Stuttering Abstraction for Model Checking

Shiva Nejati Arie Gurfinkel Marsha Chechik
Department of Computer Science, University of Toronto
Email: {shiva,arie,chechik}@cs.toronto.edu

Abstract

Abstraction is one of the most effective approaches to improving the applicability and the scalability of model-checking. The goal of abstraction is to construct a model which is small enough to analyze, yet contains enough detail to allow conclusive analysis of properties of interest. For a given concrete model, the size of its smallest possible abstraction is intimately related to the set of temporal properties preserved by the abstraction. Thus, smaller abstractions are possible if we reduce this set, for example, by disallowing the use of the next-time operator. In this paper, we improve the conclusiveness and efficiency of the 3-valued abstraction framework. We start by proposing a number of simulation relations that preserve true properties expressed in subsets of CTL without the next-time operator. We show how these simulation relations are extended into refinement relations for defining 3-valued abstractions. Using these refinement relations, we give a new abstraction method that results in more conclusive abstract models.

1 Introduction

Abstraction is arguably the most effective technique for dealing with the state explosion problem in model-checking. The goal of abstraction is to build an approximation of a model which is smaller than the original model, and yet enables conclusive analysis of properties of interest.

Typically, the set of behaviors of an abstract model either subsumes, i.e. *over-approximates*, or is subsumed by, i.e. *under-approximates*, that of its corresponding concrete model. In this paper, we consider branching-time logic CTL [6]. Abstraction based on over-approximation preserves truth of universally-quantified CTL properties (ACTL), i.e. if an ACTL property holds in the abstract model, it is guaranteed to hold in the concrete one. Dually, under-approximation abstraction preserves falsity of ACTL. The situation is reversed for existentially-quantified CTL properties (ECTL): over-approximation preserves falsity, and under-approximation preserves truth. In order to reason about full CTL, we adopt the approach taken in [16, 19, 8], where abstract models have both types of behaviors: those used for the computation of universal properties, and those for existential. These models can be represented by 3-valued formalisms [13], enabling 3-valued valuations of CTL properties. If these properties evaluate to *true* or *false* in the abstract model, then they are *true* or *false* in the concrete model, respectively [16]. Otherwise, CTL properties can evaluate to *unknown*, which yields no information about their values in the concrete model.

Whenever the analysis is inconclusive, the abstract model must be refined to a more precise one [5]. Rebuilding and refining an abstract model is expensive; moreover, each

(a)
$P ::$
$\quad input: \; x > 0$
$\quad pc = 1: \; \mathbf{if}\; x > 5 \;\mathbf{then}\; x := x + 5 \;\mathbf{else}\; x := x + 10 \;\mathbf{fi}$
$\quad pc = 2: \; \mathbf{while}\; true \;\mathbf{do\; if}\; odd(x) \;\mathbf{then}\; x := -1 \;\mathbf{else}\; x := x + 1 \;\mathbf{fi\; od}$

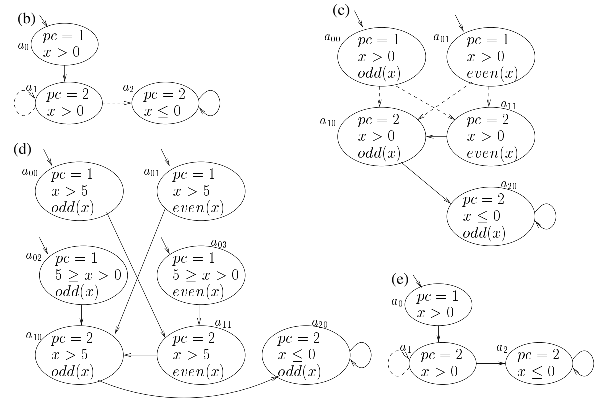

Figure 1. A concrete program P (a) and its abstractions: (b) the 3-valued abstraction of (a); (c) the refinement of (b) with predicate $odd(x)$; (d) the refinement of (c) with predicate $x > 5$; and (e) the stuttering abstraction of (a).

(a)
$P ::$
$\quad x > 0$
$\quad \mathbf{while}\; true \;\mathbf{do}\; x = x - 1 \;\mathbf{od}$

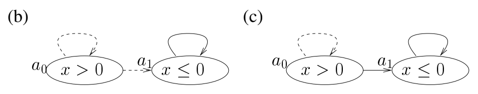

Figure 2. A program P (a) and its abstractions: (b) 3-valued; and (c) stuttering.

refinement typically increases the state-space of the abstract model leading to models that are too large to analyze. The goal of this work is to improve the conclusiveness and efficiency of the 3-valued abstraction framework.

To motivate our work, we use the example from [27], shown in Fig. 1. Suppose we are interested in checking the safety property "is it possible to reach a state in which $x \leq 0$", formalized as $\varphi = EF(x \leq 0)$ and clearly satisfied by the program in Fig. 1(a). Since the state-space of the concrete model is infinite, we construct an abstract model (see Fig. 1(b)) where all of the states that agree on the predicate $x \leq 0$ and on the value of the process counter *pc* are collapsed into a single abstract state. This process results in three abstract states, referred to as a_0, a_1 and a_2. For example, a_2 corresponds to all states of the program where $pc = 2$ and $x \leq 0$. Transition relation of the abstract model is constructed as follows: a transition between abstract states a_i and a_j is (a) *true* if for every concrete state mapped to a_i, there exists a transition to some concrete state mapped to a_j (the $\forall\exists$ condition [8]); (b) *maybe*, or *unknown*, if there exists a transition from some concrete state mapped to a_i to some concrete state mapped to a_j (the $\exists\exists$ condition [8]); (c) *false*, or absent, if neither of the above conditions hold. We use solid and dashed lines to represent *true* and *maybe* transitions, respectively. In our example, we are certain that the program transitions from a_0 to a_1, whereas it can possibly get stuck in a_1 or move to state a_2. The value of the property $EF(x \leq 0)$ in the model in Fig. 1(b) is *maybe*, as illustrated by the path a_0, a_1, a_2 containing a *maybe* transition.

Aiming to get a conclusive result, we refine the model using predicate $odd(x)$. The result, shown in Fig. 1(c), is still inconclusive for φ, leading to another refinement. This time, we add the predicate $x > 5$ (see Fig. 1(d)), and obtain a model that satisfies φ. As illustrated by this example, the process of computing a conclusive abstract model may require many refinement steps and may fail to converge if the models become too large. Furthermore, adding a predicate always increases the statespace, but is not guaranteed to increase conclusiveness of the resulting model with respect to properties under analysis.

Consider an alternative approach, referred to as *stuttering* abstraction. Rather than refining and thus increasing the state-space of the abstract model, we relax conditions for computing abstract transitions so that fewer of them are *maybe*. In particular, we set an abstract transition between states a_i and a_j to *true* if every concrete state mapped to a_i can *reach* some state mapped to a_j. This condition relaxes the $\forall\exists$ condition. Fig. 1(e) shows an example of this transformation. The transition between a_1 and a_2 is *true* in this model because every concrete state satisfying $pc = 2$ and $x > 0$ can reach, in one or two steps, a state satisfying $x \leq 0$. The resulting model may not preserve properties containing a next-time operator, but is conclusive for φ. Note that this model is much smaller than the one in Fig. 1(d).

In this paper, we explore several relations, like the one presented above, for building abstract models that facilitate a more precise analysis on "small" state-spaces at the expense of dropping the preservation of certain temporal operators. In particular, we choose to drop the next-time operators. Abstract and concrete models are specified at a different level of abstraction, where a sequence of low-level states may be mapped into a single high-level abstract state [1]. The next-time operators, used for reasoning about states reached via a single transition, are not meaningful in this context. In fact, it has been argued [18] that the next-time operators should not be used in specifications (or high-level models) at all, because they make the specifications more expressive than what they should really be.

The main contribution of the paper is a new abstraction method for building 3-valued abstract models. When compared to the traditional 3-valued approach, our abstract models are significantly smaller, while being conclusive for subsets of CTL properties. Not only can our method avoid the application of unnecessary refinement steps, it may allow us to construct conclusive models when the conventional abstraction refinement method fails to converge altogether. For example, Fig. 2 illustrates a program P (Fig. 2(a)) and its 3-valued abstraction (Fig. 2(b)) obtained using the predicate $x > 0$. Consider the property $\varphi = EF(x \leq 0)$ which holds in P. Since the domain of x is unbounded, a 3-valued abstract model conclusive for $\varphi = EF(x \leq 0)$ cannot be obtained after a *finite* number of refinement steps. However, the abstract model computed using our approach (see Fig. 2(c)) is conclusive for φ. Its success comes from the fact that every concrete state where $x > 0$ can reach a state where $x \leq 0$ in a finite number of steps; thus, the transition between a_0 and a_1 in the model in Fig. 2(c) is *true*.

The rest of this paper is organized as follows. We start by giving the necessary background in Section 2. We study a number of simulation relations characterized by subsets of ECTL in Section 3. Further, we show how these simulation relations can be extended into the corresponding refinement relations. In Section 4, we propose a general abstraction method for building abstract models based on the refinement relations introduced in Section 3. We discuss efficient computation of abstract models and their potential applications in Sections 5 and 6. Section 7 concludes the paper. Proof sketches of theorems are given in the Appendix.

2 Background

In this section, we fix the notation and review CTL model-checking over 3-valued (Kleene) logic.

Kleene logic. Kleene logic [17] is a 3-valued logic with elements t, m, and f, interpreted as *true*, *maybe*, and *false*, respectively. These values form a lattice under the truth ordering $\sqsupseteq$ as shown in Fig. 4(a). Conjunction and disjunction in this logic are interpreted as meet and join of the lattice, respectively. For example, $\mathsf{t} \wedge \mathsf{m} = \mathsf{m}$ and $\mathsf{m} \vee \mathsf{t} = \mathsf{t}$. In addition, negation is: $\neg\mathsf{t} = \mathsf{f}$; $\neg\mathsf{m} = \mathsf{m}$; $\neg\mathsf{f} = \mathsf{t}$. For convenience, we denote the Kleene logic by $\mathbf{3}$, and the classical logic (with elements t and f) by $\mathbf{2}$.

The values of the Kleene logic can also be ordered by an *information ordering* $\preceq$, forming a meet-semilattice shown in Fig. 4(b). Intuitively, $a \preceq b$ means that a contains less information, or is less complete, than b. Thus, t and f are incomparable, and are both more complete than m.

Models. We model systems using 3-valued Kripke structures. A 3-valued Kripke structure is a tuple $K = (\Sigma, s_0, A, I, R)$, where Σ is a finite set of states; $s_0 \in \Sigma$ is a unique initial state; A is a set of atomic propositions; $I : \Sigma \times A \to \mathbf{3}$ is an interpretation function that assigns a value to each atomic proposition in each state; and $R : \Sigma \times \Sigma \to \mathbf{3}$ is a total transition relation that assigns a value to each transition between a given pair of states. For any two states s and t, $R(s,t) = \mathsf{f}$ is interpreted as the absence of a transition between s and t. By convention, such transitions are not shown in the graphical presentation of the model. An example of a 3-valued Kripke structure is given in Fig. 5(c), where the values of q in a_0 and p in a_1 are m, and transitions (a_1, a_2), (a_0, a_1), (a_1, a_1) have values t, m and f, respectively. Recall that dashed and solid lines represent m- and t-transitions, respectively.

Clearly, any classical Kripke structure can be seen as a 3-valued Kripke structure that does not make use of the value m, i.e., its interpretation function is $I : \Sigma \times A \to \mathbf{2}$, and its transition relation is $R : \Sigma \times \Sigma \to \mathbf{2}$. We refer to such structures as *classical* or *2-valued* Kripke structures. There are several ways to extend Kripke structures to Kleene logic (e.g., Modal Transition Systems (MTS) [19], Kripke Modal Transition Systems (KMTS) [16]); however, all of them are equally expressive [13].

A 3-valued Kripke structure $K = (\Sigma, s_0, A, I, R)$ can be decomposed into two classical Kripke structures that capture all *definite* and all *possible* behaviors of K [16]. We refer to these behaviors of K as $K^{\sqsupseteq \mathsf{t}} = (\Sigma, s_0, A, I^{\sqsupseteq \mathsf{t}}, R^{\sqsupseteq \mathsf{t}})$ and $K^{\sqsupseteq \mathsf{m}} = (\Sigma, s_0, A, I^{\sqsupseteq \mathsf{m}}, R^{\sqsupseteq \mathsf{m}})$, respectively, where $R^{\sqsupseteq l}$ and $I^{\sqsupseteq l}$ are defined as $R^{\sqsupseteq l}(s,t) \triangleq R(s,t) \sqsupseteq l$ and $I^{\sqsupseteq l}(s,p) \triangleq I(s,p) \sqsupseteq l$ for $l \in \{\mathsf{t}, \mathsf{m}\}$. Classical Kripke structures do not distinguish between possible and definite behaviors, i.e., $K^{\sqsupseteq \mathsf{m}} = K^{\sqsupseteq \mathsf{t}}$.

A path emanating from a state s of a classical Kripke structure K is a sequence of states $s_0, s_1, \ldots$, such that $s_0 = s$ and $R(s_i, s_{i+1})$ for every $i \geq 0$. A path π is *finite* if the sequence $s_0, s_1, \ldots$ is finite, and *infinite* otherwise. A set of all paths from s is denoted by $\Pi(s)$, and the ith state of a given path $\pi \in \Pi(s)$ – by π_i. A sequence of states π is a path in a 3-valued Kripke structure K iff it is a path in $K^{\sqsupseteq \mathsf{m}}$.

$$
\begin{array}{lcl}
||\ell||(s) & \triangleq & \ell \\
||p||(s) & \triangleq & I(s,p) \\
||\varphi \wedge \psi||(s) & \triangleq & ||\varphi||(s) \wedge ||\psi||(s) \\
||\varphi \vee \psi||(s) & \triangleq & ||\varphi||(s) \vee ||\psi||(s) \\
||\neg\varphi||(s) & \triangleq & \neg||\varphi||(s) \\
||EX\varphi||(s) & \triangleq & \bigvee_{s' \in \Sigma} (R(s,s') \wedge ||\varphi||(s')) \\
||EG\varphi||(s) & \triangleq & \bigvee_{\pi \in \Pi(s)} \bigwedge_{i \geq 0} (||\varphi||(\pi_i) \wedge R(\pi_i, \pi_{i+1})) \\
||E[\varphi U \psi]||(s) & \triangleq & \bigvee_{\pi \in \Pi(s)} \bigvee_{i \geq 0} \big(||\psi||(\pi_i) \wedge \\
 & & \quad \bigwedge_{j<i} (R(\pi_j, \pi_{j+1}) \wedge ||\varphi||(\pi_j))\big)
\end{array}
$$

Figure 3. 3-valued semantics of CTL.

Temporal logic. *Computational Tree Logic* (CTL) [6] is a branching-time temporal logic defined by the following grammar:

$$
\begin{array}{rcl}
\varphi & = & \ell \mid p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \neg\varphi \mid EX\varphi \mid AX\varphi \mid EF\varphi \\
 & & \mid AF\varphi \mid EG\varphi \mid AG\varphi \mid E[\varphi\ U\ \varphi] \mid A[\varphi\ U\ \varphi],
\end{array}
$$

where $p \in A$ is an atomic proposition and ℓ is a constant. The meaning of the temporal operators is: given a state and paths emanating from it, φ holds in one (EX) or all (AX) next states; φ holds in some future state along one (EF) or all (AF) paths; φ holds globally along one (EG) or all (AG) paths, and φ holds until a point where ψ holds along one (EU) or all (AU) paths. Some properties which hold in the Kripke structure in Fig. 5(a) are $AG(p \Rightarrow EFq)$, $EG(\neg p \vee \neg q)$ and AXq. We use the term *witness* to refer to a path or a collection of paths that are used to prove an existential property. A witness for EU and EF is a finite path, whereas a witness for EG is an infinite path.

We write $||\varphi||^K(s)$ to indicate the value of φ in the state s of K, and $||\varphi||(s)$ when K is clear from the context. The value of the formula φ in a 3-valued Kripke structure K is its value in the initial state, i.e., $||\varphi||^K \triangleq ||\varphi||^K(s_0)$. Temporal operators EX, EG, and EU together with the propositional connectives form an adequate set (i.e., all other operators can be defined from them). For example, $EF\varphi \triangleq E[\mathsf{t}\ U\ \varphi]$ and $AG\varphi \triangleq \neg EF\neg\varphi$. The formal 3-valued semantics of CTL is given in Fig. 3. For example, $||EXq||(a_0) = \mathsf{m}$ in the model shown in Fig. 5(c). The value is obtained because q is f in a_2 and t in a_1, and the transitions from a_0 to these states are m. In the case of classical Kripke structures, the definition of CTL in Fig. 3 is equivalent to its classical interpretation.

We identify several fragments of CTL. A formula φ is said to be *positive* if it does not contain any negations. We say that φ is *existential* (or in ECTL) if it is positive and only contains existential temporal operators. ECTL$_{-X}$ and ECTL$_U$ are fragments of ECTL where the temporal connectives are $\{EU, EG\}$ and $\{EU\}$, respectively. CTL$_{-X}$ and CTL$_U$ are formed by adding negation to ECTL$_{-X}$ and ECTL$_U$, respectively.

Logical characterization and preservation. Let L be a subset of CTL, $K_1 = (\Sigma_1, s_0, A_1, I_1, R_1)$ and $K_2 =$

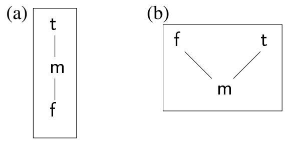

Figure 4. Kleene logic: (a) truth ordering, and (b) information ordering.

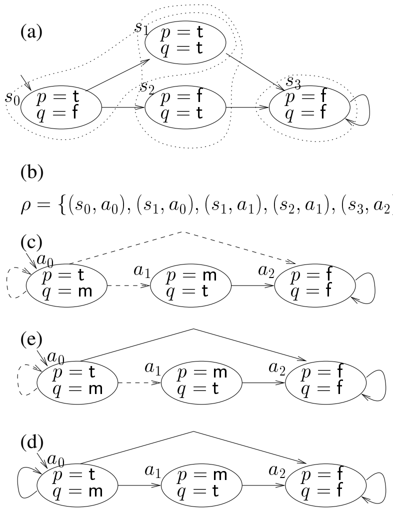

Figure 5. A concrete model (a), the abstraction relation ρ (b), and the abstractions of (a) built based on ρ: (c) the 3-valued abstraction, (d) the divergence-blind abstraction, and (e) the stuttering abstraction.

$(\Sigma_2, t_0, A_2, I_2, R_2)$ be 2-valued Kripke structures, and $\rho \subseteq \Sigma_1 \times \Sigma_2$ be some relation over K_1 and K_2. Then for every $s \in \Sigma_1$ and $t \in \Sigma_2$,

$\rho(s,t) \Rightarrow \forall \varphi \in L \cdot (||\varphi||(t) \Rightarrow ||\varphi||(s))$ iff *ρ preserves L*
$\rho(s,t) \Leftrightarrow \forall \varphi \in L \cdot (||\varphi||(t) \Rightarrow ||\varphi||(s))$ iff ρ is *logically characterized by L*

Clearly, if ρ is logically characterized by L, it also preserves L.

When K_1 and K_2 are 3-valued Kripke structures, L is interpreted w.r.t. its 3-valued semantics, and the notion of preservation and logical characterization changes from implication to *less than* in the information ordering, i.e. from $||\varphi||(t) \Rightarrow ||\varphi||(s)$ to $||\varphi||(t) \preceq ||\varphi||(s)$.

3 From Simulation to Refinement

In this section, we define three 3-valued refinement relations that are logically characterized by different subsets of CTL.

We start with three simulation relations: *classical*, *divergence-blind*, and *stuttering* and then extend them to refinement relations. Classical simulation is characterized by full ECTL, while the other two only preserve subsets of ECTL without the next-time operator. Divergence-blind simulation (adapted from the notion of divergence-blind equivalence [26]) is limited to reasoning about existential properties with finite-length witnesses, whereas stuttering simulation drops this limitation. For a given simulation that preserves a fragment L of ECTL, a corresponding refinement is a relation that preserves L together with negation. Here, we apply this extension only to our simulation relations, but it can be used for arbitrary simulations as well.

We follow the approach taken in [16] for lifting simulation to refinement. It can be seen as combining two simulation relations: one preserving truth, and the other, defined in the reverse direction, preserving falsehood. The key point is that these simulation relations should only preserve positive logical properties. However, in conventional simulation relations, two similar states have the same set of positive and negative atomic propositions, and thus they preserve negated atomic proposition. Therefore, we need to redefine the simulation relations so that only positive properties are preserved.

3.1 Simulation Relations

In this section, we present modified versions of three existing simulation relations: classical simulation [23], divergence-blind simulation [26], and stuttering simulation [3, 26, 22] and subsets of ECTL that characterize them. Our definitions relax the condition on preservation of atomic propositions from *equality* to *implication*. That is, atomic propositions that hold in the less refined state also hold in the more refined one, but not vice versa.

Classical simulation. Intuitively, K_1 simulates K_2 if their initial states are related, and every transition of K_2 corresponds to some transition of K_1.

Definition 1 (classical simulation) *[16] Let $K_1 = (\Sigma_1, s_0, A, I_1, R_1)$ and $K_2 = (\Sigma_2, t_0, A, I_2, R_2)$ be classical Kripke structures. $sim : \mathcal{P}(\Sigma_1 \times \Sigma_2) \rightarrow \mathcal{P}(\Sigma_1 \times \Sigma_2)$ is a function such that for every $\rho \subseteq \Sigma_1 \times \Sigma_2$, $(s,t) \in sim(\rho)$ iff*

1. $\forall p \in A \cdot I_2(t,p) \Rightarrow I_1(s,p)$
2. $\forall t' \in \Sigma_2 \cdot (R_2(t,t') \Rightarrow \exists s' \in \Sigma_1 \cdot (R_1(s,s') \wedge \rho(s',t')))$

ρ is a simulation *relation iff it is a fixpoint of sim. Monotonicity of sim ensures existence of the largest simulation.*

Theorem 1 *[16] The largest simulation relation is logically characterized by ECTL.*

The above result is indicated by "xx" in the first row, left column of Table 1. Clearly, simulation also preserves ECTL_{-X} and ECTL_U, as indicated in Table 1.

Divergence-blind simulation. As discussed in Section 1, we are interested in a simulation relation that relates models at different levels of abstraction. This simulation should be able to map a sequence of transitions into a single transition. We define a function *match* to capture the

	Subset of ECTL				Subset of CTL		
Simulation relations	ECTL$_U$	ECTL$_{-X}$	ECTL	Refinement relations	CTL$_U$	CTL$_{-X}$	CTL
Classical	x	x	xx	Refinement	x	x	xx
Stuttering	x	xx		Stuttering	x	xx	
Divergence-blind	xx			Divergence-blind	xx		

Table 1. Summary of the logical characterization and preservation results of refinement and simulation relations: "xx" indicates logical characterization, and "x" indicates logical preservation.

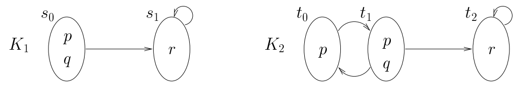

Figure 6. K_1 is related to K_2 by divergence-blind but not by stuttering simulation.

concept of mapping a single transition in one model into a finite path in the other. Given classical Kripke structures $K_1 = (\Sigma_1, s_0, A, I_1, R_1)$ and $K_2 = (\Sigma_2, t_0, A, I_2, R_2)$, states $t, t' \in \Sigma_2$ and $s \in \Sigma_1$, $\rho \subseteq \Sigma_1 \times \Sigma_2$, and $i \in \mathbb{N}$ ($\mathbb{N}$ is the set of natural numbers),

$$match_i(t, t', s, \rho) \triangleq \exists \pi \in \Pi(s) \cdot \rho(\pi_i, t') \wedge (\forall j \cdot j < i \Rightarrow \rho(\pi_j, t))$$

$match_i(t, t', s, \rho)$ holds if there exists a path of length i emanating from s and terminating in some state s', such that s' is related by ρ to t', and all other states on this path are related to t. For example, consider Kripke structures K_1 and K_2 shown in Fig. 6 and let $\rho = \{(s_0, t_0), (s_0, t_1)(s_1, t_2)\}$. $match_1(t_1, t_2, s_0, \rho)$ means that the transition (t_1, t_2) is mapped to a path of length one between s_0 (which corresponds to t_1) and s_1 (the only state which corresponds to t_2 by ρ). Similarly, $match_0(t_0, t_1, s_0, \rho)$ indicates that the transition (t_0, t_1) is mapped to a zero-length path from s_0.

Definition 2 (divergence-blind simulation) *Let $K_1 = (\Sigma_1, s_0, A, I_1, R_1)$ and $K_2 = (\Sigma_2, t_0, A, I_2, R_2)$ be classical Kripke structures. The function $db : \mathcal{P}(\Sigma_1 \times \Sigma_2) \rightarrow \mathcal{P}(\Sigma_1 \times \Sigma_2)$ is such that for every $\rho \subseteq \Sigma_1 \times \Sigma_2$, $(s, t) \in db(\rho)$ iff*

1. $\forall p \in A \cdot I_2(t, p) \Rightarrow I_1(s, p)$
2. $\forall t' \in \Sigma_2 \cdot R_2(t, t') \Rightarrow \exists i \geq 0 \cdot match_i(t, t', s, \rho)$

ρ is divergence-blind *simulation iff it is a fixpoint of db. Monotonicity of db ensures existence of the largest divergence-blind simulation.*

Let $\rho \triangleq \{(s_0, t_0), (s_0, t_1)(s_1, t_2)\}$ in the example in Fig. 6. Then ρ is divergence-blind simulation between K_1 and K_2. In this figure, unlike Fig. 5, only *true* atomic propositions are shown and all others are considered *false*, e.g. r is *false* in state s_0. Note that the set of atomic propositions that are *true* in t_0 and t_1 is a subset of the set of atomic propositions that are *true* in s_0. In this example, the loop consisting of the states t_0 and t_1 in K_2 is simulated by a single state s_0 in K_1. Thus, t_0 and t_1 are divergent states because an infinite path $t_0, t_1, t_0, t_1, \ldots$ is mapped into a single state s_0. As its name suggests, divergence-blind simulation cannot distinguish between divergent and non-divergent states.

Divergence-blind simulation ensures that negation-free existential properties with finite-length witnesses that hold in K_2 also hold in K_1. For example, $E[p\ U\ r]$ holds in both models in Fig. 6. However, this simulation does not preserve properties with infinite-length witnesses (e.g., EGp holds in K_2 and fails in K_1).

Theorem 2 *[26] A divergence-blind simulation preserves ECTL$_U$. The largest divergence-blind simulation is logically characterized by ECTL$_U$.*

The above result is indicated by "xx" in the third row, left column of Table 1.

Stuttering simulation. We now modify the definition of divergence-blind simulation to obtain a simulation that preserves a larger subset of ECTL. The result is *stuttering simulation*.

Definition 3 *Let $K_1 = (\Sigma_1, s_0, A, I_1, R_1)$ and $K_2 = (\Sigma_2, t_0, A, I_2, R_2)$ be classical Kripke structures. The function $fair : \mathcal{P}(\Sigma_1 \times \Sigma_2) \rightarrow \mathcal{P}(\Sigma_1 \times \Sigma_2)$ is such that for every $\rho \subseteq \Sigma_1 \times \Sigma_2$, $(s, t) \in fair(\rho)$ iff*

$$\forall \pi \in \Pi(t) \cdot \exists s' \in \Sigma_1 \cdot \exists t' \in \pi \cdot R_1(s, s') \wedge \rho(s', t')$$

Intuitively, a pair of states (s, t) is in *fair*(ρ) if every infinite path emanating from t has a state that is related to a successor of s via ρ. This can be seen as a fairness constraint. It states that an infinite path π in K_2 is not fair if no transition in π can be mapped to some transition in K_1. Obviously, this can happen only when *all* states in π are mapped to a single state in K_1. For example, in the model in Fig. 6, a path generated by the loop consisting of t_0 and t_1 is not fair. Extending this condition to every state of K_2, we have: K_2 is fair with respect to K_1 if for every infinite path π in K_2 and every state π_i in π, there exists a transition (π_j, π_{j+1}), for $j \geq i$, that is mapped to a transition in K_1.

We use this fairness condition to convert divergence-blind simulation to stuttering simulation. This approach is different from the one taken in [26, 7]: the latter is based on the fact that divergent states have equal labels. In our case, however, no such assumption can be made. For example, in the model in Fig. 6, states t_0 and t_1 are divergent, yet their labels are different.

Definition 4 (stuttering simulation) *Let $K_1 = (\Sigma_1, s_0, A, I_1, R_1)$ and $K_2 = (\Sigma_2, t_0, A, I_2, R_2)$ be classical Kripke structures. The function $stut : \mathcal{P}(\Sigma_1 \times \Sigma_2) \to \mathcal{P}(\Sigma_1 \times \Sigma_2)$ is such that for every $\rho \subseteq \Sigma_1 \times \Sigma_2$, $(s,t) \in stut(\rho)$ iff $(s,t) \in db(\rho) \cap fair(\rho)$. A relation ρ is* stuttering *simulation iff ρ is a fixpoint of $stut$. Monotonicity of $stut$ ensures existence of the largest stuttering simulation.*

Stuttering simulation can distinguish between divergent and non-divergent states, and in the example in Fig. 6, it does not include the pairs (s_0, t_0) and (s_0, t_1).

Theorem 3 *Stuttering simulation preserves ECTL$_{-X}$. The largest stuttering simulation is logically characterized by ECTL$_{-X}$.*

Stuttering simulation preserves ECTL$_{-X}$ and ECTL$_U$ properties, as indicated in the second row, left column of Table 1. Clearly, any simulation relation is also stuttering simulation, and any stuttering simulation is divergence-blind simulation as well. Thus, simulation relation is the smallest (i.e., the strongest) among the relations defined in this section, and requires full ECTL for its logical characterization.

3.2 Refinement Relations

Now we extend simulation relations between classical models to refinement relations between 3-valued models. A 3-valued Kripke structure K_c is a *refinement* of a 3-valued Kripke structure K_α if all definite behaviors of K_α are present in K_c, and all possible behaviors of K_c are present in K_α. Refinement can be seen as a simulation relation between $K_\alpha^{\sqsupseteq \mathsf{m}}$ and $K_c^{\sqsupseteq \mathsf{m}}$ whose inverse is a simulation between $K_c^{\sqsupseteq \mathsf{t}}$ and $K_\alpha^{\sqsupseteq \mathsf{t}}$, and is defined formally below.

Definition 5 (refinement) *[16] Let $K_c = (\Sigma_c, s_0, A, I_c, R_c)$ and $K_\alpha = (\Sigma_\alpha, a_0, A, I_\alpha, R_\alpha)$ be 3-valued Kripke structures. A relation $\rho \subseteq \Sigma_c \times \Sigma_\alpha$ is a* refinement *between K_c and K_α iff for all $(s,a) \in \Sigma_c \times \Sigma_\alpha$, if $\rho(s,a)$ then*

1. $\forall p \in A \cdot I_\alpha(a,p) \preceq I_c(s,p)$
2. $\forall a' \in \Sigma_\alpha \cdot \left(R_\alpha^{\sqsupseteq \mathsf{t}}(a,a') \Rightarrow \exists s' \in \Sigma_c \cdot \left(R_c^{\sqsupseteq \mathsf{t}}(s,s') \wedge \rho(s',a')\right)\right)$
3. $\forall s' \in \Sigma_c \cdot \left(R_c^{\sqsupseteq \mathsf{m}}(s,s') \Rightarrow \exists a' \in \Sigma_\alpha \cdot \left(R_\alpha^{\sqsupseteq \mathsf{m}}(a,a') \wedge \rho(s',a')\right)\right)$

We say that K_c refines K_α if there exists a refinement relation between them that relates their initial states.

Theorem 4 *[16] Refinement* preserves *CTL properties over 3-valued logic. The largest refinement relation is logically characterized by 3-valued CTL.*

Recall from Section 2 that a 3-valued Kripke structure K induces two classical Kripke structures $K^{\sqsupseteq \mathsf{t}}$ and $K^{\sqsupseteq \mathsf{m}}$. This allows us to lift a definition of simulation from classical to 3-valued models. Let K_α and K_c be 3-valued Kripke structures, and *sim* be a monotone function defining a simulation relation (see Definition 1). We denote an extension of *sim* to $K_c^{\sqsupseteq \mathsf{t}}$ and $K_\alpha^{\sqsupseteq \mathsf{t}}$ by $sim^{\sqsupseteq \mathsf{t}}$, and its extension to $K_\alpha^{\sqsupseteq \mathsf{m}}$ and $K_c^{\sqsupseteq \mathsf{m}}$ by $sim^{\sqsupseteq \mathsf{m}}$. According to Definition 5, a refinement relation between K_c and K_α is equal to the fixpoint of the intersection of functions $sim^{\sqsupseteq \mathsf{t}}$ and $sim^{\sqsupseteq \mathsf{m}}$. Similarly, divergence-blind refinement and stuttering refinement can be defined based on the extensions of functions db and $stut$, respectively.

Definition 6 *Let $K_c = (\Sigma_c, s_0, A, I_c, R_c)$ and $K_\alpha = (\Sigma_\alpha, a_0, A, I_\alpha, R_\alpha)$ be 3-valued Kripke structures. Let ref, dbRef, stutRef : $\mathcal{P}(\Sigma_c \times \Sigma_\alpha) \to \mathcal{P}(\Sigma_c \times \Sigma_\alpha)$ be such that for every $\rho \subseteq \Sigma_c \times \Sigma_\alpha$:*

$$(s,a) \in \mathit{ref}(\rho) \Leftrightarrow (s,a) \in sim^{\sqsupseteq \mathsf{t}}(\rho) \text{ and } (a,s) \in sim^{\sqsupseteq \mathsf{m}}(\rho^{-1})$$
$$(s,a) \in \mathit{stutRef}(\rho) \Leftrightarrow (s,a) \in stut^{\sqsupseteq \mathsf{t}}(\rho) \text{ and } (a,s) \in stut^{\sqsupseteq \mathsf{m}}(\rho^{-1})$$
$$(s,a) \in \mathit{dbRef}(\rho) \Leftrightarrow (s,a) \in db^{\sqsupseteq \mathsf{t}}(\rho) \text{ and } (a,s) \in db^{\sqsupseteq \mathsf{m}}(\rho^{-1})$$

where $\rho^{-1} \triangleq \{(s,a) \mid (a,s) \in \rho\}$. Then, ρ is refinement, divergence-blind refinement, or stuttering refinement iff it is a fixpoint of ref, dbRef, or stutRef, respectively. Moreover, monotonicity guarantees existence of the largest refinement.

Theorem 5 *(a) Refinement preserves CTL, and the largest refinement relation is logically characterized by CTL. (b) Divergence-blind refinement preserves CTL$_U$, and the largest divergence-blind refinement is logically characterized by CTL$_U$. (c) Stuttering refinement preserves CTL$_{-X}$, and the largest stuttering refinement is logically characterized by CTL$_{-X}$.*

As with simulation relations, refinement is also stuttering refinement, and stuttering refinement is divergence-blind refinement as well. These results are summarized in the right column of Table 1.

4 Building an Abstract Model

In this section, we show how to build 3-valued abstract models based on the refinement relations defined in Section 3, i.e., given a concrete model $K_c = (\Sigma_c, s_0, A, I_c, R_c)$, we construct a 3-valued abstract model $K_\alpha = (\Sigma_\alpha, a_0, A, I_\alpha, R_\alpha)$. Without loss of generality, we assume that K_c is a 2-valued (but possibly infinite-state) Kripke structure. We assume that we are given a set of abstract states Σ_α, an initial abstract state a_0, and a *left-total* relation $\rho \subseteq \Sigma_c \times \Sigma_\alpha$ that relates s_0 and a_0 and guarantees that Σ_α is a sound abstraction of Σ_c for propositional logic PL. Formally,

1. $(s_0, a_0) \in \rho$
2. $\rho(s,a) \Rightarrow \forall \varphi \in PL \cdot ||\varphi||(a) \preceq ||\varphi||(s)$

For example, consider abstracting the concrete model shown in Fig. 5(a), where the desired abstract statespace $\Sigma_\alpha = \{a_0, a_1, a_2\}$ (see, e.g., Fig. 5(c)), and the initial state of the abstract system is a_0. An abstraction relation ρ, shown in Fig. 5(b), satisfies the above conditions.

Our goal is to build the abstract transition relation R_α, so that for a given fragment L of CTL,

$$\rho(s,a) \Rightarrow \forall \varphi \in L \cdot ||\varphi||(a) \preceq ||\varphi||(s)$$

i.e., ρ is the appropriate refinement relation between K_α and K_c. For the model in Fig. 5(a) and the corresponding abstraction relation, such models are shown in Fig. 5(c)-(e).

One transition relation that satisfies the above conditions is R_α^{m}: it is completely unspecified, i.e., $\forall a, b \in \Sigma_\alpha \cdot R_\alpha^{\mathsf{m}}(a,b) = \mathsf{m}$. Furthermore, the given ρ is a refinement relation between K_c and the resulting K_α (which we refer to as K_α^{m}) for every subset of CTL, because any non-trivial CTL formula evaluates to m on K_α^{m}, and m is the lowest value according to the information ordering of $\mathbf{3}$.

We now aim to iteratively refine R_α^{m}. At each iteration, we check one transition to see whether its value can be changed from m to either t or f. Conditions for such changes for each of the refinement relations introduced in Section 3 are given below. We show that the smaller the fragment of CTL characterizing a refinement relation, the weaker are these conditions, and thus the more conclusive are the models based on this relation.

Refinement abstraction. A 3-valued model K_α is a refinement of a 2-valued model K_c iff every t-transition in K_α is simulated by a transition in K_c, and every transition in K_c is simulated by a t- or an m-transition in K_α. Thus, for any two states a, b in K_α, the transition between them is t iff every state of K_c that corresponds to a has a successor state that corresponds to b. Similarly, a transition between a and b is f iff there is no concrete state corresponding to a in K_c with a successor state that corresponds to b.

Theorem 6 *[12] ρ is a refinement between K_c and K_α if for all $a, b \in \Sigma_\alpha$,*

1. $R_\alpha(a,b) = \mathsf{f} \Leftrightarrow \neg(\exists s, s' \in \Sigma_c \cdot \rho(s,a) \wedge R_c(s,s') \wedge \rho(s',b))$
2. $R_\alpha(a,b) = \mathsf{t} \Leftrightarrow \big(\forall s \in \Sigma_c \cdot \exists s' \in \Sigma_c \cdot \rho(s,a) \Rightarrow \rho(s',b) \wedge R_c(s,s')\big)$

This theorem suggests the following procedure for constructing R_α: given a transition (a,b), attempt to prove the first condition on the right of $\Leftrightarrow$. If this condition holds, assign f to $R_\alpha(a,b)$. Otherwise, try the second condition on the right of $\Leftrightarrow$ and assign t to $R_\alpha(a,b)$ if possible. Otherwise, leave $R_\alpha(a,b)$ as m. We refer to the resulting model K_α as *refinement abstraction*. The refinement abstraction for the model in Fig. 5(a) and the corresponding abstraction relation is shown in Fig. 5(c).

There are two reasons why the value m may remain: either none of the conditions holds, or conclusive answers could not be obtained using chosen decision procedures. We choose to allow the possibility of such imprecision in our method and, for this and other refinement relations, replace $\Leftrightarrow$ by $\Rightarrow$.

Theorem 7 *ρ is a refinement between K_c and K_α if for all $a, b \in \Sigma_\alpha$,*

1. $R_\alpha(a,b) = \mathsf{f} \Rightarrow \neg(\exists s, s' \in \Sigma_c \cdot \rho(s,a) \wedge R_c(s,s') \wedge \rho(s',b))$
2. $R_\alpha(a,b) = \mathsf{t} \Rightarrow \big(\forall s \in \Sigma_c \cdot \exists s' \in \Sigma_c \cdot \rho(s,a) \Rightarrow \rho(s',b) \wedge R_c(s,s')\big)$

Divergence-blind abstraction. If K_α and K_c are related by divergence-blind refinement, a transition between states a and b in K_α is t iff each state of K_c corresponding to a can reach a state corresponding to b by a path of length zero or more, going only through states mapped to a. This condition is captured by the predicate *match*, defined in Section 3.1. Similarly, the transition must be f iff there are no concrete states corresponding to a that can reach a state corresponding to b.

Theorem 8 *ρ is divergence-blind refinement between K_c and K_α if for all $a, b \in \Sigma_\alpha$,*

1. $R_\alpha(a,b) = \mathsf{f} \Rightarrow \neg(\exists s, s' \in \Sigma_c \cdot \rho(s,a) \wedge R_c(s,s') \wedge \rho(s',b))$
2. $R_\alpha(a,b) = \mathsf{t} \Rightarrow \big(\forall s \in \Sigma_c \cdot \rho(s,a) \Rightarrow \exists i \geq 0 \cdot \mathit{match}_i(a,b,s,\rho)\big)$

Note that the condition for t abstract transitions in Theorem 8 is weaker than the one in Theorem 6, whereas the one for f is the same. Thus, a divergence-blind abstraction contains more t-transitions and the same number of f transitions as a corresponding refinement-based abstraction. Fig. 5(d) illustrates the divergence-blind abstraction of the model in Fig. 5(a). This abstraction has more t transitions than the refinement abstraction (see Fig. 5(c)), and thus enables more conclusive model-checking results. For example, properties EFq and $EF\neg q$ are t in the former model, and are m in the latter.

Theorem 9 *Let K_α^r and K_α^{db} be a refinement and a divergence-blind refinement abstraction of some classical Kripke structure K_c, respectively. Then, the identity mapping $id = \{(a,a) \mid a \in \Sigma_\alpha\}$ is divergence-blind refinement between K_α^{db} and K_α^r: $\forall \varphi \in CTL_U \cdot ||\varphi||^{K_\alpha^r}(a) \preceq ||\varphi||^{K_\alpha^{db}}(a)$.*

That is, checking a property φ in a state a of K_α^r is less conclusive than checking φ in the same state of K_α^{db}.

Note that a t-transition in the divergence-blind abstract model that is simulated by a path of length zero in the concrete model may cause an infinite path in the abstract model that is not simulated by any infinite path in the concrete one, e.g., (a_0, a_0) in the model in Fig. 5(d). Such paths are called *spurious* [5]. Spurious paths come from the *locality* of the construction of R_α: conditions for determining the value of an abstract transition $R_\alpha(a,b)$ depend only on concrete states related to a and b. Hence, when the value of a transition is being determined, we do not know whether this transition is located on an abstract infinite path that does not have a corresponding concrete infinite path. One way to avoid such problems is to move to "global", and more expensive, computations. Alternatively, we may choose to forfeit some of the precision to gain faster analysis and yet remove most of spurious paths. We discuss this approach below.

Stuttering abstraction. Stuttering refinement can be seen as divergence-blind refinement with an additional constraint that every infinite path in one model must be matched by an infinite path in the other. Thus, we need to eliminate spurious looping paths. According to Definition 4, a loop π is spurious if all the states in π can be mapped to a single concrete state that does not have any successors related to some state in π. Such spurious loops can be eliminated by adding appropriate fairness constraints. Since checking all loops in the abstract model is expensive and since we prefer to maintain the locality of the construction of the transition relation, we instead change conditions for the construction of R_α: a $\mathtt{t}$-transition in the abstract model should be mapped to a path of length *at least one* in the concrete.

Theorem 10 *ρ is stuttering refinement between K_c and K_α if for all $a, b \in \Sigma_\alpha$,*

1. $R_\alpha(a,b) = \mathtt{f} \Rightarrow \neg(\exists s, s' \in \Sigma_c \cdot \rho(s,a) \wedge R_c(s,s') \wedge \rho(s',b))$
2. $R_\alpha(a,b) = \mathtt{t} \Rightarrow \big(\forall s \in \Sigma_c \cdot \rho(s,a) \Rightarrow \exists i \geq 1 \cdot \mathit{match}_i(a,b,s,\rho)\big)$

A stuttering abstraction built according to the rules of Theorem 10 for the model in Fig. 5(a) is shown in Fig. 5(e). Note that it differs from the divergence-blind abstraction in the less definite values of (a_0, a_0) and (a_0, a_1). However, this model still contains more definite transitions than the refinement-based abstraction, and as such is more conclusive with respect to CTL_{-X}.

Theorem 11 *Let K_α^r and K_α^{stut} be a refinement and a stuttering refinement abstraction of some classical Kripke structure K_c, respectively. Then, the identity mapping $id = \{(a,a) \mid a \in \Sigma_\alpha\}$ is stuttering refinement between K_α^{stut} and K_α^r: $\forall \varphi \in CTL_{-X} \cdot ||\varphi||^{K_\alpha^r}(a) \preceq ||\varphi||^{K_\alpha^{stut}}(a)$.*

That is, checking a property φ in a state a of K_α^r is less conclusive than checking φ in the same state of K_α^{stut}. Combining this with Theorem 9, we get $\forall \varphi \in \mathrm{CTL}_U \cdot ||\varphi||^{K_\alpha^r}(a) \preceq ||\varphi||^{K_\alpha^{stut}}(a) \preceq ||\varphi||^{K_\alpha^{db}}(a)$.

5 Computing Divergence-Blind and Stuttering Abstractions

Refinement abstractions, defined in Theorems 6 and 7, can be computed automatically: given a pair of states, call a theorem prover to check whether conditions for $\mathtt{t}$- or $\mathtt{f}$-transitions are being met [14, 12]. In this section, we explore a similar approach for constructing divergence-blind and stuttering abstractions.

Computing these abstractions for a pair of abstract states (a,b), a concrete state s and a relation ρ requires computing $\exists i \cdot \mathit{match}_i(a,b,s,\rho)$, i.e., determining whether a state mapped into b is *reachable* from s while going only through states mapped to a. However, reachability is not expressible in first-order logic and thus not locally computable (see [21]) . Instead, we propose to compute *distance-bounded* reachability for a given bound k:

$$F_{n,k}^\rho(a,b,s) \triangleq \exists i \cdot n \leq i \leq k \ \wedge \mathit{match}_i(a,b,s,\rho)$$

Theorem 12 *Let n and $k \geq n$ be fixed numbers, $a, b \in \Sigma_\alpha$, $s \in \Sigma_c$, and $\rho \subseteq \Sigma_c \times \Sigma_\alpha$. Then, the k-bounded reachability formula $F_{n,k}^\rho(a,b,s) = \bigvee_{i=n}^k \Phi_i^\rho(a,b,s)$, where for $1 < i \leq k$,*

$$\begin{aligned}
\Phi_0^\rho(a,b,s) &\triangleq \rho(s,b)\\
\Phi_1^\rho(a,b,s) &\triangleq \exists s' \in \Sigma_c \cdot R_c(s,s') \wedge \Phi_0^\rho(a,b,s')\\
\Phi_i^\rho(a,b,s) &\triangleq \exists s' \in \Sigma_c \cdot R_c(s,s') \wedge \rho(s',a) \wedge \Phi_{i-1}^\rho(a,b,s')
\end{aligned}$$

Divergence-blind abstractions, requiring reachability in *zero* or more steps, are computed using $F_{0,k}^\rho(a,b,s)$, whereas stuttering abstractions, where reachability is in *one* or more steps, are computed using $F_{1,k}^\rho(a,b,s)$.

Note that in conventional abstraction frameworks (e.g., [14]), a set of concrete states corresponding to a given abstract state a can be represented as a quantifier-free formula. Further, in guarded-command programs, the set of states that have successors satisfying a predicate ψ, i.e., $\{s \mid \exists s' \in \Sigma_c \cdot R_c(s,s') \wedge s' \models \psi\}$, can be written as a quantifier-free formula as well [14, 12]. Therefore, existential quantifiers in $F_{n,k}^\rho(a,b,s)$ can be eliminated, yielding a first-order quantifier-free formula, the validity of which can be *automatically* checked by a theorem prover.

The conclusiveness of divergence-blind and stuttering abstraction depends on the bound k: the larger the k, the more likely the resulting abstraction to have more $\mathtt{t}$-transitions. One way to pick k is by putting a bound on the time alloted to proving the k-bounded reachability formula: starting at $k = 0$ or $k = 1$, increase k until either the formula becomes valid or the time runs out. For example, to prove that the value of the transition from a_1 to a_2 in the model in Fig. 1(e) is $\mathtt{t}$, we start from $k = 1$, since we are computing stuttering abstraction, and stop at $k = 2$ because in every concrete state s corresponding to a_1, either x is odd, and therefore, the immediate successor of s is related to a_2, or x is even, and the successor of its successor is related to a_2.

In some cases, the above procedure may compute reachability even for unbounded models. For example, the variable x is unbounded in the model in Fig. 2, and thus there is

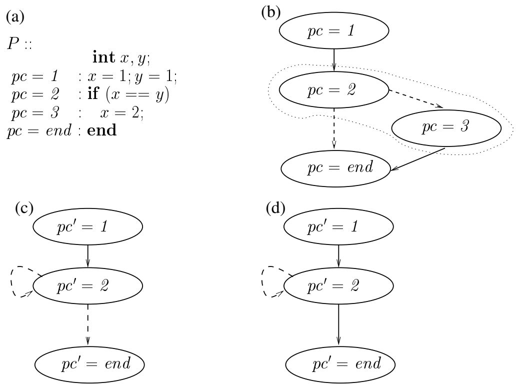

Figure 7. A program P (a) and its abstractions: a fine-grain abstraction (b); a coarse-grain abstraction (c); and a stuttering abstraction (d).

no fixed bound on the length of paths from states where $x > 0$ to those where $x \leq 0$. In order to prove that the value of the transition from a_0 to a_1 is $\mathtt{t}$, we have to compute the reachability formula for an arbitrary bound – $\bigvee_{i \in \mathbb{N}} \Phi_i^{\rho}(a, b, s)$. Note, however, that in Fig. 2(a), a state in which $x \leq i$ can reach a state where $x \leq 0$ after i steps. Furthermore, a theorem prover can easily prove the validity of the formula $\forall x \cdot x > 0 \Rightarrow \exists i \cdot x \leq i$. Therefore, the value of the transition from a_0 to a_1 in Fig. 2(c) is proven to be $\mathtt{t}$.

6 Application

We believe that stuttering abstraction can be exploited to boost the efficiency of software model-checkers [15, 2, 4]. Software model-checkers that follow the CEGAR framework [5] typically use the *control flow automaton (CFA)* to construct the initial abstraction of the concrete program. CFA is a state machine with each state corresponding to a single basic block, i.e., a single value of the program counter *pc*, of the concrete program. For example, CFA for a concrete program in Fig. 7(a) is shown in Fig. 7(b). Note that initially no information about the value of the condition $(x == y)$ is available, and thus both transitions out of the state *pc=2* are *maybe*. Many properties, such as $EF(pc = end)$, are inconclusive on this model.

Clearly, the above formula should be *true*: regardless of the value of the condition, the **if** statement has to terminate! One solution is to represent the **if** statement by a single abstract state, thus clustering blocks $(pc = 2)$ and $(pc = 3)$ (indicated by a dashed line in Fig. 7(b)). The best possible abstraction over the new abstract state-space is shown in Fig. 7(c), where $(pc' = 2)$ corresponds to states where $(pc = 2)$ or $(pc = 3)$; and $(pc' = 1)$ and $(pc' = end)$ correspond to $(pc = 1)$ and $(pc = end)$, respectively. The property $EF(pc = end)$ is still inconclusive in this model. The problem is that the transition between $(pc' = 2)$ and $(pc' = end)$ is *maybe*: not all concrete states with $(pc = 2)$ or $(pc = 3)$ can reach a state with $(pc = end)$ in a single step. This is exactly the problem addressed by stuttering abstraction, shown in Fig. 7(d), on which our property is conclusive.

As shown in the above example, the ability to change granularity of abstract states can be a useful tool for software model-checking. Not only can the resulting model be more conclusive on some properties, but also, having fewer states, it can have a noticeable impact on the efficiency of software model-checkers. Stuttering abstraction supports such changes in blocking, as long as the properties checked do not include the "next" operator.

7 Conclusion

In this paper, we explored the trade-offs between conclusiveness of an abstraction and the language of properties that can be checked using this abstraction. We have identified three simulation relations that preserve truth of properties expressed in various subsets of ECTL and showed how to extend these to refinement relations that preserve respective subsets of CTL. The framework for turning simulation into refinement is general and can be applied to other simulations as well.

We also showed how to construct abstract models based on these refinements, yielding models that are more conclusive, for identified subsets of CTL, than those built using the standard 3-valued abstraction [12]. Further, we studied the efficient computation of such abstraction models and illustrated it on a few examples. For a larger case-study (of an alternating bit protocol), please refer to [25]. The model-construction approach is general as well, and can be applied to arbitrary abstraction frameworks that are defined using a *left-total* abstraction relation with suitable properties. Some examples of such frameworks are predicate [14] and Cartesian [2] abstractions.

In [27, 11, 9, 10, 20], an abstraction refinement framework is proposed where $\mathtt{t}$-transitions are treated as transitions into sets of states, and referred to as *focused* or *disjunctive* transitions. When compared to traditional approaches, this treatment improves the conclusiveness of the abstraction. For example, the problem in Fig. 1 is solved by adding new predicates, as shown in Fig. 1(c). Focused transitions require the abstract domain to include all sets of abstract states. This can make the abstract domain very large, and hence, model-checking becomes infeasible. In contrast, our approach regains additional precision without enlarging the abstract domain and is easier to apply in practice. More detailed comparisons between these two are left for future work.

In the future, we plan to implement our framework and study the effectiveness of our abstraction relations on realistic examples. In particular, we need to address the problem of the efficient use of a theorem prover for computing abstractions. Also, we aim to investigate the relationship between succinctness and conclusiveness of abstract models.

Acknowledgment. We thank Mehrdad Sabetzadeh and Mihaela Gheorghiu for their comments on earlier versions of this paper. Financial support for this research has been provided by NSERC and CITO.

References

[1] M. Abadi and L. Lamport. "The Existence of Refinement Mappings". *Theor. Computer Science*, 82(2):253–284, 1991.

[2] T. Ball, A. Podelski, and S. Rajamani. "Boolean and Cartesian Abstraction for Model Checking C Programs". In *TACAS*, vol. 2031 of *LNCS*, 2001.

[3] M. C. Browne, E. M. Clarke, and O. Grumberg. "Characterizing Finite Kripke Structures in Propositional Temporal Logic". *Theor. Computer Science*, 59(1-2):115–131, 1988.

[4] S. Chaki, E. Clarke, A. Groce, S. Jha, and H. Veith. "Modular Verification of Software Components in C". In *ICSE*, pp. 385–395, 2003.

[5] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. "Counterexample-Guided Abstraction Refinement". In *CAV*, vol. 1855 of *LNCS*, pp. 154–169, 2000.

[6] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[7] D. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. PhD thesis, Eindhoven University of Technology, July 1996.

[8] D. Dams, R. Gerth, and O. Grumberg. "Abstract Interpretation of Reactive Systems". *ACM Transactions on Programming Languages and Systems*, 2(19):253–291, 1997.

[9] D. Dams and K. Namjoshi. "The Existence of Finite Abstractions for Branching Time Model Checking". In *LICS*, 2004.

[10] D. Dams and K. Namjoshi. "Automata as Abstractions". In *VMCAI*, 2005.

[11] L. de Alfaro, P. Godefroid, and R. Jagadeesan. "Three-Valued Abstractions of Games: Uncertainty, but with Precision". In *LICS*, 2004.

[12] P. Godefroid, M. Huth, and R. Jagadeesan. "Abstraction-based Model Checking using Modal Transition Systems". In *CONCUR*, vol. 2154 of *LNCS*, 2001.

[13] P. Godefroid and R. Jagadeesan. "On the Expressiveness of 3-Valued Models". In *VMCAI*, vol. 2575 of *LNCS*, pp. 206–222, 2003.

[14] S. Graf and H. Saïdi. "Construction of Abstract State Graphs with PVS". In *CAV*, vol. 1254 of *LNCS*, 1997.

[15] T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. "Lazy abstraction". In *POPL*, pp. 58–70, 2002.

[16] M. Huth, R. Jagadeesan, and D. A. Schmidt. "Modal Transition Systems: A Foundation for Three-Valued Program Analysis". In *ESOP*, vol. 2028 of *LNCS*, 2001.

[17] S. C. Kleene. *Introduction to Metamathematics*. New York: Van Nostrand, 1952.

[18] L. Lamport. "What Good is Temporal Logic?". In *IFIP 9th World Computer Congress*, vol. 83 of *Information Processing*, pp. 657–668, 1983.

[19] K.G. Larsen and B. Thomsen. "A Modal Process Logic". In *LICS*, 1988.

[20] K.G. Larsen and L. Xinxin. "Equation solving using modal transition systems". In *LICS*, 1990.

[21] L. Libkin. *Elements of Finite Model Theory*. Springer-Verlag, 2003.

[22] P. Manolios. *"Mechanical Verification of Reactive Systems"*. PhD thesis, University of Texas, Austin, 2001.

[23] R. Milner. "An Algebraic Definition of Simulation between Programs". In *2nd Inter. Joint Conf. on AI*, pp. 481–489, 1971.

[24] S. Nejati. "Refinement Relations on Partial Specifications". Master's thesis, Univ. of Toronto, July 2003.

[25] S. Nejati. "Formal Analysis of the Alternating Bit Protocol Using Stuttering Refinement". CSRG Tech. Report 487, Univ. of Toronto, 2004.

[26] R. De Nicola and F. Vaandrager. "Three Logics for Branching Bisimulation". *J. of the ACM*, 42(2):458–487, 1995.

[27] S. Shoham and O. Grumberg. "Monotonic Abstraction-Refinement for CTL". In *TACAS*, vol. 2988 of *LNCS*, 2004.

A Selected Theorems

In this appendix, we provide a proof for Theorem 10. A proof of Theorem 8 is similar. Proofs of Theorems 9 and 11 follow from the construction of refinement, divergence-blind, and stuttering abstractions. For proofs of Theorems 3 and 5 please refer to [24]. A proof of Theorem 12 follows trivially by induction on i.

Theorem 10. ρ is stuttering refinement between K_c and K_α if for all $a, b \in \Sigma_\alpha$,

1. $R_\alpha(a,b) = \mathsf{f} \Rightarrow \neg(\exists s, s' \in \Sigma_c \cdot \rho(s,a) \wedge R_c(s,s') \wedge \rho(s',b))$
2. $R_\alpha(a,b) = \mathsf{t} \Rightarrow \big(\forall s \in \Sigma_c \cdot \rho(s,a) \Rightarrow \exists i \geq 1 \cdot match_i(a,b,s,\rho)\big)$

Proof:
Let ρ be a relation between K_c and K_α. By Def. 6, ρ is stuttering refinement between K_c and K_α iff $\rho = stutRef(\rho)$. Since $stutRef$ is monotonically decreasing, $stutRef(\rho) \subseteq \rho$. Thus, our goal is to show $\rho \subseteq stutRef(\rho)$.

Let $s \in \Sigma_c$, $a \in \Sigma_\alpha$. We prove $\rho(s,a) \Rightarrow (s,a) \in stutRef(\rho)$.

By Def. 6, we prove:

I. $\rho(s,a) \Rightarrow (s,a) \in stut^{\sqsupseteq \mathsf{t}}(\rho)$, and

II. $\rho(s,a) \Rightarrow (a,s) \in stut^{\sqsupseteq \mathsf{m}}(\rho^{-1})$.

Assumptions:
A.1: $\rho(s,a) \Rightarrow \forall \varphi \in PL \cdot ||\varphi||(a) \preceq ||\varphi||(s)$ (see page 6)
A.2: Since K_c is a 2-valued Kripke structure, $K_c^{\sqsupseteq \mathsf{t}} = K_c^{\sqsupseteq \mathsf{m}} = K_c$
A.3: $R_\alpha(a,b) = \mathsf{f} \Rightarrow \neg(\exists s, s' \in \Sigma_c \cdot \rho(s,a) \wedge R_c(s,s') \wedge \rho(s',b))$
A.4: $R_\alpha(a,b) = \mathsf{t} \Rightarrow (\forall s \in \Sigma_c \cdot \rho(s,a) \Rightarrow \exists i \geq 1 \cdot match_i(a,b,s,\rho))$

Below we only give a proof of **I.**. Proof of **II** is similar.

By Def. 4, we need to prove:

(a) $\rho(s,a) \Rightarrow \forall p \in A \cdot I_\alpha^{\sqsupseteq \mathsf{t}}(a,p) \Rightarrow I_c^{\sqsupseteq \mathsf{t}}(s,p)$:

$\rho(s,a)$
$\Rightarrow$ by **A.1**
$\forall \varphi \in PL \cdot ||\varphi||(a) \preceq ||\varphi||(s)$
$\Rightarrow$ by properties of Kleene $\preceq$
$\forall \varphi \in PL \cdot ||\varphi||(a) \sqsupseteq \mathsf{t} \Rightarrow ||\varphi||(s) \sqsupseteq \mathsf{t}$
$\Rightarrow$ by renaming and since $A \subset PL$
$\forall p \in PL \cdot ||p||(a) \sqsupseteq \mathsf{t} \Rightarrow ||p||(s) \sqsupseteq \mathsf{t}$
$\Rightarrow$ by semantics of CTL, using structures K_α and K_c
$\forall p \in A \cdot I_\alpha(p,a) \sqsupseteq \mathsf{t} \Rightarrow I_c(p,s) \sqsupseteq \mathsf{t}$
$\Rightarrow$ notation change (see page 3)
$\forall p \in A \cdot I_\alpha^{\sqsupseteq \mathsf{t}}(a,p) \Rightarrow I_c^{\sqsupseteq \mathsf{t}}(s,p)$

(b) $\rho(s,a) \Rightarrow$
$\forall b \in \Sigma_\alpha \cdot R_\alpha^{\sqsupseteq \mathsf{t}}(a,b) \Rightarrow \exists i \geq 0 \cdot match_i^{\sqsupseteq \mathsf{t}}(a,b,s,\rho)$:

Let $b \in \Sigma_\alpha$:

$\rho(s,a) \wedge R_\alpha^{\sqsupseteq \mathsf{t}}(a,b)$
$\Rightarrow$ notation change (see page 3)
$\rho(s,a) \wedge R_\alpha(a,b) \sqsupseteq \mathsf{t}$
$\Rightarrow$ by **A.4**
$\rho(s,a) \wedge \big(\forall s' \in \Sigma_c \cdot \rho(s',a) \Rightarrow \exists i \geq 1 \cdot match_i(a,b,s',\rho)\big)$
$\Rightarrow$ by **A.2**
$\rho(s,a) \wedge \big(\forall s' \in \Sigma_c \cdot \rho(s',a) \Rightarrow \exists i \geq 1 \cdot match_i^{\sqsupseteq \mathsf{t}}(a,b,s',\rho)\big)$
$\Rightarrow$ by specialization, modus ponens and domain expanding
$\exists i \geq 0 \cdot match_i^{\sqsupseteq \mathsf{t}}(a,b,s,\rho)$

(c) $\rho(s,a) \Rightarrow (s,a) \in fair^{\sqsupseteq \mathsf{t}}(\rho)$:

Let $\pi_*^{\sqsupseteq \mathsf{t}} \in \Pi(a)$:

$\rho(s,a)$
$\Rightarrow$ by definition of path (see page 3)
$\rho(s,a) \wedge \exists b \in \pi_*^{\sqsupseteq \mathsf{t}} \cdot R_\alpha(a,b) \sqsupseteq \mathsf{t}$
$\Rightarrow$ by **A.4** and specialization
$\exists i \geq 1 \cdot match_i(a,b,s,\rho)$
$\Rightarrow$ by definition of *match*
$\exists i \geq 1 \cdot \exists \pi \in \Pi(s) \cdot \rho(\pi_i, b) \wedge$
$\forall j \cdot j < i \Rightarrow \rho(\pi_j, a)$
$\Rightarrow$ by domain splitting
$\exists i \geq 1 \cdot \exists \pi \in \Pi(s) \cdot \rho(\pi_i, b) \wedge$
$\forall j \cdot 0 < j < i \Rightarrow \rho(\pi_j, a)$
$\Rightarrow$ by $\exists$-elimination and domain splitting
$\exists \pi \in \Pi(s) \cdot \big(i_0 > 1 \Rightarrow (\rho(\pi_{i_0}, b) \wedge$
$\forall j \cdot 1 < j < i_0 \Rightarrow \rho(\pi_j, a) \wedge \rho(\pi_1, a))\big) \bigvee$
$\big(i_0 = 1 \Rightarrow \rho(\pi_1, b)\big)$
$\Rightarrow$ by letting $s' = \pi_1$ and weakening
$\exists \pi \in \Pi(s) \cdot \rho(s',a) \vee \rho(s',b)$
$\Rightarrow$ by definition of path (see page 3) and since $s' = \pi_1$
$R_c(s,s') \wedge (\rho(s',a) \vee \rho(s',b))$
$\Rightarrow$ by $\exists$-introduction and since $a, b \in \pi_*^{\sqsupseteq \mathsf{t}}$
$\exists a' \in \pi_*^{\sqsupseteq \mathsf{t}} \cdot R_c(s,s') \wedge \rho(s',a')$
$\Rightarrow$ by $\exists$-introduction and since $s' \in \Sigma_c$

$\exists s'' \in \Sigma_c \cdot \exists a' \in \pi_*^{\sqsupseteq \mathsf{t}} \cdot R_c(s,s'') \wedge \rho(s'',a')$
$\Rightarrow$ by Def. 3 and since $\pi_*^{\sqsupseteq \mathsf{t}} \in \Pi(a)$ is an arbitrary path
$(s,a) \in fair^{\sqsupseteq \mathsf{t}}(\rho)$

□