})$ ($k \neq k'$ and $A^k, A^{k'} \in ABS_l$): This means that the transition taken is from an abstracted state $s \in S_a(A^k)$ and the target is an abstracted state $s' \in S_a(A^{k'})$.
   Since $enabled(t_i, C^r) = true$ and $\rho^r_l \neq \epsilon$, then $trig(t_i) \neq \epsilon$. By definition of the abstraction there exists a transition $t^a_i \in Trans^A_l$ s.t. $src(t^a_i) = a^k_{strt}$, $trig(t^a_i) = trig(t_i)$, $grd(t^a_i) = grd(t_i)\&\bot$, and $trgt(t^a_i) = a^{k'}_{strt}$. Since $C^r \preceq C'^p$, then if $enabled(t_i, C^r) = true$ then $enabled(t^a_i, C'^p) = \bot$.
   Since $trig(t_i) \neq \epsilon$, then $act(t_i) = skip$. By definition of the abstraction $modif(t^a_i) = \{cg^k, cg^{k'}\}$. Therefore, if $\sigma \preceq \sigma'$ then $act(t_i)(\sigma, C^r) \preceq act(t^a_i)(\sigma', C'^p)$
   (c) Otherwise ($s, trgt(t_i) \in S_a(A^k)$): This means that the the transition t_i taken is a transition within the abstraction of A^k. Since $enabled(t_i, C^r) = true$ and $\rho^r_l \neq \epsilon$, then $trig(t_i) \neq \epsilon$. By definition of the abstraction, this means that $\rho^r_l \in Trig(A^k)$. By the definition of the abstraction, $modif(t_i) \subseteq V(A^k)$. Consider transition $\tau^k_1 \in \Delta(A^k)$. It holds that $enabled(\tau^k_1, C'^p) = \bot$. We define $t^a_i = \tau^k_1$.
   Since $trig(t_i) \neq \epsilon$, then $act(t_i) = skip$. By definition of the abstrac-

tion, for every $v \in V(A^k)$, $act(t_1^k)(\sigma', C'^p)(v) = \perp$. Therefore, we can conclude that if $\sigma \preceq \sigma'$ then $act(t_i)(\sigma, C^r) \preceq act(t_a^i)(\sigma', C'^p)$.

Note that there can be *several* transitions from $(t_1, ..., t_y)$ that are abstracted by a *single* τ_1^k. A single occurrence of τ_1^k replaces all of these transitions.

We define $step'^p = TRAN S(j, (t_1^a, ..., t_{y'}^a))$: If there are several transitions $t_{i_1}, ..., t_{i_d}$ executed in the current $TRANS$ step for which $t_{i_j}^a = \tau_1^k$ (for a specific $A^k \in ABS_l$), then τ_1^k replaces the first occurrence of abstracted transition in $(t_1, ..., t_q)$. Since the first execution of τ_1^k sets all variables in $V(A^k)$ to $\perp$, then the effect of executing only the first occurrence of τ_1^k σ^p is the same as executing several occurrences of τ_1^k. Since for every $t \in \{t_1, ..., t_q\}$, $trig(t') \neq \epsilon$ then $act(t) = skip$, then this step does not change the event queues. We can conclude that $C^{r+1} \preceq C'^{p+1}$.

– $step^r = TRAN S(j, (t))$.

Assume $id_j^r = id_j'^p = l$, and $src(t) = s$. By definition of the semantics, $\rho_l^r = \rho_l'^p = \epsilon$. We want to show that if $enabled(t, C^r)$, then there exists $t' \in Trans_l'$ s.t. $enabled(t', C'^p)$. We then define $step'^p = TRAN S(j, (t'))$ and show that $C^{r+1} \preceq C'^{p+1}$. We separate the proof to different cases, described here:

1. For every $A^k \in ABS_l$, $\Delta(A^k) \cap \omega_l'^p = \phi$

Otherwise (for some $A^k \in ABS_l$, $\Delta(A^k) \cap \omega_l'^p \neq \phi$), and:

2. $s \in \omega_l'^p$ and for every $A^k \in ABS_l$, $\neg(a_{strt}^k \lhd s)$
3. $s \in \omega_l'^p$, for some $A^k \in ABS_l$, and for some $i \in \{1, ..., f+1\}$: $a_i^k \in \omega_l'^p$, and $a_i^k \lhd s$
4. $s \in \omega_l'^p$, and for some $A^k \in ABS_l$, $a_{end}^k \in \omega_l'^p$ and $a_{end}^k \lhd s$
5. $s \in \omega_l'^p$, for every $A^k \in ABS_l$, if $\Delta(A^k) \cap \omega_l'^p \neq \phi$ and $a_{strt}^k \lhd s$ then $a_{strt}^k \in \omega_l'^p$ (possibly more than one such k)
6. $s \notin \omega_l'^p$, for some $A^k \in ABS_l$, $s \in S(A^k)$, for some $i \in \{1, ..., f+1\}$, $a_i^k \in \omega_l'^p$, and $trgt(t) \notin S(A^k)$
7. $s \notin \omega_l'^p$, for some $A^k \in ABS_l$, $s \in S(A^k)$, $a_{end}^k \in \omega_l'^p$, and $trgt(t) \notin S(A^k)$
8. $s \notin \omega_l'^p$, for some $A^k \in ABS_l$, $s \in S(A^k)$, $a_{strt}^k \in \omega_l'^p$, and $trgt(t) \notin S(A^k)$
9. $s \notin \omega_l'^p$, for some $A^k \in ABS_l$, $s \in S(A^k)$, $a_{strt}^k \in \omega_l'^p$, and $trgt(t) \notin S(A^k)$
10. $s \notin \omega_l'^p$, for some $A^k \in ABS_l$, $s \in S(A^k)$, for some $i \in \{1, ..., f+1\}$: $a_i^k \in \omega_l'^p$, and $trgt(t) \notin S(A^k)$
11. $s \notin \omega_l'^p$, for some $A^k \in ABS_l$, $s \in S(A^k)$, $a_{f+1}^k \in \omega_l'^p$, and $trgt(t) \notin S(A^k)$
12. $s \notin \omega_l'^p$, for some $A^k \in ABS_l$, $s \in S(A^k)$, $a_{end}^k \in \omega_l'^p$, and $trgt(t) \notin S(A^k)$

1. For every $A^k \in ABS_l$, $\Delta(A^k) \cap \omega_l'^p = \phi$: This means that the current state of SM_l^A does not include any abstraction.
   Since $c_l^r \preceq c_l'^p$ then $\omega_l'^p = \omega_l^r$. $enabled(t, C^r) = true$, therefore for every $s' \in \omega_l^r$ s.t. $s' \lhd s$, and for every $t' \in Trans_l$ s.t. $src(t') = s'$, either $trig(t') \neq \rho_l^r$ or $grd(t')(\sigma) = false$.
   For every $s' \in \omega_l'^p$ where $s' \lhd s$, it holds that $s' \in \omega_l^r$ as well (since $\omega_l'^p = \omega_l^r$). By definition of the abstraction, for every $t_a' \in Trans_l^A$ s.t. $src(t_a') = s'$ there exists a matching $t' \in Trans_l$ where $src(t') = s'$. Also, by definition of the abstraction $trig(t_a') = trig(t')$ and $grd(t_a') = grd(t')$.

Since $\rho_l^r = \rho_l'^p$ and $\sigma^r \preceq \sigma'^p$, then either $trig(t'_a) \neq \rho_l'^p$ or $grd(t'_a)(\sigma'^p) \in \{false, \bot\}$.
By definition of the abstraction, there exists a matching $t_a \in Trans_l^A$ s.t. $src(t_a) = s$, $trig(t_a) = trig(t)$ and $grd(t_a) = grd(t)$. For similar reasons, since $trig(t) = \rho_l^r$ and $grd(t) = true$ then $trig(t_a) = \rho_l'^p$ and $grd(t_a) \in \{\bot, true\}$.
Therefore, since $enabled(t, C^r)$ then also $enabled(t_a, C'^p)$.
Thus, $step^p = TRANS(j, (t_a))$ is possible from C'^p.
By definition of the abstraction, $act(t_a)$ either equals to $act(t)$, or equals to $act(t)$ with addition of manipulation of cg variables. Thus, by lemma 3, and since $\sigma^r \preceq \sigma'^p$, $act(t)(\sigma^r, C^r) \preceq act(t_a)(\sigma'^p, C'^p)$. We can then conclude that $C^{r+1} \preceq C'^{p+1}$.

For some $A^k \in ABS_l$, $\Delta(A^k) \cap \omega_l'^p \neq \phi$, and:

2. $s \in \omega_l'^p$ and for every $A^k \in ABS_l$, $\neg(a_{strt}^k \lhd s)$: This means that the current state of SM_l^A includes the abstraction, and the transition taken is from a state s which is in an orthogonal region to each of the abstractions. For the same reasoning as in the previous item, there exists $t_a \in Trans_l^A$ s.t. $src(t_a) = s$ and $enabled(t_a, C'^p) \in \{\bot, true\}$.
   By definition of the abstraction, $act(t_a)$ either equals to $act(t)$, or equals to $act(t)$ with addition of manipulation of cg variables. Thus, by lemma 3, and since $\sigma^r \preceq \sigma'^p$, $act(t)(\sigma^r, C^r) \preceq act(t_a)(\sigma'^p, C'^p)$. We can then conclude that $C^{r+1} \preceq C'^{p+1}$.
3. For some $A^k \in ABS_l$, and for some $i \in \{1, ..., f+1\}$: $a_i^k \in \omega_l'^p$, $s \in \omega_l'^p$ and $a_i^k \lhd s$: This means that the current state of SM_l^A includes the abstraction, and the transition taken is from a state s which includes the abstraction. By definition of the abstraction, there exists a transition $t_a \in Trans_l^A$ s.t. $src(t_a) = a_i^k$, $trig(t_a) = \epsilon$, $grd(t_a) = true$, and $trgt(t_a) = a_{end}^k$. This means that $enabled(t_a, C'^p)$.
   We define $step'^p = TRANS(j, (t_a))$ and clearly $C^r \preceq C'^{p+1}$.
   Note that we match C'^{p+1} to C^r and not to C^{r+1}. We prove stuttering simulation, and this step of π' is part of the matching interval, continuing in the handling of a_{end}^k.
4. For some $A^k \in ABS_l$, $a_{end}^k \in \omega_l'^p$, $s \in \omega_l'^p$ and $a_{end}^k \lhd s$: This means that the current state of SM_l^A includes the abstraction, and the transition taken is from a state s which includes the abstraction. By definition of the abstraction, there exists a transition $t_a \in Trans_l^A$ s.t. $src(t_a) = a_{end}^k$, $trig(t_a) = \epsilon$, $grd(t_a) = true$, and $trgt(t_a) = a_{strt}^k$. This means that $enabled(t_a, C'^p)$.
   We define $step'^p = TRANS(j, (t_a))$ and clearly $C^r \preceq C'^{p+1}$.
   Note that we match C'^{p+1} to C^r and not to C^{r+1}. We prove stuttering simulation, and this step of π' is part of the matching interval, continuing in the handling of a_{strt}^k.
5. $s \in \omega_l'^p$, for every $A^k \in ABS_l$, if $\Delta(A^k) \cap \omega_l'^p \neq \phi$ and $a_{strt}^k \lhd s$ then $a_{strt}^k \in \omega_l'^p$ (there can possibly be more than one such k): This means that the current state of SM_l^A includes the abstraction, and the transition taken is from a state s which includes the abstraction. By definition of

the abstraction, there exists a transition $t_a \in Trans_l^A$ s.t. $src(t_a) = s$, $trig(t_a) = trig(t)$ and $grd(t_a) = grd(t)$.
For every transition $t'_a \in Trans_l^A$ s.t. $src(t'_a) = s' \in \omega_l'^p$ and $s' \lhd s$:
(a) If $s' = a^k_{strt}$ then by definition of the abstraction structure, $grd(t'_a) = \bot$ or $grd(t'_a) = grd\&\bot$. Therefore $enabled(t'_a, C'^p) = \bot$.
(b) If $s' \neq a^k_{strt}$ then $s' \in \omega_l^r$. By definition of the abstraction there exists $t' \in Trans_l$ s.t. $src(t') = s'$, $trig(t') = trig(t'_a)$ and $grd(t') = grd(t'_a)$. Since $enabled(t', C^r) = false$ (otherwise t is not enabled), and since $C^r \preceq C'^p$ then $enabled(t'_a, C^p) \in \{false, \bot\}$ as well.
From the above we conclude that $enabled(t_a, C'^p) \in \{\bot, true\}$. We define $step'^p = TRANS(j, (t_a))$ and $C^{r+1} \preceq C'^{p+1}$ (reasoning regarding the correctness w.r.t. the value of the variables is similar to the previous case).

6. $s \notin \omega_l'^p$, for some $A^k \in ABS_l$, $s \in S(A^k)$, for some $i \in \{1, ..., f+1\}$, $a^k_i \in \omega_l'^p$, and $trgt(t) \notin S(A^k)$: Notice that if $s \notin \omega_l'^p$, and since $c_l^r \preceq c_l'^t$, then $s \notin S_l^A$, and thus for some $A^k \in ABS_l$, $s \in S(A^k)$. This means that the current state of SM_l^A includes the abstraction, the transition taken is from a state s which is abstracted by $\Delta(A^k)$, and the target is not a state abstracted by $\Delta(A^k)$. By definition of the abstraction, there exists a transition $t_a \in Trans_l^A$ s.t. $src(t_a) = a^k_i$, $trig(t_a) = \epsilon$, $grd(t_a) = true$, and $trgt(t_a) = a^k_{end}$. This means that $enabled(t_a, C'^t)$.
We define $step'^t = TRANS(j, (t_a))$ and clearly $C^r \preceq C'^{p+1}$.
Note that we match C'^{p+1} to C^r and not to C^{r+1}. We prove stuttering simulation, and this step of π' is part of the matching interval, continuing in the handling of a^k_{end}.
7. $s \notin \omega_l'^p$, for some $A^k \in ABS_l$, $s \in S(A^k)$, $a^k_{end} \in \omega_l'^p$, and $trgt(t) \notin S(A^k)$: Notice that if $s \notin \omega_l'^p$, and since $c_l^r \preceq c_l'^p$, then $s \notin S_l^A$, and thus for some $A^k \in ABS_l$, $s \in S(A^k)$. This means that the current state of SM_l^A includes the abstraction, the transition taken is from a state s which is abstracted by $\Delta(A^k)$, and the target is not a state abstracted by $\Delta(A^k)$.
By definition of the abstraction, there exists a transition $t_a \in Trans_l^A$ s.t. $src(t_a) = a^k_{end}$, $trig(t_a) = \epsilon$, $grd(t_a) = true$, and $trgt(t_a) = a^k_{strt}$. This means that $enabled(t_a, C'^p)$.
We define $step'^p = TRANS(j, (t_a))$ and clearly $C^r \preceq C'^{p+1}$.
Note that we match C'^{p+1} to C^r and not to C^{r+1}. We prove stuttering simulation, and this step of π' is part of the matching interval, continuing in the handling of a^k_{strt}.
8. $s \notin \omega_l'^p$, for some $A^k \in ABS_l$, $s \in S(A^k)$, $a^k_{strt} \in \omega_l'^p$, and $trgt(t) \notin S(A^k)$: Notice that if $s \notin \omega_l'^p$, and since $c_l^r \preceq c_l'^p$, then $s \notin S_l^A$, and thus for some $A^k \in ABS_l$, $s \in S(A^k)$. This means that the current state of SM_l^A includes the abstraction, the transition taken is from a state s which is abstracted by $\Delta(A^k)$, and the target is not a state abstracted by $\Delta(A^k)$.
By definition of the abstraction there exists a matching transition $t_a \in Trans_l^A$ s.t. $src(t_a) = a^k_{strt}$, $trig(t_a) = trig(t)$, $grd(t_a) = grd(t)\&\bot$.

Since $C^r \preceq C'^p$, then if $enabled(t, C^r) = true$ then $enabled(t_a, C'^p) = \bot$. We define $step^p = TRANS(j, (t_a))$.
By definition of the abstraction, $act(t_a)$ equals to $act(t)$ with addition of manipulation of cg variables. Thus, based on lemma 3, and since $\sigma^r \preceq \sigma'^p$, $act(t)(\sigma^r, C^r) \preceq act(t_a)(\sigma'^p, C'^p)$.
By definition of the abstraction either $trgt(t_a) = trgt(t)$ (if $trgt(t) \in S_l^A$) or $trgt(t_a) = a_{strt}^{k'}$ for $A^{k'} \in ABS_l$ and $k' \neq k$ (if $trgt(t) \in S(A^{k'})$).
We conclude that $C^{r+1} \preceq C'^{p+1}$.

The following lemma is a result of the above items. The lemma states that if a transition t was taken in Γ during some computation π, and that transition is not abstracted by a single abstraction construct $\Delta(A^k)$, then the matching transition t_a is also taken in the matching interval in the matching computation π' of Γ^A.

Lemma 5. *Let $C^r \in \pi$ be a configuration s.t. $step^r = TRANS(j, (t_1, ..., t_q))$ (possibly $q = 1$) and $id_j^r = l$. For every transition $t \in \{t_1, ..., t_q\}$ s.t. there is no $A^k \in ABS_l$ for which $src(t), trgt(t) \in Sa(A^k)$:*
Assume $C^r \preceq C'^p$ and $C^{r+1} \preceq C'^{p+1}$. Then $step'^p = TRANS(j, (t_1^a, ..., t_{q'}^a))$ and there exists $t^a \in \{t_1^a, ..., t_{q'}^a\}$ s.t. t_a is the matching transition of t. Thus the following properties hold:

- *If $src(t) \in S_l^A$ then $src(t_a) = src(t)$. Otherwise ($src(t) \in S(A^k)$ for some k) then $src(t_a) = a_{strt}^k$.*
- *If $trgt(t) \in S_l^A$ then $trgt(t_a) = trgt(t)$. Otherwise ($trgt(t) \in S(A^k)$ for some k) then $trgt(t_a) = a_{strt}^k$.*
- *$trig(t_a) = trig(t)$*
- *either $grd(t_a) = grd(t)$ or $grd(t_a) = grd(t)\&\bot$*
- *$act(t_a)$ includes $act(t)$ with possible additional manipulation of cg variables.*

We will further need the following definition: Given some active state s in a state machine SM_l, we define the previous step in a computation which caused reaching to state s.

Definition 57 *Assume some computation $\pi = C^0, step^0, C^1, step^1, ...$ over a system Γ, a configuration $C^r \in \pi$, and a state $s \in \omega_l^r$. We define the* previous configuration *leading to s as $prev(\pi, r, l, s) = C^{r'}$ where $r' \lhd r$ if the following hold:*

1. *For every $r' \lhd r'' \lhd r$: $s \in \omega_l^{r''}$*
2. *$step^{r'} = TRANS(j, (t_1, ..., t_q))$ where $id_j^{r'} = l$, and there exists $i \in \{1, ..., q\}$ s.t. $s \leq trgt(t_i)$*
3. *For every $r' \lhd r'' \lhd r$: if $step^{r''} = TRANS(j, (t_1, ..., t_q))$ and $id_j^{r''} = l$, then for every $i \in \{1, ..., q\}$: $s \not\leq src(t_i)$*

If no such $C^{r'}$ exists then $prev(\pi, r, l, s) = \epsilon$

Notice that when $prev(\pi, r, l, s) = C^{r'}$ is not ϵ, then we assure that every transition of state machine SM_l that was executed between $step^{r'}$ and $step^r$

was either in an orthogonal region to s or in a state inside s. The execution of $step^{r'}$ caused the current state of SM_l to move to state s, since the target of one of the transitions executed in $step^{r'}$ is either s or a state containing s.

9. $s \notin \omega_l'^p$, for some $A^k \in ABS_l$, $s \in S(A^k)$, $a^k_{strt} \in \omega_l'^p$, and $trgt(t) \in S(A^k)$: Notice that if $s \notin \omega_l'^p$, and since $c_l^r \preceq c_l'^p$, then $s \notin S_l^A$, and thus for some $A^k \in ABS_l$, $s \in S(A^k)$. This means that the current state of SM_l^A includes the abstraction, the transition taken is from a state s which is abstracted by $\Delta(A^k)$, and the target is a state abstracted by $\Delta(A^k)$ as well. By definition of the current $TRANS$ step, $\rho_l^r = \epsilon$.
The definition of the matching interval for this step includes a small induction proof which states the following: For every configuration $C^r \in \pi$ s.t. $step^r = TRANS(j, (t_1, ..., t_q))$, and a matching abstract configuration C'^p s.t. $C^r \preceq C'^p$: If for some $t \in \{t_1, ..., t_q\}$ and for some $k \in \{1, ..., g\}$, $a^k_{strt} \in \omega_l'^p$, and $src(t), trgt(t) \in S_a(A^k)$, then $step'^p = TRANS(j, (t^a_1, ..., t^a_{q'}))$, and there exists $t^a \in \{t^a_1, ..., t^a_{q'}\}$ s.t. $C^{r+1} \preceq C'^{p+1}$, $src(t^a) = a^k_{strt}$, and $trgt(t^a) = a^k_1$.
For the case where $q > 1$ the property holds by the definition of matching interval for $TRANS$ step with multiple transitions. The case where $q = 1$ is handled in this item.
This property can be added to the entire induction proof, but it is relevant only in this item, where we define the interval matching such a step.
In the following we define the matching interval for $step^r$, and show that the matching interval terminates at abstracted state a^k_1.
Assume we can determine that $\sigma'^p(cg^k_l) \neq 0$. We can then define the matching interval as follows. It holds that $enabled(\tau^k_2, C'^p) = \perp$. We define $step'^p = TRANS(j, \tau^k_2)$, and differentiate according to $act(t)$:
(a) If $GEN(..) \notin act(t)$ then clearly $C^{r+1} \preceq C'^{p+1}$.
(b) If $GEN(ev') \in act(t)$ then $C^r \preceq C'^{p+1}$. We define stuttering inclusion, and this step is part of the matching interval, continuing in the handling of a^k_i.
The correctness of $C^{r+1} \preceq C'^{p+1}$ or $C^r \preceq C'^{p+1}$ w.r.t. the value of variables holds since by definition of the abstraction, all variables whose value might modify on $act(t)$ are set to $\perp$ on $act(\tau^k_2)$. For any other variable, since $C^r \preceq C'^p$ then the correct relation remains after $step^r$ and $step'^p$.
It now remains to be shown that indeed $\sigma'^p(cg^k_l) \neq 0$. We mark the the step representing the beginning of the current RTC step on thread j as $step^{r'} = DISP(j, ev)$. We denote the matching $DISP$ step on π' as $step'^{p'} = DISP(j, ev)$. Consider $prev(\pi, r, l, s)$.
If $prev(\pi, r, l, s) = \epsilon$ then for every $\alpha \in \{0, ..., r\}$, $s \in \omega_l^\alpha$. Thus $s \in init_l$. By our requirement from abstraction sets, and since $trig(t) = \epsilon$, we know that $grd(t) \neq true$. Since $enabled(t, C^r)$, we know that $grd(t)(\sigma^r) = true$. We denote by r'' s.t. $r'' < r$ the maximal index for which $grd(t)(\sigma^{r''}) = false$ and $grd(t)(\sigma^{r''+1}) = true$.

- If $r'' = 0$: this means that $grd(t)(\sigma^{\alpha}) = true$ for every $\alpha \in \{0, ..., r\}$. Assume there exists $t' \in \Delta(A^k)$ that was executed in some step r''' prior to the current step. By the property resented at the beginning of this item, this step did not occur in the current RTC step (otherwise, the interval matching that execution should traverse from a^k_{strt} to a^k_1). Consider the $EndRTC$ step on thread j that followed $step^{r'''}$, denoted $step^{r^*} = EndRTC(j, \epsilon)$. Since a transition executed on that RTC step, then $\rho^{r^*}_l = \epsilon$. Also, $grd(t)(\sigma^{r^*}) = true$. Since $s \in \omega^{r^*}_l$, then clearly $enabled(t, C^{r^*})$. This is in contradiction to the fact that an $EndRTC$ step occurred. We can then conclude that no transition $t' \in \Delta(A^k)$ has been executed prior to the current step.
  The above means that for every $\alpha' \in \{0, ..., p\}$, $a^k_{strt} \in \omega'^{\alpha'}_l$. Since $grd(t)(\sigma^0) = true$, $trig(t) = \epsilon$ and $s \in c^0_l$, then $\sigma'^0(cg^k_l) = 1$ (by the base definition). The only transition that can change the value of cg^k_l to 0 is a transition from a^k_{strt} to a^k_1, and we know such transition was not taken. Thus we can conclude that $\sigma'^p(cg^k_l) \neq 0$.
- If $r'' > 0$. Assume $step^{r''}$ is executed on state machine $SM_{l'}$ (possibly $l' = l$). We first notice if $grd(t)(\sigma^{r''}) = false$ and $grd(t)(\sigma^{r''+1}) = true$, then $step^{r''} = TRANS(j', (t'_1, ..., t'_y))$ and for some $t' \in \{t'_1, ..., t'_h\}$, $v \in modif(t')$. Assume $id^{r''}_{j'} = l'$ ($SM_{l'}$ executes $step^{r''}$). By the definition of abstraction of systems, this means that $thread(l') = thread(l)$, thus $j' = j$. We separate between two cases, based on the relation between r'' and r':

  (a) If $r'' < r'$: We show that for every $\alpha \in \{r'', ..., r'\}$, SM_l did not execute any transition in $step^{\alpha}$. Assume, by contradiction, that such a transition did execute. Consider the $EndRTC$ step on thread j that followed $step^{\alpha}$ (denoted $step^{r'''}$). Since a $TRANS$ step occurred in this RTC step, then $\rho^{r'''}_l = \epsilon$. We also know that $grd(t)(\sigma^{r'''}) = true$. Since $s \in \omega^{r'''}_l$, then clearly $enabled(t, C^{r^*})$. This is in contradiction to the fact that an $EndRTC$ step occurred in $step^{r'''}$. We can then conclude that $l' \neq l$. Since SM_l and $SM_{l'}$ are on the same thread, then they cannot both execute a RTC step simultaneously.
  Denote the beginning of the RTC step that ended on $step^{r'''}$ with $step^{r^*} = DISP(j, e)$. Denote the step on π' that matches $step^{r^*}$ as $step'^{p^*} = DISP(j, e)$. From the above, we can conclude that for every $\alpha' \in \{p^*, ..., p'\}$, $a^k_{strt} \in \omega'^{\alpha'}_l$. Moreover, from the property presented in the beginning of the item we can conclude that for every $\alpha' \in \{p^*, ..., p\}$, $a^k_{strt} \in \omega'^{\alpha'}_l$.
  Since for some $v \in GRDV(A^k)$, $v \in modif(t')$, then we know that for some $\beta \in \{p^*, ..., p'\}$, cg^k_l was set to 1 on $step'^{\beta}$ (If t' has a matching transition t'_a, then during the execution of t'_a (by lemma 5). If t' is abstracted by $\Delta(A^{k'})$, then during execution of transition from $a^{k'}_{strt}$ to $a^{k'}_1$ that occurred due to the property in the beginning of this item).

The only transition that can set cg_l^k to 0 is a transition from a_{strt}^k to a_1^k. Since we know that for every $\alpha' \in \{p^*, ..., p\}$, $a_{strt}^k \in \omega_l'^{\alpha'}$, then no such transition executed. We can then conclude that $\sigma^p(cg_l^k) \neq 0$.

(b) If $r'' > r'$: This means that $l' = l$ (no other SM on thread j can execute between $step^{r'}$ and $step^r$). We know that $a_{strt}^k \in \omega_l'^p$. By lemma 7 and by the property from the beginning of this item we can conclude that for every $\alpha' \in \{p', ..., p\}$, $a_{strt}^k \in \omega_l'^{\alpha'}$.
Since for some $v \in GRDV(A^k)$, $v \in modif(t')$, then we know that for some $\beta \in \{p', ..., p\}$, cg_l^k was set to 1 on $step'^{\beta}$ (If t' has a matching transition t'_a, then during the execution of t'_a (by lemma 5). If t' is abstracted by $\Delta(A^{k'})$, then during execution of transition from $a_{strt}^{k'}$ to $a_1^{k'}$ that occurred due to the property in the beginning of this item).
The only transition that can set cg_l^k to 0 is a transition from a_{strt}^k to a_1^k. Since we know that for every $\alpha' \in \{p^*, ..., p\}$, $a_{strt}^k \in \omega_l'^{\alpha'}$, then no such transition executed. We can then conclude that $\sigma^p(cg_l^k) \neq 0$.

If $prev(\pi, r, l, s) = C^{r''} \neq \epsilon$. By definition of $prev$, $step^{r''} = TRANS(j, (t_1, ..., t_q))$ s.t. for some $t' \in \{t_1, ..., t_q\}$, $s \lhd trgt(t')$.
We separate between two cases, and show that under both cases $\sigma'^p(cg^k) \neq 0$:

(a) $r'' > r'$: This means that the execution of the transition leading to s occurred after the beginning of the RTC step. We first show that it is not possible that $t' \in \Delta(A^k)$. Assume it is (i.e., both $src(t')$ and $trgt(t')$ are in $S(A^k)$). Consider the first transition in $\Delta(A^k)$ that was executed in the current RTC step. By the property presented in the beginning of this item, that transition caused a traversal from a_{strt}^k to a_1^k. By lemma 7 and since $a_{strt}^k \in \omega_l'^p$ we can conclude that no such abstracted transition was executed.
If $trgt(t') \notin S(A^k)$ (i.e., $trgt(t')$ contains s and is not abstracted): By definition of the abstraction, t' has a matching transition $t'_a \in TR_l^A$, and by lemma 5, t'_a was executed at $step'^{p''}$ that matches $step^{r''}$. Since we have no history and $s \in \omega_l^{r''+1}$, then $a_{strt}^k \in \omega_l'^{p''+1}$. By lemma 7 we can conclude that for every $\alpha' \in \{p'', ..., p\}$, $a_{strt}^k \in \omega_l'^{\alpha'}$. Since only transitions exiting a_{strt}^k can set cg_l^k to 0, then we can conclude that $\sigma^p(cg_l^k) \neq 0$.
If $trgt(t') \in S(A^k)$ then $src(t') \notin S(A^k)$. Since t' is not an abstracted transition, then there exists a matching transition $t'_a \in TR_l^A$. By lemma 5, there exists $step'^{p''}$ that matches $step^{r''}$ and t'_a is executed on $step'^{p''}$. By definition of the abstraction, $cg_l^k = 1 \in act(t'_a)$. Since $src(t') \notin S(A^k)$ then $src(t'_a) \notin \Delta(A^k)$. Since there are no cross hierarchy transitions, then $a_{strt}^k \notin \omega_l'^{p''}$. The only transitions that can set cg_l^k to 0 are transitions exiting a_{strt}^k. We can then conclude that $\sigma'^{p''+1}(cg_l^k) \neq 0$.

We now show that if $\sigma'^{p''+1}(cg_l^k) \neq 0$ then $\sigma'^{p}(cg_l^k) \neq 0$ as well. The only transitions that can set cg_l^k to 0 are transitions from a_{strt}^k to a_1^k. We know that $a_{strt}^k \in \omega_l'^{p''+1}$ and $a_{strt}^k \in \omega_l'^{p}$. If we show that for every $\alpha' \in \{p''+1, ..., p\}$, $a_{strt}^k \in \omega_l'^{\alpha'}$ then clearly no such transition is taken and $\sigma'^{p}(cg_l^k) \neq 0$. Assume this is not the case, and that for some $\alpha' \in \{p''+1, ..., p-1\}$, $a_{strt}^k \notin \omega_l'^{\alpha'}$. By the definition of the abstraction, in order to return to a_{strt}^k the transition from a_{end}^k to a_{strt}^k had to be executed at some $step^{\beta}$ s.t. $\alpha' < \beta < p$. However, by lemma 7, the interval that includes $step'^{\beta}$ either leaves the abstraction or is part of an $EndRTC$ step. It is not possible that $step'^{\beta}$ is part of an $EndRTC$ step, since by lemma 4 there had to be a matching $EndRTC$ step in π, and we know that the last $EndRTC$ step on thread j occurred at $step^{r'}$, which is prior to $step^{r''}$ that matches $step^{p''}$. Also, we know that $prev(\pi, r, l, s) = C^{r''}$, thus for every $r'' < r^* \leq r$, $s \in \omega_l^{r^*}$. Therefore, since we assume stuttering simulation on the prefix of the computations, then abstraction $\Delta(A^k)$ must be part of the configuration for every $\omega_l'^{p^*}$, for $p''+1 < p^* \leq p$ and it is not possible that $step'^{\beta}$ leaves the abstraction. Thus, for every $p''+1 \leq \alpha' \leq p$, $a_{strt}^k \in \omega_l'^{\alpha'}$.
We conclude that $\sigma'^{p}(cg^k) \neq 0$.

(b) $r' > r''$: This means that the execution of the transition leading to s occurred before the beginning of the RTC step. By definition of $prev$, this means that for every $r'' < r^* \leq r$, $s \in \omega_l^{r^*}$. Since the transition execution ($step^{r''}$) occurred before the $DISP$ step ($step^{r'}$), then by the semantics there exists $r'' < r''' < r'$ s.t. $step^{r'''} = EndRTC(j, \epsilon)$ where $id_j^{r'''} = l$. Consider the last such $EndRTC$ step on state machine SM_l (meaning: for every $r''' < r^* < r'$, if $step^{r^*} = EndRTC(j, \epsilon)$ then $id_j^{r^*} \neq l$). By the semantics of $EndRTC$ step and since $s \in \omega_l^{r'''}$, we know that $enabled(t, C^{r'''}) = false$. We also know that $enabled(t, C^{r}) = true$. This means that there exists some variable $v \in GRDV(A^k)$ s.t. the value of v was modified between steps r''' and r. Formally, there exists $r''' < \tilde{r} < r$ s.t. $step^{\tilde{r}} = TRANS(j, (t_1, ..., t_q))$, and for some $\tilde{t} \in \{t_1, ..., t_q\}$, $v \in modif(\tilde{t})$ and $\sigma^{\tilde{r}}(v) \neq \sigma^{\tilde{r}+1}(v)$.
Consider the configuration that matches $C^{r''+1}$ on π', denoted $C'^{p''+1}$. By definition of $prev$, for every $r^* \in \{r''+1, ..., r\}$, $s \in \omega_l^{r^*}$. Thus $\Delta(A^k) \cap \omega'^{p^*} \neq \phi$ for every $p^* \in \{p''+1, ..., p\}$. Assume $step'^{p'''} = EndRTC(j, \epsilon)$ matches $step^{r'''}$. Thus, $a_{strt}^k \in \omega_l'^{p'''}$ (by Lemma 2).
We want to show that for every $p^* \in \{p''', ..., p\}$: $a_{strt}^k \in \omega_l'^{p^*}$. Assume this is not the case. Since for every $p^* \in \{p''', ..., p\}$, $\Delta(A^k) \cap \omega'^{p^*} \neq \phi$, and $a_{strt}^k \in \omega_l'^{p'''}$, then this means that a transition from a_{strt}^k to a_1^k was executed at some $step'^{p^*}$, for $p''' < p^* \leq p$. Since $a_{strt}^k \in \omega_l'^{p}$ then the transition from a_{end}^k to a_{strt}^k had to be executed on some $step'^{p^{**}}$ for $p^* < p^{**} \leq p$. By lemma 7, this means that $step'^{p^{**}}$ is

either part of an $EndRTC$ step or a step leaving the abstraction. It is not part of an $EndRTC$ step, since the last $EndRTC$ step on the current state machine is in $step'^{p'''}$ (and $p''' < p^{**}$). It is not possible that $step'^{p^{**}}$ leaves the abstraction, since we know $\Delta(A^k)$ is part of the configuration for all the steps until $step'^{p}$. We can then conclude that for every $p^* \in \{p''', ..., p\}$: $a^k_{strt} \in \omega'^{p^*}_l$.
Recall that $r''' < \tilde{r} < r$, $step^{\tilde{r}} = TRANS(j, (t_1, ..., t_q))$ and for some $\tilde{t} \in \{t_1, ..., t_q\}$, $v \in modif(\tilde{t})$ and $\sigma^{\tilde{r}}(v) \neq \sigma^{\tilde{r}+1}(v)$. Consider the last such $\tilde{r}$ (meaning, for every $r^* \in \{\tilde{r}+1, ..., r\}$: $\sigma^{r^*}(v) = \sigma^{\tilde{r}+1}(v)$
We separate between two cases, whether or not $\tilde{t}$ is an abstracted transition.

i. If $\tilde{(t)}$ is not an abstracted transition. Assume $\tilde{(t)} \in TR_{l'}$ (possibly $l' = l$). Then by definition of the abstraction, there exists a matching transition $\tilde{(t)}_a \in TR^A_{l'}$. By lemma 5 there exists a $step'^{\tilde{p}}$ on π' that matches $step^{\tilde{r}}$ and $\tilde{(t)}_a$ is executed on $step'^{\tilde{p}}$. By the definition of the abstraction, cg^k_l is set to 1 on $act(\tilde{(t)}_a)$. Thus, $\sigma'^{\tilde{p}+1}(cg^k_l) \neq \epsilon$. The only transitions that can set cg^k_l to 0 are transitions exiting a^k_{strt}. However, we know that such transition was not taken, since for every $p^* \in \{p''', ..., p\}$: $a^k_{strt} \in \omega'^{p^*}_l$, and $p''' < \tilde{p} < p$. We can therefore conclude that $\sigma'^{p}(cg^k_l) \neq 0$.
ii. If $\tilde{t}$ is an abstracted transition. First of all, note that $\tilde{t'} \notin \Delta(A^k)$. This holds as a result of the property presented at the beginning of this item. Otherwise, by the property, the matching interval should traverse from a^k_{strt} to a^k_1, which did not occur, since for every $p^* \in \{p''', ..., p\}$: $a^k_{strt} \in \omega'^{p^*}_l$
Since we assume variables can be modified only by state machines under the same thread, then we know that $\tilde{t} \in TR_{l'}$ s.t. $thread(l') = j$. Under a single thread j RTC steps are executed one after the other. Assume that the $DISP$ step initiating the RTC step that includes $step^{\tilde{r}}$ is $step^{\tilde{r}'''} = DISP(j, \epsilon)$ and $id^{\tilde{r}'''}_j = l'$. Since $r''' < \tilde{r} < r$, then $r''' < \tilde{r}''' < \tilde{r}$.
Assume $\tilde{t} \in \Delta(A^{k'})$ ($A^{k'} \in ABS_{l'}$), and consider the first transition $\tilde{t'}$ executed in the RTC step initiated by $step^{\tilde{r}'''}$ that is abstracted by $\Delta(A^{k'})$. This means that $\tilde{t'} \in TR(A^{k'})$, and for some $\tilde{r'} \in \{\tilde{r}''', ..., \tilde{r}\}$, $step^{\tilde{r'}} = TRANS(j, (t_1, ..., t_q))$, and $\tilde{t'} \in \{t_1, ..., t_q\}$. Also, for every $r^* \in \{\tilde{r}''', ..., \tilde{r'} - 1\}$, if $step^{r^*} = TRANS(j, (t'_1, ..., t'_q))$ then for every $t \in \{t'_1, ..., t'_q\}, t \notin TR(A^{k'})$.
Consider the step matching $step^{\tilde{r'}}$ on π', $step'^{\tilde{p'}}$. Since $\tilde{t'}$ is the first executed transition in the RTC step that is abstracted by $ABS^{k'}$ then $a^{k'}_{strt} \in \omega'^{\tilde{p'}}_l$. Therefore $a^{k'}_1 \in \omega'^{\tilde{p'}+1}_l$ (this is a result of the property presented at the beginning of this item, stating that if we execute an abstracted transition and the matching abstract configuration is at a^k_{strt}, then the matching step executes a transition from a^k_{strt} to a^k_1).

Since $v \in modif(\tilde{t})$ and $v \in GRDV(A^k)$, then by the definition of the abstraction, $cg_l^k = 1 \in act(t^a)$ for every transition t^a where $src(t^a) = a_{strt}^{k'}$ and $trgt(t^a) = a_1^{k'}$. The only transitions that can set cg_l^k to 0 are transitions from a_{strt}^k. We know that such transitions were not taken, since for every $p^* \in \{p', ..., p\}$: $a_{strt}^k \in \omega_l'^{p^*}$, and $p' < \tilde{p} < p$. We can therefore conclude that $\sigma'^p(cg_l^k) \neq 0$.

From the above we conclude that indeed $\sigma'^p(cg_l^k) \neq 0$.

We formalize the property presented in the previous item in the following lemma:

Lemma 6. *For every configuration $C^r \in \pi$ s.t. $step^r = TRANS(j, (t_1, ..., t_q))$, and a matching abstract configuration C'^p s.t. $C^r \preceq C'^p$: If for some $t \in \{t_1, ..., t_q\}$ and for some $k \in \{1, ..., g\}$, $a_{strt}^k \in \omega_l'^p$, and $src(t), trgt(t) \in S(A^k)$, then $step'^p = TRANS(j, (t_1^a, ..., t_{q'}^a))$, and there exists $t^a \in \{t_1^a, ..., t_{q'}^a\}$ s.t. $C^{r+1} \preceq C'^{p+1}$, $src(t^a) = a_{strt}^k$, and $trgt(t^a) = a_1^k$.*

10. $s \notin \omega_l'^p$, for some $A^k \in ABS_l$, $s \in S(A^k)$, for some $i \in \{1, ..., f+1\}$: $a_i^k \in \omega_l'^p$, and $trgt(t) \in S(A^k)$: Notice that if $s \notin \omega_l'^p$, and since $c_l^r \preceq c_l'^p$, then $s \notin S_l^A$, and thus for some $A^k \in ABS_l$, $s \in S(A^k)$. This means that the current state of SM_l^A includes the abstraction, the transition taken is from a state s which is abstracted by $\Delta(A^k)$, and the target is a state abstracted by $\Delta(A^k)$ as well. By definition of the current $TRANS$ step, $\rho_l^r = \epsilon$.
The difference between C^r and C^{r+1} is in the value of variables and/or in the value of some event queue (if $GEN(e) \in act(t)$). We first consider the variables and show that $\sigma^{r+1} \preceq \sigma'^p$:
For every variable $v \notin modif(t)$, $\sigma^r(v) = \sigma^{r+1}(v)$, and thus clearly either $\sigma'^p(v) = \perp$ or $\sigma'^p(v) = \sigma^{r+1}(v)$.
For every variable $v \in modif(t)$, we show that $\sigma'^p(v) = \perp$: Since $v \in modif(t)$ then by definition $v \in V(A^k)$. By the definition of the abstract model, if $a_i^k \in \omega_l'^p$, then there exists $C'^{p'} \in \pi'$ s.t. $a_{strt}^k \in \omega_l'^{p'}$ and $step'^{p'} = TRANS(j, (t_1', ..., t_q'))$ s.t. there exists $t_a' \in \{t_1', ..., t_q'\}$ where $src(t_a') = a_{strt}^k$ and $trgt(t_a') = a_1^k$. This means that for every $v \in V(A^k)$, $\sigma'^{p'+1}(v) = \perp$. Since $\Delta(A^k)$ can be entered only through a_{strt}^k, we can conclude that $step'^{p'}$ occurred in the current RTC step (meaning, there is no $p'' \in \{p', ..., p\}$ s.t. $step'^{p''} = EndRTC(j, \epsilon)$). We also know that for every $p'' \in \{p', ..., p\}$, $\omega_l'^{p''} \cap \Delta(A^k) \neq \phi$.
Since variables can be modified only by state machines on the same thread, then if v is modified in some step p'' where $p'' \in \{p', ..., p\}$, then $step'^{p''} = TRANS(j, (t_1, ..., t_q))$ and $id_j'^{p''} = l$. By the definition of the abstraction, if for some $t_a' \in \{t_1, ..., t_q\}$, t_a' modifies $v \in V^A$ then one of the following holds:

- t_a' is an abstracted transition, in this case $v = \perp \in act(t_a')$, or

- t'_a is not an abstracted transition, in this case consider t' the matching transition of t'_a. By definition of the abstraction, $v = e \in act(t')$ and "if $(isIn(A^k))$ $v = \perp$; else $v = e$;"$\in act(t'_a)$. Since $isIn(A^k) = true$, then $act(t'_a)(\sigma'^{p''}, C'^{p''}) = \perp$.

We can then conclude that $\sigma'^{p}(v) = \perp$.

We define the matching step based on $act(t)$:

(a) If $GEN(e) \notin act(t)$ (for some event e) - this means that the difference between C^r and C^{r+1} is only in the value of variables. We do not define a matching step in π', and clearly $C^{r+1} \preceq C'^{p}$.

(b) If $GEN(e) \in act(t)$ (for some event e) then by definition of the abstraction $e \in EV(A^k)$. By definition of the abstraction there exists a transition $t_a \in Trans'_l$ s.t. $src(t_a) = a^k_i$, $trgt(t_a) = a^k_{i+1}$, $trig(t_a) = \epsilon$ and $grd(t_a) = true$. Since $\rho'^{p}_l = \epsilon$, then $enabled(t_a, C'^{p}) = true$. We define $step'^{p} = TRANS(j, t_a)$. By definition of the abstraction it is possible that the action on t_a is $GEN(e)$, therefore $C^{r+1} \preceq C'^{p+1}$.

11. $s \notin \omega'^{p}_l$, for some $A^k \in ABS_l$, $s \in S(A^k)$, $a^k_{f+1} \in \omega'^{p}_l$, and $trgt(t) \notin S(A^k)$: Notice that if $s \notin \omega'^{p}_l$, and since $c^r_l \preceq c'^{p}_l$, then $s \notin S^A_l$, and thus for some $A^k \in ABS_l$, $s \in S(A^k)$. This means that the current state of SM^A_l includes the abstraction, the transition taken is from a state s which is abstracted by $\Delta(A^k)$, and the target is a state abstracted by $\Delta(A^k)$ as well.
    Every computation on Γ^A can enter one of $\Delta(A^k)$ states only by entering a^k_{strt}. Recall that there can be at most f events generated in a single RTC step within the states abstracted by $\Delta(A^k)$ in Γ. If we reach a_{f+1} then by the definition of the abstraction, f events were generated in the current RTC step within the states abstracted by $\Delta(A^k)$ in Γ. Therefore, $GEN(e) \notin act(t)$. For the same reasoning as in the previous item, we do not define a matching step in π', and $C^{r+1} \preceq C'^{p}$.
12. $s \notin \omega'^{p}_l$, for some $A^k \in ABS_l$, $s \in S(A^k)$, $a^k_{end} \in \omega'^{p}_l$, and $trgt(t) \in S(A^k)$: Notice that if $s \notin \omega'^{p}_l$, and since $c^r_l \preceq c'^{p}_l$, then $s \notin S^A_l$, and thus for some $A^k \in ABS_l$, $s \in S(A^k)$. This means that the current state of SM^A_l includes the abstraction, the transition taken is from a state s which is abstracted by $\Delta(A^k)$, and the target is a state abstracted by $\Delta(A^k)$ as well.
    This situation is not possible. Every computation on Γ^A can enter the one of $\Delta(A^k)$ states only by entering a^k_{strt}. Recall that there can be at most f events generated in a single RTC step within the abstracted states in Γ. The simulation is defined s.t. an execution of an abstracted transition t in Γ matches an execution of a transition in Γ^A only if $GEN(e) \in act(t)$. If $GEN(e) \notin act(t)$ then there is no execution of a matching transition in Γ^A. Therefore, it is not possible that t is an abstracted transition and $a^k_{end} \in \omega'^{p}_l$.

The following lemma is based on the definition of the simulation relation:

Lemma 7. *The interval that includes a move from state a^k_i for $1 \leq i \leq f+1$ to a^k_{end} matches a concrete transition that either leaves the abstraction or an EndRTC step.*

□

6 Using Abstraction

We now present the applicability of our abstraction framework through an example. We consider a system Γ describing a travel agent (of class *Agent*) that books flights and communicates with both airline databases (of class DB) and clients. We assume Γ includes n different DB objects, where the behavior of each DB is defined in Fig. 1. The single *Agent* object in Γ communicates with clients (modeled as the environment) and with all of the DBs. The *Agent* behavior is as follows: upon receiving a flight request from a client, it requests a price offer from all DBs by sending event *evGetPrc* to them. After getting an answer from the DBs (via *evRetPrc*), it chooses an offer, reserves the flight from the relevant DB (via *evAprvFlt*) and rejects the offers from the rest of the DB (via *evDenyFlt*).

Assume now we create an abstract system Γ^A, where the DBs are abstracted as in Fig. 3 (the *Agent* remains concrete). If *Agent* state machine includes x states, then Γ has $(12*n+x)$ states, whereas Γ^A has $(4*n+x)$ states. Moreover, Γ^A does not include the pieces of code in the actions of the transitions of DBs, which may be complicated. E.g., the method *calcPrc*() is not part of the abstract state machine of DB, and this method might include complex computations.

Assume we want to verify the property describing that on all computations of Γ, if *Agent* orders a flight from some DB, then all the DBs returned an answer to the *Agent* before the *Agent* chooses an offer. For this property it is enough to consider only the *interface* of the DBs. The property is not affected, for example, by the calculation of a price by the DBs. It is an outcome only of the information that every DB can consume an event *evGetPrc*, and can send an event *evRetPrc*. We can therefore verify the property on Γ^A. If the property holds, then we can conclude that Γ also satisfies the property.

Consider another property: we want to verify that due to a single request from the client, *space* decreases by at most 1. Clearly, when verifying the property on Γ^A, the result is $\perp$, since Γ^A abstracts the variable *space*. This means that we cannot conclude whether or not the property holds on Γ by model checking Γ^A. However, it might be possible to *refine* Γ^A, and create a different abstraction Γ'^A for which this property can be verified. Following, in section 7 we present how to refine an abstract system when the verification does not succeed.

7 Refinement

Once we have an abstract system Γ^A, we model check our LTL_x property $A\psi$ over the abstract system. Since variables in Γ^A can have the value $\perp$, then $(\Gamma^A \models A\psi) \in \{true, false, \perp\}$. If $(\Gamma^A \models A\psi) = true$, then from Theorem 56 the property holds on Γ as well. If $(\Gamma^A \models A\psi) \in \{false, \perp\}$ then due to Γ^A being an over-approximation we cannot determine whether or not the property holds on Γ. Typical model checkers provide the user with a CEX in case verification

does not succeed. A CEX π^A on Γ^A is either a finite computation or a lasso computation s.t. either $(\pi^A \models \psi) = false$ or $(\pi^A \models \psi) = \bot$.

Next we present a CEGAR-like algorithm for refining Γ^A based on π^A. The refinement step suggests how to create a new abstract system Γ'^A, where one or more of the abstracted states of Γ are removed from the abstraction sets. Since the concrete system Γ is finite, the $CEGAR$ algorithm ultimately terminates and returns a correct result.

If $(\pi^A \models \psi) = \bot$ then we cannot determine the value of the property. If $(\pi^A \models \psi) = false$, then this CEX might be spurious. In both cases we search for a computation π on Γ s.t. $\pi \preceq_s \pi^A$. Given π^A, we inductively construct π w.r.t. π^A. Note that if the concrete model enables non-determinism, then there might be more than one matching concrete CEXs. In this case, all the matching concrete CEXs are simultaneously constructed. Intuitively, the construction of π follows the steps of π^A, maintaining the stuttering inclusion. During the construction, if for some prefix of π^A: $C'^0, step'^0, ..., step'^{p-1}, C'^p$ it is not possible to extend any of the matching concrete computations based on $step'^p$, then π^A is a spurious CEX and we should refine the system.

Base: Given $C'^0 = (c_1'^0, ..., c_n'^0, q_1'^0, ..., q_m'^0, id_1'^0, ..., id_m'^0, \sigma'^0)$, the initial configuration of π^A. We define the following initial configuration for π: $C^0 = (c_1^0, ..., c_n^0, q_1^0, ..., q_m^0, id_1^0, ..., id_m^0, \sigma^0)$.

- For every $1 \leq i \leq n$, $\omega_i^0 = \{s | s \in init_i \wedge \forall s' : s \lhd s' \rightarrow s \in init_i\}$
- For every $1 \leq i \leq n$, $\rho_i^0 = \epsilon$.
- For every $1 \leq i \leq n$ and for every $r \in R_i$, $H_i^0(r) = s$ s.t. $s \in init_i$ and $parent(s) = r$.
- For every $1 \leq j \leq m$, $q_j^0 = q_j'^0 = \phi$
- For every $1 \leq j \leq m$, $id_j^0 = id_j'^0 = 0$
- For every $v \in V$, $\sigma^0(v) = \sigma'^0(v)$

It is immediate to see that C'^0 is an initial configuration and also that $C^0 \preceq C'^0$

Step: Assume that for the first p steps of π^A: $C'^0, step'^0, C'^1, step'^1, ..., C'^{p-1}, step'^{p-1}, C'^p$ there exists a partial computation $\pi = C^0, step^0, C^1, step^1, ..., C^r$ over Γ s.t. there are two sequences of integers $0 = i_0 < i_1 < i_2 < ... < i_l = r + 1$ and $0 = i'_0 < i'_1 < i'_2 < ... < i'_l = p + 1$ and for every $0 \leq k < l$:
For every $j \in \{i_k, ..., (i_{k+1} - 1)\}$ and for every $j' \in \{i'_k, ..., (i'_{k+1}) - 1\}$: $C^j \preceq C'^{j'}$

Note that the requirement on interval $(l - 1)$ induces $C^r \preceq C'^p$.

The matching extension of π is defined based on $step'^p$:

- $step'^p = DISP(j, ev)$
  By definition, $id_j'^p = 0$, $q_j'^p \neq \phi$ and $top(q_j'^p) = ev$. Since $C^r \preceq C'^p$, then $id_i^r = id_i'^p$ and $q_i^r = q_i'^p$ for every i. Therefore, $id_j^r = 0$, $q_j^r \neq \phi$ and $top(q_j^r) = ev$ as well, and $step^r = DISP(j, ev)$ is a possible step from C^r.
  By definition of $DISP$ step, C'^{p+1} differs from C'^p in the following: $q_j'^{p+1} = pop(q_j'^p)$, $id_j'^{p+1} = trgt(ev)$ and $\rho'^{p+1}_{trgt(ev)} = type(ev)$.
  By definition of $DISP$ step, C^{r+1} differs from C^r in the following: $q_j^{r+1} = pop(q_j^r)$, $id_j^{r+1} = trgt(ev)$ and $\rho^{r+1}_{trgt(ev)} = type(ev)$.
  Since $C^r \preceq C'^p$, it is clear that $C^{r+1} \preceq C'^{p+1}$ as well.

– $step'^p = ENV(j, ev)$.
Since the environment is always enabled, then an ENV step s.t. $step^r = ENV(j, ev)$ is possible from C^r, and clearly $C^{r+1} \preceq C'^{p+1}$.
– $step'^p = EndRTC(j, \epsilon)$
Assume $id_j^r = id_j'^p = l > 0$. If $stable(c_l^r, C^r)$ then we define $step^r = EndRTC(j, \epsilon)$ and clearly $C^{r+1} \preceq C'^{p+1}$. Otherwise, $stable(c_l^r, C^r) = false$, in which case we might need to refine.
If $stable(c_l^r, C^r) = false$ then there exists $s \in \omega_l^r$ and $t_c \in Trans_l$ where $src(t_c) = s$ s.t. $trig(t_c) = \rho_l^r$ and $grd(t_c)(\sigma^r) = true$. Assume SM_l^A is an abstraction of SM_l w.r.t. $ABS_l = \{A^1, ..., A^g\}$.
1. If for every $k \in \{1, ..., g\}$, $s \notin S(A^k)$ then by definition of SM_l^A, $s \in S_l^A$ and there exists a matching transition $t_a \in Trans_l^A$ s.t. $src(t_a) = s$, $trig(t_c) = trig(t_a)$ and $grd(t_c) = grd(t_a)$. By definition of the simulation, $s \in \omega_l^r$. Since $trig(t_c) = trig(t_a)$ and $stable(c_l'^p, C'^p) \in \{\bot, true\}$ then this means that $grd(t_a)(\sigma'^p) \in \{false, \bot\}$. However, we know that for every $v \in V$, either $\sigma'^p(v) = \bot$ or $\sigma'^p(v) = \sigma^r(v)$. Since $grd(t_c) = grd(t_a)$,then it is not possible that $grd(t_c)(\sigma^r) = true$ and $grd(t_a)(\sigma'^p) = false$. We conclude that $grd(t_a)(\sigma'^p) = \bot$. We choose some variable $v \in V$ s.t. $\sigma'^p(v) = \bot$ and v effects the evaluation of $grd(t_a)$ and refine Γ^A w.r.t. v. We present how to refine Γ^A w.r.t. some variable v in section 7.1.
2. If for some $k \in \{1, ..., g\}$, $s \in S(A^k)$ (s is an abstracted state). Since $c^r \preceq c'^p$ then $\Delta(A^k) \cap \omega_l'^p \neq \phi$. From lemma 2 we conclude that $a_{strt}^k \in \omega_l'^p$. We separate between different cases.
   - If $\rho_l^r = \epsilon$ and for every abstracted variable $v \in V(A^k)$, $\sigma'^p(v) = \bot$: this means that the abstraction has been traversed. It is possible that the concrete model might reach a stable state after traversing abstracted transitions without GEN. Recall that these transitions do not have a matching transition in the abstraction. Continue from C^r the RTC step on abstracted transitions (transitions $t' \in Trans(A^k)$), as long as it is on transitions without GEN. Since the by definition of the abstraction, these transitions can only modify variables from $V(A^k)$, which have the value $\bot$ in σ'^p, then on all such reachable configurations $C^{r'} \preceq C'^p$. When cannot progress anymore, check $stable(c_l^{r'}, C^{r'})$. If for *one of the reachable configurations* $stable(c_l^{r'}, C^{r'}) = true$ then we define $step^{r'} = EndRTC(j, \epsilon)$ and clearly $C^{r'+1} \preceq C'^{p+1}$. Otherwise, there exists $s' \in \omega_l^{r'}$ and $t_c' \in Trans_l$ where $src(t_c') = s'$ s.t. $trig(t_c') = \epsilon$ and $grd(t_c')(\sigma^{r'}) = true$.
     * If $s' \in S_l^A$: then $s' \in \omega_l'^p$, and there exists a transition $t_a' \in Trans_l^A$ s.t. $src(t_a') = s'$, $trig(t_a') = trig(t_c') = \epsilon$ and $grd(t_c') = grd(t_a')$. Since $stable(c_l'^p, C'^p) \in \{\bot, true\}$ and since for every $v \in V$, either $\sigma'^p(v) = \bot$ or $\sigma'^p(v) = \sigma^{r'}(v)$, then $grd(t_a')(\sigma'^p) = \bot$. We choose some variable $v \in V$ s.t. $\sigma'^p(v) = \bot$ and v effects the evaluation of $grd(t_a')$ and refine Γ^A w.r.t. v.
     * If $s' \notin S_l^A$: then for some $k' \in \{1, ..., g\}$, $s' \in S(A^{k'})$. We refine the abstraction by removing s' from $S(A^k)$.

- Otherwise: we refine the abstraction by removing s from $S(A^k)$.

– $step'^p = TRANS(j, (t_1^a, ..., t_q^a))$
For every $i \in \{1, ..., q\}$ we match transition t_i^a with (possibly more than one) transition $t_i \in Trans_l$.

1. If $src(t_i^a) \in S_l$: Then $src(t_i^a) \in \omega_l^r$. By definition of the abstraction, there exists a matching transition $t \in Trans_l$ s.t. $src(t) = src(t_i^a)$, $trig(t) = trig(t_i^a)$, $grd(i) = grd(t_i^a)$, and $act(t) = act(t_i^a) = skip$ (since $trig(t_i^a) = \rho_l'^p \neq \epsilon$). If $enabled(t, C^r) = true$ then we define $t_i = t$.
Otherwise, if $enabled(t, C^r) = false$, then we need to refine. We separate between the different cases that can cause $enabled(t, C^r) = false$ and $enabled(t_i^a, C'^p) \in \{\bot, true\}$:
   - For every $t' \in Trans_l$ s.t. $src(t') \lhd src(t)$ and $src(t') \in \omega_l^r$ it holds that $enabled(t', C^r) = false$: Since $\rho_l^r = \rho_l'^p$, this means that $grd(t_i^a)(\sigma'^p) \in \{\bot, true\}$ and $grd(t)(\sigma^r) = false$. Since $\sigma^r \preceq \sigma'^p$ we conclude that $grd(t_i^a)(\sigma'^p) = \bot$. We choose some variable $v \in V$ s.t. $\sigma'^p(v) = \bot$ and v effects the evaluation of $grd(t_i^a)$ and refine Γ^A w.r.t. v.
   - There exists $t' \in Trans_l$ s.t. $src(t') \lhd src(t)$, $src(t') \in \omega_l^r$ and $enabled(t', C^r) = true$.
     (a) If $src(t') \in \omega_l'^p$, then by definition of the abstraction, there exists $t'_a \in Trans_l^A$ s.t. $src(t'_a) = src(t')$, $trig(t') = trig(t'_a)$, $grd(t') = grd(t'_a)$. Since $\rho_l^r = \rho_l'^p$, this means that $grd(t'_a)(\sigma'^p) \in \{false, \bot\}$ and $grd(t')(\sigma^r) = true$. Since $\sigma^r \preceq \sigma'^p$ we conclude that $grd(t'_a)(\sigma'^p) = \bot$. We choose some variable $v \in V$ s.t. $\sigma'^p(v) = \bot$ and v effects the evaluation of $grd(t'_a)$ and refine Γ^A w.r.t. v.
     (b) If $src(t') \notin \omega_l'^p$, then by definition of the abstraction, $src(t') \in S(A^k)$ for some $k \in \{1, ..., g\}$. Also, by definition of the simulation, $\Delta(A^k) \cap \omega_l'^p \neq \epsilon$. We refine the abstraction by removing $src(t')$ from $S(A^k)$.
2. If $src(t_i^a) \notin S_l$: Then $src(t_i^a) \notin \omega_l^r$. Since $trig(t_i^a) = \rho_l'^p \neq \epsilon$, then $src(t_i^a) = a_{strt}^k$ for some $k \in \{1, ..., g\}$.
   - If $trgt(t_i^a) = a_1^k$: If there exists a *maximal* set of *orthogonal* and *enabled* transitions $t'_1, ..., t'_{q'} \in Trans(A^k)$, then t_i is defined by $t'_1, ..., t'_{q'}$.
     If no such set of transitions exist, then we need to refine. Refine by removing a state $s' \in S(A^k) \cap \omega_l^r$ from $S(A^k)$.
   - Otherwise, this means that either (1) $trgt(t_i^a) = a_{strt}^{k'}$ for $k' \in \{1, ..., g\}$ and $k' \neq k$, or (2) for every $k' \in \{1, ..., g\}$, $trgt(t_i^a) \notin S_a(A^{k'})$ ($trgt(t_i^a) \in S_l$): By definition of the abstraction, there exists a matching transition $t' \in Trans_l$ s.t. $src(t') \in S_a(A^k)$, $trig(t') = trig(t_i^a)$, and
     * if $trgt(t_i^a) = a_{strt}^{k'}$ then $trgt(t') \in S(A^{k'})$
     * if $trgt(t_i^a) \in S_l$ then $trgt(t') = trgt(t_i^a)$

     Also, by definition of the abstraction, $grd(t_i^a) = grd(t')\&\bot$.
     (a) If $src(t') \in \omega_l^r$ and $enabled(t', C^r) = true$: then define $t_i = t'$.

(b) If $src(t') \in \omega_l^r$ and $enabled(t', C^r) = false$: then need to refine. We separate between the different cases that can cause $enabled(t', C^r) = false$ and $enabled(t_i^a, C'^p) \in \{\bot, true\}$:
* For every $t'' \in Trans_l$ s.t. $src(t'') \lhd src(t')$ it holds that $enabled(t'', C^r) = false$: Since $\rho_l^r = \rho_l'^p$, this means that $grd(t')(\sigma'^p) \in \{\bot, true\}$ and $grd(t')(\sigma^r) = false$. Since $\sigma^r \preceq \sigma'^p$ we conclude that $grd(t')(\sigma'^p) = \bot$. We choose some variable $v \in V$ s.t. $\sigma'^p(v) = \bot$ and v effects the evaluation of $grd(t')$ and refine Γ^A w.r.t. v.
* There exists $t'' \in Trans_l$ s.t. $src(t'') < src(t')$, $src(t'') \in \omega_l^r$ and $enabled(t'', C^r) = true$. By definition of the abstraction, $src(t'') \in S_a(A^k)$. We refine the abstraction by removing $src(t'')$ from the abstracted states.

(c) If $src(t') \notin \omega_l^r$, then need to refine. Remove $src(t')$ from $S(A^k)$.

We define $step^r = TRANS(j, (t_1, ..., t_q))$. For every such transition, since $\rho_l^r \neq \epsilon$, then for every $i \in \{1, ..., q\}$, either $act(t_i) = skip$ or $act(t_i)$ changes the value of cg^k variables. By definition of the abstraction, for every $i \in \{1, ..., q\}$, $act(t_i^a)$ possibly sets the value of variables from V to $\bot$ and changes the value of cg^k variables. Since $C^r \preceq C'^p$, then clearly $C^{r+1} \preceq C'^{p+1}$.

– $step'^p = Trans(j, t_a)$

Assume $id_j^r = id_j'^p = l$ and $src(t_a) = s$. We know that $enabled(t_a, C'^p) \in \{true, \bot\}$. We separate to the following different cases:

1. If $src(t_a) \in \omega_l^r$
2. If $src(t_a) \notin \omega_l^r$ and for some $k \in \{1, ..., g\}$, $src(t_a) = a_{end}^k$
3. If $src(t_a) \notin \omega_l^r$ and for some $k \in \{1, ..., g\}$, $src(t_a) = a_i^k$
4. If $src(t_a) \notin \omega_l^r$ and for some $k \in \{1, ..., g\}$, $src(t_a) = a_{strt}^k$

1. If $src(t_a) \in \omega_l^r$: By definition of the abstraction, there exists a matching transition $t \in Trans_l$ s.t. $src(t) = src(t_a)$, $trig(t) = trig(t_a)$, and $grd(t) = grd(t_a)$. If $enabled(t, C^r) = true$ then we define $step^r = TRANS(j, t)$. By definition of the abstraction construction, $act(t_a)$ differs from $act(t)$ in the following:
   - $act(t_a)$ might include manipulation of cg variables
   - assignments of type $v = e$ in $act(t)$ might be replaced with "if $(isIn(A^k))$ $v = \bot$; else $v = e$;" in $act(t_a)$.

   Since $C^r \preceq C'^p$ and specifically $\sigma^r \preceq \sigma'^p$, then clearly $act(t)(\sigma^r, C^r) \preceq act(t_a)(\sigma'^p, C'^p)$. By definition of the matching transition, either $trgt(t_a) = trgt(t)$ or $trgt(t_a) = a_{strt}^{k'}$ and $trgt(t) \in A^{k'}$. Thus, we can conclude that $C^{r+1} \preceq C'^{p+1}$.

   Otherwise, if $enabled(t, C^r) = false$, then we need to refine. We separate between the different cases that can cause $enabled(t, C^r) = false$ and $enabled(t_a, C'^p) \in \{\bot, true\}$:
   - For every $t' \in Trans_l$ s.t. $src(t') \lhd src(t)$ and $src(t') \in \omega_l^r$ it holds that $enabled(t', C^r) = false$: Since $\rho_l^r = \rho_l'^p$, this means that $grd(t_a)(\sigma'^p) \in \{\bot, true\}$ and $grd(t)(\sigma^r) = false$. Since $\sigma^r \preceq \sigma'^p$ we conclude that $grd(t_a)(\sigma'^p) = \bot$. We choose some variable $v \in V$ s.t. $\sigma'^p(v) = \bot$ and v effects the evaluation of $grd(t_a)$ and refine Γ^A w.r.t. v.

- There exists $t' \in Trans_l$ s.t. $src(t') \lhd src(t)$, $src(t') \in \omega_l^r$ and $enabled(t', C^r) = true$.
  (a) If $src(t') \in \omega_l'^p$, then by definition of the abstraction, there exists a matching transition $t'_a \in Trans_l^A$ s.t. $src(t'_a) = src(t')$, $trig(t') = trig(t'_a)$, $grd(t') = grd(t'_a)$. Since $\rho_l^r = \rho_l'^p$, this means that $grd(t'_a)(\sigma'^p) \in \{false, \bot\}$ and $grd(t')(\sigma^r) = true$. Since $\sigma^r \preceq \sigma'^p$ we conclude that $grd(t'_a)(\sigma'^p) = \bot$. We choose some variable $v \in V$ s.t. $\sigma'^p(v) = \bot$ and v effects the evaluation of $grd(t'_a)$ and refine Γ^A w.r.t. v.
  (b) If $src(t') \notin \omega_l'^p$, then by definition of the abstraction, $src(t') \in S(A^k)$ for $k \in \{1, ..., g\}$. Since $c_l^r \preceq c_l'^p$, this means that $a_{strt}^k \in \omega_l'^p$ (it is not possible that one of $\Delta(A^k) \setminus \{a_{strt}^k\}$ is in $\omega_l'^p$ since these states have null outgoing transitions, and thus $enabled(t_a, C'^p) = false$). We separate between two cases, whether or not the abstraction has been traversed.
     * If there exists an abstracted variable $v \in V(A^k)$ s.t. $\sigma'^p(v) \neq \bot$: this means that the abstraction has just been entered. We refine the abstraction by removing $src(t')$ from the abstracted states.
     * If for every abstracted variable $v \in V(A^k)$, $\sigma'^p(v) = \bot$: this means that the abstraction has been traversed. It is possible that the concrete model might reach a state where $enabled(t, C^{r'}) = true$ after traversing abstracted transitions without GEN. Continue from C^r the RTC step on abstracted transitions (transitions $t' \in Trans(A^k)$), as long as it is on transitions without GEN. Since by the definition of the abstraction, these transitions can only modify variables from $V(A^k)$, which have the value $\bot$ from the abstraction, then on all such reachable configurations $C^{r'} \preceq C'^p$. For *every* such reachable $C^{r'}$, if $enabled(t, C^{r'})$ then we define $step^{r'} = TRANS(j, t)$.
       Otherwise, if on all possible reachable configurations $C^{r'}$ it holds that $enabled(t, C^{r'}) = false$, then we need to refine.
       · If for some reachable configuration $C^{r'}$, and for every $t'' \in Trans_l$ s.t. $src(t'') \lhd src(t)$ and $src(t'') \in \omega_l^{r'}$ it holds that $enabled(t'', C^{r'}) = false$, then this means that $grd(t)(\sigma^{r'}) = false$. Since $enabled(t_a, C'^p) = true$, then $grd(t_a)(\sigma'^p) \in \{\bot, true\}$. Since $\sigma^r \preceq \sigma'^p$ we conclude that $grd(t_a)(\sigma'^p) = \bot$. We choose some variable $v \in V$ s.t. $\sigma'^p(v) = \bot$ and v effects the evaluation of $grd(t_a)$ and refine Γ^A w.r.t. v.
       · Otherwise, refine the abstraction by removing $src(t')$ from the abstracted states.
2. If $src(t_a) \notin \omega_l^r$ and for some $k \in \{1, ..., g\}$, $src(t_a) = a_{end}^k$. By definition of the abstraction, $trgt(t_a) = a_{strt}^k$. Continue on $step'^p$ without matching a step on π. It holds that $C^r \preceq C'^{p+1}$.
3. If $src(t_a) \notin \omega_l^r$ and for some $k \in \{1, ..., g\}$, $src(t_a) = a_i^k$: By definition of the abstraction, $trgt(t_a)$ is either a_{end}^k or a_{i+1}^k

- If $trgt(t_a) = a^k_{end}$: Continue on $step'^p$ without matching a step on π. It holds that $C^r \preceq C'^{p+1}$.
- If $trgt(t_a) = a^k_{i+1}$: By definition of the abstraction, $act(t_a) = GEN(EV(A^k))$. Assume the event generated on $step'^p$ is $ev \in EV(A^k)$. Continue from C^r the RTC step on abstracted transitions (transitions $t' \in Trans(A^k)$), as long as it is on transitions without GEN. Since by the definition of the abstraction, these transitions can only modify variables from $V(A^k)$, which have the value $\perp$ from the abstraction, then on all of them $C^{r'} \preceq C'^p$. On *every* such reachable $C^{r'}$, if there exists a transition $t'_c \in Trans(A^k)$ where $GEN(ev) \in act(t'_c)$ and $enabled(t'_c, C^{r'}) = true$ then $step^{r'} = Trans(j, t'_c)$. For same reasoning as before, $C^{r'+1} \preceq C'^{p+1}$.
  Otherwise, if on all possible reachable configurations $C^{r'}$ no such t'_c, then need to refine. For some $s \in \omega^r_l$ s.t. $s \in S(A^k)$, remove s from $S(A^k)$.

4. If $src(t_a) \notin \omega^r_l$ and for some $k \in \{1, ..., g\}$, $src(t_a) = a^k_{strt}$: We separate between the different cases for $trgt(t_a)$
   - $trgt(t_a) = a^k_1$: Since $\rho'^p_l = \epsilon$, then $t_a = \tau^k_2$. For every abstracted transition $t \in Trans(A^k)$ s.t. $enabled(t, C^r)$ define $step^r = TRANS(j, t)$. By definition of the abstraction, $modif(t) \subseteq V(A^k)$. Since for every $v \in V(A^k)$, $\sigma'^{p+1}(v) = \perp$, and since $C^r \preceq C'^p$, then $C^r \preceq C'^{p+1}$.
     Otherwise, if no such t exists, then need to refine. For some $s \in \omega^r_l$ s.t. $s \in S(A^k)$, remove s from $S(A^k)$.
   - Otherwise: This means that either (1) $trgt(t_a) \in S_l$ or (2) $trgt(t_a) = a^{k'}_{strt}$ for $k' \in \{1, ..., g\}$ and $k' \neq k$: By definition of the abstraction, there exists a matching transition $t \in Trans_l$ s.t. $src(t) \in S_a(A^k)$, $trig(t) = trig(t_a)$, and $grd(t_a) = grd(t)\&\perp$.
     (a) If $src(t) \in \omega^r_l$ and $enabled(t, C^r) = true$: then define $step^r = TRANS(j, t)$.
     By definition of the abstraction construction, $act(t_a)$ differs from $act(t)$ in the following:
     * $act(t_a)$ might include manipulation of cg variables
     * assignments of type $v = e$ in $act(t)$ might be replaced with "if $(isIn(A^k))$ $v = \perp$; else $v = e$;" in $act(t_a)$.

     Since $C^r \preceq C'^p$ and specifically $\sigma^r \preceq \sigma'^p$, then clearly $act(t)(\sigma^r, C^r) \preceq act(t_a)(\sigma'^p, C'^p)$. By definition of the matching transition, either $trgt(t_a) = trgt(t)$ or $trgt(t_a) = a^{k'}_{strt}$ and $trgt(t) \in A^{k'}$. Thus, we can conclude that $C^{r+1} \preceq C'^{p+1}$.
     (b) If $src(t) \in \omega^r_l$ and $enabled(t.C^r) = false$: then need to refine. We separate between the different cases that can cause $enabled(t, C^r) = false$ and $enabled(t_a, C'^p) \in \{\perp, true\}$:
     * For every $t' \in Trans_l$ s.t. $src(t') \lhd src(t)$ it holds that $enabled(t', C^r) = false$: Since $\rho^r_l = \rho'^p_l$, this means that $grd(t)(\sigma'^p) \in \{\perp, true\}$ and $grd(t)(\sigma^r) = false$. Since $\sigma^r \preceq \sigma'^p$ we conclude that $grd(t)(\sigma'^p) = \perp$. We choose some variable $v \in V$ s.t. $\sigma'^p(v) =$

$\perp$ and v effects the evaluation of $grd(t_a)$ and refine Γ^A w.r.t. v.

* There exists $t' \in Trans_l$ s.t. $src(t') \lhd src(t)$, $src(t') \in \omega_l^r$ and $enabled(t', C^r) = true$. By definition of the abstraction $src(t') \in S(A^k)$. We refine the abstraction by removing $src(t')$ from $S(A^k)$.

(c) If $src(t) \notin \omega_l^r$, then we need to refine. For some $s \in \omega_l^r$ s.t. $s \in S(A^k)$, remove s from $S(A^k)$.

If we were able to construct π s.t. $\pi \preceq \pi^A$, then one of the following holds: (a) If $(\pi^A \models \psi) = false$ then no need to check π. By construction, $\pi \not\models \psi$, and we can conclude that $\Gamma \not\models A\psi$, (b) If $(\pi^A \models \psi) = \perp$ then we check π w.r.t. ψ. If $\pi \not\models \psi$ then again π is a concrete CEX and we conclude that $\Gamma \not\models A\psi$. Otherwise $(\pi \models \psi)$, the abstraction is too coarse and we need to refine. Notice that in the latter case, since $(\pi^A \models \psi) = \perp$ then there exists $v \in V$ which effects the value of ψ, and v has the value $\perp$. We then refine Γ^A in order to have a concrete value on v.

Consider the example system presented in section 6, and consider a property that addresses the variable *space*. Recall that under the abstraction presented for this example, such a property is evaluated to $\perp$, since the variable *space* is abstracted. During the refinement, state $WaitForDB$ is suggested for refinement, and is removed from the abstraction. We can then create a *refined* system Γ'^A, where DB objects are abstracted w.r.t. a new abstraction set $A' = \{Idle, PriceProcessor, UpdateDB\}$. The property can then be verified on Γ'^A, and we can conclude that it holds on the concrete system.

7.1 Refining Γ^A w.r.t. variable v for which $\sigma'^p(v) = \perp$:

We trace π^A back to find the variable that gave v the value $\perp$. The only place where a variable is initially assigned the value $\perp$ is a transition from a_{strt}^k to a_1^k in some $\Delta(A_i)$. Thus, the tracing back of π^A terminates at C'^α s.t. $a_{strt}^k \in \omega_i^\alpha$.

Formally: for $\alpha = p - 1$ until $\alpha = 0$, if $step'^\alpha = Trans(j, (t_1, ..., t_y))$, and for some $t \in \{t_1, ..., t_y\}$ $v \in modif(act(t))$ then if the value of v is based on the value of v' for which $\sigma'^\alpha(v') = \perp$ then continue tracing back on v'. Stop tracing back on v' at $step'^\alpha = TRANS(j, (t_1, ..., t_y))$ where $\sigma'^\alpha(v') \neq \perp$, $\sigma'^{\alpha+1}(v') = \perp$, and the value of v' does not depend on any variable ($v' := \perp$ is executed on the action).

We find the matching Γ-configuration C^β in π s.t. $C^\beta \preceq C'^\alpha$, and refine the model by removing from the abstraction a state $s \in S(A_i^k)$ s.t. $s \in \omega_i^\beta$.

8 Conclusion

In this work we presented a CEGAR-like method for abstraction and refinement of behavioral UML systems.

It is important to note that our framework is completely automatic. An initial abstraction can be one that abstracts entire state machines, based on the given

property. We presented a basic and automatic refinement method. Heuristics can be applied during the refinement stage in order to converge in less iterations. For example, when refining due to a variable v whose value is $\perp$, we can refine by adding all abstracted transitions that modify v (or v's cone-of-influence). Note, however, that there always exists a tradeoff between quick convergence and the growth in size of the abstract system.

References

1. Istvn Majzik Adm Darvas and Balzs Beny. Verification of UML statechart models of embedded systems. In *In Proc. 5th IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop (DDECS 2002), IEEE Computer Society TTTC*, pages 70–77, 2002.
2. Grady Booch, James E. Rumbaugh, and Ivar Jacobson. The unified modeling language user guide. *J. Database Manag.*, 10(4):51–52, 1999.
3. William Chan, Richard J. Anderson, Paul Beame, Steve Burns, Francesmary Modugno, David Notkin, and Jon Damon Reese. Model checking large software specifications. *IEEE Trans. Software Eng.*, 24(7):498–520, 1998.
4. E. M. Clarke and W. Heinle. Modular translation of statecharts to smv. Technical Report CMU-CS-00-XXX, Carnegie-Mellon University School of Computer Science, 2000.
5. Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. *Journal of the ACM*, 50(5):752–794, 2003.
6. E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT press, December 1999.
7. Werner Damm, Bernhard Josko, Amir Pnueli, and Angelika Votintseva. Understanding UML: A formal semantics of concurrency and communication in real-time UML. In *Formal Methods for Components and Objects (FMCO'02)*, volume 2852 of *LNCS*, pages 71–98, Leiden, The Netherlands, November 2002. Springer.
8. Jori Dubrovin and Tommi A. Junttila. Symbolic model checking of hierarchical uml state machines. In *Application of Concurrency to System Design (ACSD'08)*, pages 108–117, Xi'an, China, June 2008. IEEE.
9. Harald Fecher, Michael Huth, Heiko Schmidt, and Jens Schönborn. Refinement sensitive formal semantics of state machines with persistent choice. *Electron. Notes Theor. Comput. Sci.*, 250(1):71–86, September 2009.
10. Harald Fecher and Jens Schönborn. Uml 2.0 state machines: Complete formal semantics via core state machines. In *Proceedings of the 11th International Workshop, FMICS 2006 and 5th International Workshop, PDMC Conference on Formal Methods: Applications and Technology*, FMICS'06/PDMC'06, pages 244–260, Berlin, Heidelberg, 2007. Springer-Verlag.
11. Object Management Group. OMG Unified Modeling Language (UML) Infrastructure, version 2.4. ptc/2010-11-16, 2010.
12. Orna Grumberg, Yael Meller, and Karen Yorav. Applying software model checking techniques for behavioral UML models. In *Formal Methods (FM'12)*, volume 7436 of *LNCS*, pages 277–292, Paris, France, August 2012. Springer.
13. Arie Gurfinkel and Marsha Chechik. Why waste a perfectly good abstraction? In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, LNCS, pages 212–226. Springer, 2006.

14. Diego Latella, István Majzik, and Mieke Massink. Automatic verification of a behavioural subset of UML statechart diagrams using the spin model-checker. *Formal Asp. Comput.*, 11(6):637–664, 1999.
15. Shuang Liu, Yang Liu, tienne Andr, Christine Choppy, Jun Sun, Bimlesh Wadhwa, and JinSong Dong. A formal semantics for complete UML state machines with communications. In *integrated Formal Methods (iFM'13)*, volume 7940 of *LNCS*, pages 331–346. Springer, June 2013.
16. Kumar Madhukar, Ravindra Metta, Priyanka Singh, and R. Venkatesh. Reachability verification of rhapsody statecharts. In *International Conference on Software Testing, Verification and Validation Workshops (ICSTW '13)*, pages 96–101, Washington, DC, USA, 2013. IEEE Computer Society.
17. Yael Meller, Orna Grumberg, and Karen Yorav. Verifying behavioral UML systems via CEGAR. In *integrated Formal Methods (iFM'13)*, LNCS. Springer, September 2014.
18. Erich Mikk, Yassine Lakhnech, Michael Siegel, and Gerard J. Holzmann. Implementing statecharts in promela/spin. In *Workshop on Industrial-Strength Formal Specification Techniques (WIFT'98)*, pages 90–101, Boca Raton, FL, USA, October 1998. IEEE Computer Society.
19. Iulian Ober, Susanne Graf, and Ileana Ober. Validating timed UML models by simulation and verification. *STTT*, 8(2):128–145, 2006.
20. IST-2001-33522 OMEGA. http://www-omega.imag.fr, 2001.
21. Ivan Paltor and Johan Lilius. Formalising UML state machines for model checking. In *The Unified Modeling Language - Beyond the Standard (UML'99)*, volume 1723 of *LNCS*, pages 430–445, Fort Collins, CO, USA, October 1999. Springer.
22. A. Pnueli. The temporal logic of programs. In *Proceedings of the Eighteenth Annual Symposium on Foundations of Computer Science (FOCS'77)*, 1977.
23. Christian Prehofer. Behavioral refinement and compatibility of statechart extensions. *Electron. Notes Theor. Comput. Sci.*, 295:65–78, May 2013.
24. Greg Reeve and Steve Reeves. Logic and refinement for charts. In *Proceedings of the 29th Australasian Computer Science Conference - Volume 48*, ACSC '06, pages 13–23, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.
25. Ingo Schinz, Tobe Toben, Christian Mrugalla, and Bernd Westphal. The rhapsody UML verification environment. In *Software Engineering and Formal Methods (SEFM'04)*, pages 174–183, Beijing, China, September 2004. IEEE Computer Society.
26. Peter Scholz. Incremental design of statechart specifications. *Sci. Comput. Program.*, 40(1):119–145, May 2001.
27. Carl-Johan H. Seger and Randal E. Bryant. Formal verification by symbolic evaluation of partially-ordered trajectories. *Form. Methods Syst. Des.*, 6(2):147–189, March 1995.
28. A. J. H. Simons, M. P. Stannett, K. E. Bogdanov, and W. M. L. Holcombe. Plug and play safely: Rules for behavioural compatibility. In *International Conference on Software Engineering and Applications (IASTED'02)*, pages 263–268, Cambridge, MA, USA, 2002.