A divide & conquer approach to until and until stable model checking

Canh Minh Do, Yati Phyo, and Kazuhiro Ogata
School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)
Nomi, Ishikawa 923-1211 Japan
{canhdominh,yatiphyo,ogata}@jaist.ac.jp

***Abstract*—The paper describes a technique to mitigate the notorious state space explosion in model checking. The technique is called a divide & conquer approach to until and until stable model checking. As indicated by the name, the technique is dedicated to until and until stable properties that are expressed as $\varphi_1 \mathcal{U} \varphi_2$ and $\varphi_1 \mathcal{U} \Box\varphi_2$, respectively, where φ_1, φ_2 are state propositions. For real-time system analysis, some interesting systems requirements are expressed as until and until stable properties. For example, a clock is running and shows a correct time until a certain time has passed or until the clock stops due to an empty battery or other failures. Therefore, it is worth focusing on the properties. For each property, we prove a theorem that the proposed technique is correct and design an algorithm based on the theorem to support the technique.**

***Index Terms*—until properties, until stable properties, linear temporal logic (LTL), model checking.**

I. Introduction

Model checking [1] is an automatic verification technique for verifying finite-state hardware and software systems. It has proven to be a tremendously successful technique to verify requirements for a variety of systems. However, there are still some challenges to tackle in model checking, one of them is the state space explosion, the most annoying one. Many techniques are devised to mitigate the problem, such as ordered binary decision diagrams [2], partial order reduction [3], and abstraction [4]–[6]. Such techniques mitigate the problem to some extent, but the problem still remains and often prevents some model checking experiments from being carried out.

To address the problem, our research group came up with a divide & conquer approach to model checking leads-to properties [7] expressed as $\varphi_1 \rightsquigarrow \varphi_2$, conditional stable properties [8] expressed as $\varphi_1 \rightsquigarrow \Box\varphi_2$, and eventual (or eventually) properties [9] expressed as $\Diamond\varphi$, where $\varphi, \varphi_1, \varphi_2$ are state propositions. Although leads-to properties, conditional stable properties, and eventual properties can be expressed in LTL, it is necessary to individually prove the correctness of each of the three divide & conquer approaches to leads-to, eventual, and conditional stable model checking, come up with each algorithm. This is because it is not straightforward to uniformly deal with the three different properties so as to mitigate the state space explosion. Likewise, it is not straightforward to deal with until and until stable properties that are expressed $\varphi_1 \mathcal{U} \varphi_2$ and $\varphi_1 \mathcal{U} \Box\varphi_2$, respectively, where φ_1, φ_2 are state propositions. Hence, it is meaningful to prove the correctness of the divide & conquer approach to until and until stable model checking and design each algorithm for each property so as to mitigate the state space explosion.

Real-Time Maude (RT-Maude) [10] is a language and tool supporting the formal specification and analysis of real-time and hybrid systems where time information is taken into account. RT-Maude is implemented in Maude [11] as an extension of Full Maude [12]. RT-Maude specifications are executable and simulate the progress in systems under analysis by *timed rewriting*. *Tick rules* are used to formalize the time advance in systems in which the time increment is given in the form of either a concrete value or a variable. The former is called *time-deterministic* and is used in discrete time domains, while the latter is called *time-nondeterministic* because the time increment is an arbitrary number that will be decided based on the *time sampling strategy* (time mode) specified by users among some time sampling strategies supported by RT-Maude, and is used in dense time domains. RT-Maude supports time-bounded linear temporal logic model checking that can analyze all behaviors of a system from a given initial state up to a certain duration. By restricting model checking up to a certain duration, the set of reachable states is a finite set so that model checking experiments can be carried out. RT-Maude usually uses some properties specified in LTL to express systems requirements among which until and until stable properties are used [10], [13]. Besides, RT-Maude supports dedicated commands to check for until and until stable properties, meaning that until and until stable properties are interesting properties in real-time systems. Furthermore, the reachable state space of a real-time system is often huge because the behavior of the system changes over time. Therefore, it is worth focusing on mitigating the state space explosion in until and until stable model checking.

This paper describes an ongoing work that extends the divide & conquer approach so as to handle until and until stable properties expressed as $\varphi_1 \mathcal{U} \varphi_2$ and $\varphi_1 \mathcal{U} \Box\varphi_2$, respectively, where φ_1, φ_2 are state propositions. Until properties informally say that the first argument is true *until* its second argument is true, which is required to happen. Until stable properties informally say that the first argument is true *until*

This research was partially supported by JSPS KAKENHI Grant Number JP19H04082.
DOI reference number: 10.18293/SEKE2022-058

its second argument is true and continues to be true (*stable*) subsequently, which is required to happen. We can see that if until stable properties hold, then until properties also hold. In this paper, for each property, we prove a theorem that the proposed technique is correct and design an algorithm based on the theorem to support the technique. A basic idea of the technique is that the reachable state space from each initial state is split into multiple layers, generating multiple sub-state spaces, and conducting model checking experiments for each sub-state space. If the size of each sub-state space is much smaller than the one of the original reachable state space, it is feasible to conduct model checking experiments with the approach even though it is infeasible to do so for the original reachable state space due to the state space explosion. That is the key to mitigating the state space explosion in model checking with the technique.

The rest of the paper is organized as follows. Sect. II mentions some preliminaries. Sect. III proves a theorem for the divide & conquer approach to until model checking. Sect. IV describes an algorithm that is constructed based on the theorem. Sect. V proves a theorem for the divide & conquer approach to until stable model checking. Sect. VI describes an algorithm that is constructed based on the theorem. Sect. VII mentions some existing related work. Sect. VIII finally concludes the paper together with some future directions.

II. Preliminaries

Definition 1 (Kripke structures)**.** *A Kripke structure* $\boldsymbol{K} \triangleq \langle \boldsymbol{S}, \boldsymbol{I}, \boldsymbol{T}, \boldsymbol{A}, \boldsymbol{L} \rangle$ *consists of a set* $\boldsymbol{S}$ *of states, a set* $\boldsymbol{I} \subseteq \boldsymbol{S}$ *of initial states, a left-total binary relation* $\boldsymbol{T} \subseteq \boldsymbol{S} \times \boldsymbol{S}$ *over states, a set* $\boldsymbol{A}$ *of atomic propositions and a labeling function* $\boldsymbol{L}$ *whose type is* $\boldsymbol{S} \to 2^{\boldsymbol{A}}$*. An element* $(s, s') \in \boldsymbol{T}$ *is called a (state) transition from* s *to* s' *and may be written as* $s \to_{\boldsymbol{K}} s'$*.*

An infinite sequence $s_0, s_1, \ldots, s_i, s_{i+1}, \ldots$ of states is called a path of $\boldsymbol{K}$ iff for any natural number i, $(s_i, s_{i+1}) \in \boldsymbol{T}$. Let π be $s_0, s_1, \ldots, s_i, s_{i+1}, \ldots$ and some notations on π are defined as follows: $\pi(i)$ is s_i; π^i is $s_i, s_{i+1}, \ldots$; π_i is $s_0, s_1, \ldots, s_i, s_i, s_i, \ldots$; $\pi^{(i,j)}$ is $s_i, s_{i+1}, \ldots, s_j, s_j, s_j, \ldots$ if $i \leq j$ and $s_i, s_i, s_i, \ldots$ otherwise; $\pi^{(i,\infty)}$ is π^i, where i, j are natural numbers. A path π of $\boldsymbol{K}$ is called a computation of $\boldsymbol{K}$ iff $\pi(0) \in \boldsymbol{I}$. Let $\boldsymbol{P}_{\boldsymbol{K}}$ be the set of all paths of $\boldsymbol{K}$. Let $\boldsymbol{P}_{(\boldsymbol{K},s)}$ be $\{\pi \mid \pi \in \boldsymbol{P}_{\boldsymbol{K}}, \pi(0) = s\}$, where $s \in \boldsymbol{S}$. Let $\boldsymbol{P}^b_{(\boldsymbol{K},s)}$ be $\{\pi_b \mid \pi \in \boldsymbol{P}_{(\boldsymbol{K},s)}\}$, where $s \in \boldsymbol{S}$ and b is a natural number. Note that $\boldsymbol{P}^{\infty}_{(\boldsymbol{K},s)}$ is $\boldsymbol{P}_{(\boldsymbol{K},s)}$.

Definition 2 (Syntax of LTL)**.** *The syntax of linear temporal logic (LTL) is as follows:* $\varphi ::= a \mid \top \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi \,\mathcal{U}\, \varphi$*, where* $a \in \boldsymbol{A}$*.*

Definition 3 (Semantics of LTL)**.** *For any Kripke structure* $\boldsymbol{K}$*, any path* π *of* $\boldsymbol{K}$ *and any LTL formula* φ*,* $\boldsymbol{K}, \pi \models \varphi$ *is inductively defined as follows:*

- $\boldsymbol{K}, \pi \models a$ *iff* $a \in L(\pi(0))$
- $\boldsymbol{K}, \pi \models \top$
- $\boldsymbol{K}, \pi \models \neg\varphi_1$ *iff* $\boldsymbol{K}, \pi \not\models \varphi_1$
- $\boldsymbol{K}, \pi \models \varphi_1 \vee \varphi_2$ *iff* $\boldsymbol{K}, \pi \models \varphi_1$ *and/or* $\boldsymbol{K}, \pi \models \varphi_2$
- $\boldsymbol{K}, \pi \models \bigcirc \varphi_1$ *iff* $\boldsymbol{K}, \pi^1 \models \varphi_1$
- $\boldsymbol{K}, \pi \models \varphi_1 \,\mathcal{U}\, \varphi_2$ *iff there exists a natural number* i *such that* $\boldsymbol{K}, \pi^i \models \varphi_2$ *and for each natural number* $j < i$*,* $\boldsymbol{K}, \pi^j \models \varphi_1$

where φ_1 *and* φ_2 *are LTL formulas. Then,* $\boldsymbol{K} \models \varphi$ *iff* $\boldsymbol{K}, \pi \models \varphi$ *for all computations* π *of* $\boldsymbol{K}$*. Let* $True$ *denote* $\boldsymbol{K}, \pi \models \top$*, which always holds.*

$\bot \triangleq \neg\top$ and some other connectives are defined as follows: $\varphi_1 \wedge \varphi_2 \triangleq \neg((\neg\varphi_1) \vee (\neg\varphi_2))$, $\varphi_1 \Rightarrow \varphi_2 \triangleq (\neg\varphi_1) \vee \varphi_2$, $\varphi_1 \Leftrightarrow \varphi_2 \triangleq (\varphi_1 \Rightarrow \varphi_2) \wedge (\varphi_2 \Rightarrow \varphi_1)$, $\Diamond\varphi_1 \triangleq \top \,\mathcal{U}\, \varphi_1$, $\Box\varphi_1 \triangleq \neg(\Diamond\neg\varphi_1)$ and $\varphi_1 \rightsquigarrow \varphi_2 \triangleq \Box(\varphi_1 \Rightarrow \Diamond\varphi_2)$. $\bigcirc$, $\mathcal{U}$, $\Diamond$, $\Box$ and $\rightsquigarrow$ are called next, until, eventually, always and leads-to temporal connectives, respectively. State propositions are LTL formulas such that they do not have any temporal connectives. Until stable properties can be expressed as $\varphi_1 \,\mathcal{U}\, \Box\varphi_2$, where φ_1, φ_2 are state propositions. Although it is unnecessary to define the semantics for $\varphi_1 \,\mathcal{U}\, \Box\varphi_2$, we define it as follows:

- $\boldsymbol{K}, \pi \models \varphi_1 \,\mathcal{U}\, \Box\varphi_2$ iff there exists a natural number i such that $\boldsymbol{K}, \pi^i \models \Box\varphi_2$ and for each natural number $j < i$, $\boldsymbol{K}, \pi^j \models \varphi_1$.

III. Multiple Layer Division of Until Model Checking

Proposition 1. *Let* $\boldsymbol{K}$ *be any Kripke structure. If* φ *is any state proposition, then* $(\boldsymbol{K}, \pi \models \varphi) \Leftrightarrow (\boldsymbol{K}, \pi' \models \varphi)$ *for any paths* π *&* π' *of* $\boldsymbol{K}$ *such that* $\pi(0) = \pi'(0)$*.*

Proof. The first state $\pi(0)$ decides if $\boldsymbol{K}, \pi \models \varphi$ holds. □

Lemma 1. *Let* φ_1, φ_2 *be any state propositions of* $\boldsymbol{K}$*. Let* k *be any natural number. Then,* $(\boldsymbol{K}, \pi_k \models \varphi_1 \,\mathcal{U}\, \varphi_2) \Rightarrow (\boldsymbol{K}, \pi \models \varphi_1 \,\mathcal{U}\, \varphi_2)$*.*

Proof. If $\boldsymbol{K}, \pi_k \models \varphi_1 \,\mathcal{U}\, \varphi_2$, then there exists $i \leq k$ such that $\boldsymbol{K}, \pi^i_k \models \varphi_2$, which is equivalent to $\boldsymbol{K}, \pi^i \models \varphi_2$ from Proposition 1, and for each $j < i$, $\boldsymbol{K}, \pi^j_k \models \varphi_1$, which is equivalent to $\boldsymbol{K}, \pi^j \models \varphi_1$ from Proposition 1. Hence, $\boldsymbol{K}, \pi \models \varphi_1 \,\mathcal{U}\, \varphi_2$. □

Lemma 2. *Let* φ_1, φ_2 *be any state propositions of* $\boldsymbol{K}$*. Let* k *be any natural number. Then,* $(\boldsymbol{K}, \pi_k \models \Box\varphi_1) \wedge (\boldsymbol{K}, \pi^k \models \varphi_1 \,\mathcal{U}\, \varphi_2) \Rightarrow (\boldsymbol{K}, \pi \models \varphi_1 \,\mathcal{U}\, \varphi_2)$*.*

Proof. If $\boldsymbol{K}, \pi_k \models \Box\varphi_1$, then for each $i' \leq k$, $\boldsymbol{K}, \pi^{i'}_k \models \varphi_1$, which is equivalent to $\boldsymbol{K}, \pi^{i'} \models \varphi_1$ from Proposition 1 (1). If $\boldsymbol{K}, \pi^k \models \varphi_1 \,\mathcal{U}\, \varphi_2$ then there exists $i \geq k$ such that $\boldsymbol{K}, \pi^i \models \varphi_2$ and for each j such that $k \leq j < i$, $\boldsymbol{K}, \pi^j \models \varphi_1$ (2). From (1) and (2), we have $\boldsymbol{K}, \pi \models \varphi_1 \,\mathcal{U}\, \varphi_2$. □

Lemma 3 (Two layer division of $\varphi_1 \,\mathcal{U}\, \varphi_2$)**.** *Let* φ_1, φ_2 *be any state propositions of* $\boldsymbol{K}$*. Let* k *be any natural number. Then,*

$$
\begin{aligned}
&(\boldsymbol{K}, \pi \models \varphi_1 \,\mathcal{U}\, \varphi_2) \\
\Leftrightarrow\ &[(\boldsymbol{K}, \pi_k \models \varphi_1 \,\mathcal{U}\, \varphi_2) \Rightarrow True] \wedge \\
&[(\boldsymbol{K}, \pi_k \not\models \varphi_1 \,\mathcal{U}\, \varphi_2) \Rightarrow (\boldsymbol{K}, \pi_k \models \Box\varphi_1) \wedge \\
&\qquad\qquad (\boldsymbol{K}, \pi^k \models \varphi_1 \,\mathcal{U}\, \varphi_2)]
\end{aligned}
$$

Proof. (1) Case "only if" (⇒): The case is split into two cases: (1.1) $\boldsymbol{K}, \pi_k \models \varphi_1 \,\mathcal{U}\, \varphi_2$ and (1.2) $\boldsymbol{K}, \pi_k \not\models \varphi_1 \,\mathcal{U}\, \varphi_2$. In

(1.1), it is obvious. In (1.2), from the assumption, there exists i such that $\boldsymbol{K}, \pi^i \models \varphi_2$ and for each $j < i$, $\boldsymbol{K}, \pi^j \models \varphi_1$. Because $\boldsymbol{K}, \pi_k \not\models \varphi_1 \mathcal{U} \varphi_2$, then $k < i$. Hence, $(\boldsymbol{K}, \pi_k \models \Box\varphi_1) \wedge (\boldsymbol{K}, \pi^k \models \varphi_1 \mathcal{U} \varphi_2)$.

(2) Case "if" ($\Leftarrow$): The case is split into two cases: (2.1) $\boldsymbol{K}, \pi_k \models \varphi_1 \mathcal{U} \varphi_2$ and (2.2) $\boldsymbol{K}, \pi_k \not\models \varphi_1 \mathcal{U} \varphi_2$. In (2.1), $\boldsymbol{K}, \pi \models \varphi_1 \mathcal{U} \varphi_2$ from Lemma 1. In (2.2), $\boldsymbol{K}, \pi \models \varphi_1 \mathcal{U} \varphi_2$ from Lemma 2. □

Definition 4 (Until_L)**.** *Let L be any non-zero natural number, k be any natural number and d be any function such that $d(0)$ is 0, $d(x)$ is a natural number for $x = 1, \ldots, L$ and $d(L+1)$ is ∞.*

1) $0 \leq k < L - 1$

$$\begin{aligned}&\mathrm{Until}_L(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, k)\\ \triangleq\ &[(\boldsymbol{K}, \pi^{(d(k),d(k+1))} \models \varphi_1 \mathcal{U} \varphi_2) \Rightarrow \mathit{True}] \wedge\\ &[(\boldsymbol{K}, \pi^{(d(k),d(k+1))} \not\models \varphi_1 \mathcal{U} \varphi_2)\\ &\quad \Rightarrow (\boldsymbol{K}, \pi^{(d(k),d(k+1))} \models \Box\varphi_1) \wedge\\ &\quad\ \mathrm{Until}_L(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, k+1)]\end{aligned}$$

2) $k = L - 1$

$$\begin{aligned}&\mathrm{Until}_L(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, k)\\ \triangleq\ &[(\boldsymbol{K}, \pi^{(d(k),d(k+1))} \models \varphi_1 \mathcal{U} \varphi_2) \Rightarrow \mathit{True}] \wedge\\ &[(\boldsymbol{K}, \pi^{(d(k),d(k+1))} \not\models \varphi_1 \mathcal{U} \varphi_2)\\ &\quad \Rightarrow (\boldsymbol{K}, \pi^{(d(k),d(k+1))} \models \Box\varphi_1) \wedge\\ &\quad\ (\boldsymbol{K}, \pi^{(d(k+1),d(k+2))} \models \varphi_1 \mathcal{U} \varphi_2)]\end{aligned}$$

Theorem 1 ($L+1$ layer division of $\varphi_1 \mathcal{U} \varphi_2$)**.** *Let L be any non-zero natural number. Let $d(0)$ be 0, $d(x)$ be any natural number for $x = 1, \ldots, L$ and $d(L+1)$ be ∞. Let φ_1, φ_2 be any state propositions of $\boldsymbol{K}$. Then,*

$$(\boldsymbol{K}, \pi \models \varphi_1 \mathcal{U} \varphi_2) \Leftrightarrow \mathrm{Until}_L(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, 0)$$

Proof. By induction on L.

- Base case ($L = 1$): It follows from Lemma 3.
- Induction case ($L = l + 1$): We prove the following:

$$(\boldsymbol{K}, \pi \models \varphi_1 \mathcal{U} \varphi_2) \Leftrightarrow \mathrm{Until}_{l+1}(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, 0)$$

Let d_{l+1} be d used in $\mathrm{Until}_{l+1}(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, 0)$ such that $d_{l+1}(0) = 0$, $d_{l+1}(i)$ is an arbitrary natural number for $i = 1, \ldots, l+1$ and $d_{l+1}(l+2) = \infty$. The induction hypothesis is as follows:

$$(\boldsymbol{K}, \pi \models \varphi_1 \mathcal{U} \varphi_2) \Leftrightarrow \mathrm{Until}_l(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, 0)$$

Let d_l be d used in $\mathrm{Until}_l(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, 0)$ such that $d_l(0) = 0$, $d_l(i)$ is an arbitrary natural number for $i = 1, \ldots, l$ and $d_l(l+1) = \infty$. Because $d_{l+1}(i)$ is an arbitrary natural number for $i = 1, \ldots, l+1$, we suppose that $d_{l+1}(1) = d_l(1)$ and $d_{l+1}(i+1) = d_l(i)$ for $i = 1, \ldots, l$. Because π is any path of $\boldsymbol{K}$, π can be replaced with $\pi^{d_l(1)}$. If so, we have the following as an instance of the induction hypothesis:

$$(\boldsymbol{K}, \pi^{d_l(1)} \models \varphi_1 \mathcal{U} \varphi_2) \Leftrightarrow \mathrm{Until}_l(\boldsymbol{K}, \pi^{d_l(1)}, \varphi_1, \varphi_2, 0)$$

From Definition 4, $\mathrm{Until}_l(\boldsymbol{K}, \pi^{d_l(1)}, \varphi_1, \varphi_2, 0)$ is $\mathrm{Until}_{l+1}(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, 1)$ because $d_l(0) = d_{l+1}(0) = 0$, $d_l(1) = d_{l+1}(1)$ and $d_l(i) = d_{l+1}(i+1)$ for $i = 1, \ldots, l$ and $d_l(l+1) = d_{l+1}(l+2) = \infty$. Therefore, the induction hypothesis instance can be rephrased as follows:

$$(\boldsymbol{K}, \pi^{d_{l+1}(1)} \models \varphi_1 \mathcal{U} \varphi_2) \Leftrightarrow \mathrm{Until}_{l+1}(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, 1)$$

From Definition 4, $\mathrm{Until}_{l+1}(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, 0)$ is

$$\begin{aligned}&[(\boldsymbol{K}, \pi^{(d_{l+1}(0),d_{l+1}(1))} \models \varphi_1 \mathcal{U} \varphi_2) \Rightarrow \mathit{True}] \wedge\\ &[(\boldsymbol{K}, \pi^{(d_{l+1}(0),d_{l+1}(1))} \not\models \varphi_1 \mathcal{U} \varphi_2)\\ &\quad \Rightarrow (\boldsymbol{K}, \pi^{(d_{l+1}(0),d_{l+1}(1))} \models \Box\varphi_1) \wedge\\ &\quad\ \mathrm{Until}_{l+1}(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, 1)]\end{aligned}$$

which is

$$\begin{aligned}&[(\boldsymbol{K}, \pi^{(d_{l+1}(0),d_{l+1}(1))} \models \varphi_1 \mathcal{U} \varphi_2) \Rightarrow \mathit{True}] \wedge\\ &[(\boldsymbol{K}, \pi^{(d_{l+1}(0),d_{l+1}(1))} \not\models \varphi_1 \mathcal{U} \varphi_2)\\ &\quad \Rightarrow (\boldsymbol{K}, \pi^{(d_{l+1}(0),d_{l+1}(1))} \models \Box\varphi_1) \wedge\\ &\quad\ (\boldsymbol{K}, \pi^{d_{l+1}(1)} \models \varphi_1 \mathcal{U} \varphi_2)]\end{aligned}$$

because of the induction hypothesis instance. From Lemma 3, this is equivalent to $\boldsymbol{K}, \pi \models \varphi_1 \mathcal{U} \varphi_2$.

□

IV. A Divide & Conquer Approach to Until Model Checking Algorithm

An algorithm can be constructed based on Theorem 1, which is shown as Algorithm 1. For each initial state $s_0 \in \boldsymbol{I}$, unfolding s_0 by using $\boldsymbol{T}$ such that each node except for s_0 has exactly one incoming edge, an infinite tree whose root is s_0 is made. The infinite tree may have multiple copies of some states. Such an infinite tree can be divided into $L+1$ layers, generating multiple sub-state spaces, and conducting model checking experiments for each sub-state space. If the set of reachable states is finite, the number of different states in each layer and each sub-state space is finite. Theorem 1 makes it possible to check $\boldsymbol{K} \models \varphi_1 \mathcal{U} \varphi_2$ in a stratified way in that for each layer $l \in \{1, \ldots, L+1\}$, we can check $\boldsymbol{K}, s, d(l) \models \varphi$ for each $s \in \{\pi(d(l-1)) \mid \pi \in \boldsymbol{P}^{d(l-1)}_{(\boldsymbol{K}, s_0)}\}$, where $d(0)$ is 0, $d(x)$ is a non-zero natural number for $x = 1, \ldots, L$, $d(L+1)$ is ∞, and φ is $\varphi_1 \mathcal{U} \varphi_2$ or $\Box\varphi_1$.

$\boldsymbol{US}$ and $\boldsymbol{US'}$ are variables to which sets of states are set. Initially, $\boldsymbol{US}$ contains all initial states in $\boldsymbol{I}$ at line 1. For each layer $l = 0, 1, \ldots, L$ in the first **forall** loop, we need to do as follows. Firstly, $\boldsymbol{US'}$ that is used to collect states at each layer is set to an empty set at line 3. Secondly, the code fragment at lines 4 – 10 checks $\varphi_1 \mathcal{U} \varphi_2$ for each path that starts with each state in $\boldsymbol{US}$. If the path satisfies the formula, we do not need to take the path into account. Otherwise, we check whether the path satisfies $\Box\varphi_1$ at line 7. If so, the last state of the path is then added to $\boldsymbol{US'}$ at line 8. Otherwise, the path is a counterexample and Algorithm 1 returns Failure. Finally, $\boldsymbol{US'}$ is assigned to $\boldsymbol{US}$ for the next layer at line 11.

Just after the first **forall** loop in Algorithm 1, $\boldsymbol{US}$ contains the set of states located at bottom of the Lth layer by checking $\varphi_1 \mathcal{U} \varphi_2$ and $\Box\varphi_2$ for some paths obtained from intermediate

Algorithm 1: A divide & conquer approach to until model checking

input : $\boldsymbol{K}$ – a Kripke structure
φ_1, φ_2 – state propositions
L – a non-zero natural number
d – a function such that $d(x)$ is a non-zero natural number for $x = 1, \ldots, L$
output: Success ($\boldsymbol{K} \models \varphi_1 \mathcal{U} \varphi_2$) or Failure ($\boldsymbol{K} \not\models \varphi_1 \mathcal{U} \varphi_2$)

```
1  US ← I
2  forall l ∈ {1, ..., L} do
3  |  US' ← {}
4  |  forall s ∈ US do
5  |  |  forall π ∈ P^{d(l)}_{(K,s)} do
6  |  |  |  if K, π ⊭ φ1 U φ2 then
7  |  |  |  |  if K, π ⊨ □φ1 then
8  |  |  |  |  |  US' ← US' ∪ {π(d(l))}
9  |  |  |  |  else
10 |  |  |  |  |  return Failure
11 |  US ← US'
12 forall s ∈ US do
13 |  forall π ∈ P_{(K,s)} do
14 |  |  if K, π ⊭ φ1 U φ2 then
15 |  |  |  return Failure
16 return Success
```

layers (1st to Lth layers). For the final layer $L+1$, we check $\varphi_1 \mathcal{U} \varphi_2$ for each path that starts with each state in $\boldsymbol{US}$ in the code fragment at lines 12 – 15. If there is a path that does not satisfy the formula, Algorithm 1 returns Failure. Otherwise, Algorithm 1 returns Success at the end.

V. Multiple Layer Division of Until Stable Model Checking

Lemma 4. *Let φ be any state proposition of $\boldsymbol{K}$. Let k be any natural number. Then,* $(\boldsymbol{K}, \pi \models \Box\varphi) \Leftrightarrow (\boldsymbol{K}, \pi_k \models \Box\varphi) \wedge (\boldsymbol{K}, \pi^k \models \Box\varphi)$.

Proof. Because φ is a state proposition, whether it holds only depends on the first state of a given path. If $(\boldsymbol{K}, \pi \models \Box\varphi)$, then φ holds for $\pi(i)$ for all i, and vice versa. If $\boldsymbol{K}, \pi_k \models \Box\varphi$ and $\boldsymbol{K}, \pi^k \models \Box\varphi$, then φ holds for $\pi(i)$ for $i = 0, \ldots, k$ and φ holds for $\pi(i)$ for $i = k, \ldots$, respectively, and therefore φ holds for $\pi(i)$ for all i, and vice versa. □

Lemma 5. *Let φ_1, φ_2 be any state propositions of $\boldsymbol{K}$.* $(\boldsymbol{K}, \pi \models \Box\varphi_2) \Rightarrow (\boldsymbol{K}, \pi \models \varphi_1 \mathcal{U} \Box\varphi_2)$.

Proof. From the assumption, $\boldsymbol{K}, \pi^0 \models \Box\varphi_2$. □

Lemma 6. *Let φ_1, φ_2 be any state propositions of $\boldsymbol{K}$. Let k be any natural number. Then,* $(\boldsymbol{K}, \pi_k \models \Box\varphi_1) \wedge (\boldsymbol{K}, \pi^k \models \varphi_1 \mathcal{U} \Box\varphi_2) \Rightarrow (\boldsymbol{K}, \pi \models \varphi_1 \mathcal{U} \Box\varphi_2)$.

Proof. If $(\boldsymbol{K}, \pi_k \models \Box\varphi_1)$, then for each $i' \leq k$, $\boldsymbol{K}, \pi_k^{i'} \models \varphi_1$, which is equivalent to $\boldsymbol{K}, \pi^{i'} \models \varphi_1$ from Proposition 1 (1). If $\boldsymbol{K}, \pi^k \models \varphi_1 \mathcal{U} \Box\varphi_2$, then there exists $i \geq k$ such that $\boldsymbol{K}, \pi^i \models \Box\varphi_2$ and for each j such that $k \leq j < i$, $\boldsymbol{K}, \pi^j \models \varphi_1$ (2). From (1) and (2), we have $\boldsymbol{K}, \pi \models \varphi_1 \mathcal{U} \Box\varphi_2$. □

Lemma 7. *Let φ_1, φ_2 be any state propositions of $\boldsymbol{K}$. Let k be any natural number. Then,* $(\boldsymbol{K}, \pi_k \models \varphi_1 \mathcal{U} \Box\varphi_2) \wedge (\boldsymbol{K}, \pi^k \models \Box\varphi_2) \Rightarrow (\boldsymbol{K}, \pi \models \varphi_1 \mathcal{U} \Box\varphi_2)$.

Proof. If $\boldsymbol{K}, \pi_k \models \varphi_1 \mathcal{U} \Box\varphi_2$, then there exists $i \leq k$ such that $\boldsymbol{K}, \pi_k^i \models \Box\varphi_2$, which is equivalent to $\boldsymbol{K}, \pi_k^{i'} \models \varphi_2$ for each $i' \geq i$ (note that $\pi_k^{i'} = \pi_k^k$ if $i' \geq k$), which implies $\boldsymbol{K}, \pi^{i'} \models \varphi_2$ for $k \geq i' \geq i$ from proposition 1, and for each $j < i$, $\boldsymbol{K}, \pi_k^j \models \varphi_1$, which is equivalent to $\boldsymbol{K}, \pi^j \models \varphi_1$ from Proposition 1 (1). If $(\boldsymbol{K}, \pi^k \models \Box\varphi_2)$, then $\boldsymbol{K}, \pi^i \models \varphi_2$ for each $i \geq k$ (2). From (1) and (2), we have $\boldsymbol{K}, \pi \models \varphi_1 \mathcal{U} \Box\varphi_2$. □

Lemma 8 (Two layer division of $\varphi_1 \mathcal{U} \Box\varphi_2$)**.** *Let φ_1, φ_2 be any state propositions of $\boldsymbol{K}$. Let k be any natural number. Then,*

$$
\begin{aligned}
&(\boldsymbol{K}, \pi \models \varphi_1 \mathcal{U} \Box\varphi_2) \\
\Leftrightarrow\ &[(\boldsymbol{K}, \pi_k \models \Box\varphi_1) \Rightarrow (\boldsymbol{K}, \pi^k \models \varphi_1 \mathcal{U} \Box\varphi_2)] \wedge \\
&[(\boldsymbol{K}, \pi_k \not\models \Box\varphi_1) \Rightarrow (\boldsymbol{K}, \pi_k \models \varphi_1 \mathcal{U} \Box\varphi_2) \wedge \\
&\qquad\qquad (\boldsymbol{K}, \pi^k \models \Box\varphi_2)]
\end{aligned}
$$

Proof. (1) Case "only if" ($\Rightarrow$): The case is split into two cases: (1.1) $\boldsymbol{K}, \pi_k \models \Box\varphi_1$ and (1.2) $\boldsymbol{K}, \pi_k \not\models \Box\varphi_1$. From the assumption, there exists i such that $\boldsymbol{K}, \pi^i \models \Box\varphi_2$ and for each $j < i$, $\boldsymbol{K}, \pi^j \models \varphi_1$. In (1.1), if $k < i$, then $\boldsymbol{K}, \pi^k \models \varphi_1 \mathcal{U} \Box\varphi_2$. Otherwise, if $k \geq i, \boldsymbol{K}, \pi^k \models \Box\varphi_2$ from lemma 4. Hence, $\boldsymbol{K}, \pi^k \models \varphi_1 \mathcal{U} \Box\varphi_2$ from lemma 5. In (1.2), from the assumption, $k \geq i$. $\boldsymbol{K}, \pi^k \models \Box\varphi_2$ from lemma 4. We also have $\boldsymbol{K}, \pi_k \models \varphi_1 \mathcal{U} \Box\varphi_2$ because there exists such i in the assumption.

(2) Case "if" ($\Leftarrow$): The case is split into two cases: (2.1) $\boldsymbol{K}, \pi_k \models \Box\varphi_1$ and (2.2) $\boldsymbol{K}, \pi_k \not\models \Box\varphi_1$. In (2.1), $\boldsymbol{K}, \pi \models \varphi_1 \mathcal{U} \varphi_2$ from Lemma 6. In (2.2), $\boldsymbol{K}, \pi \models \varphi_1 \mathcal{U} \varphi_2$ from Lemma 7. □

Definition 5 ($\mathrm{UStable}_L$)**.** *Let L be any non-zero natural number, k be any natural number and d be any function such that $d(0)$ is 0, $d(x)$ is a natural number for $x = 1, \ldots, L$ and $d(L+1)$ is ∞.*

1) $0 \leq k < L - 1$

$$
\begin{aligned}
&\mathrm{UStable}_L(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, k) \\
&\triangleq\ [(\boldsymbol{K}, \pi^{(d(k), d(k+1))} \models \Box\varphi_1) \\
&\qquad \Rightarrow \mathrm{UStable}_L(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, k+1)] \wedge \\
&\quad [(\boldsymbol{K}, \pi^{(d(k), d(k+1))} \not\models \Box\varphi_1) \\
&\qquad \Rightarrow (\boldsymbol{K}, \pi^{(d(k), d(k+1))} \models \varphi_1 \mathcal{U} \Box\varphi_2) \wedge \\
&\qquad\quad (\boldsymbol{K}, \pi^{d(k+1)} \models \Box\varphi_2)]
\end{aligned}
$$

2) $k = L - 1$

$$
\begin{aligned}
&\mathrm{UStable}_L(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, k) \\
&\triangleq\ [(\boldsymbol{K}, \pi^{(d(k), d(k+1))} \models \Box\varphi_1) \\
&\qquad \Rightarrow (\boldsymbol{K}, \pi^{(d(k+1), d(k+2))} \models \varphi_1 \mathcal{U} \Box\varphi_2)] \wedge \\
&\quad [(\boldsymbol{K}, \pi^{(d(k), d(k+1))} \not\models \Box\varphi_1) \\
&\qquad \Rightarrow (\boldsymbol{K}, \pi^{(d(k), d(k+1))} \models \varphi_1 \mathcal{U} \Box\varphi_2) \wedge \\
&\qquad\quad (\boldsymbol{K}, \pi^{d(k+1)} \models \Box\varphi_2)]
\end{aligned}
$$

Theorem 2 ($L+1$ layer division of $\varphi_1 \,\mathcal{U}\, \Box\varphi_2$)**.** *Let L be any non-zero natural number. Let $d(0)$ be 0, $d(x)$ be any natural number for $x = 1, \ldots, L$ and $d(L+1)$ be ∞. Let φ_1, φ_2 be any state propositions of $\boldsymbol{K}$. Then,*

$$(\boldsymbol{K}, \pi \models \varphi_1 \,\mathcal{U}\, \Box\varphi_2) \Leftrightarrow \mathrm{UStable}_L(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, 0)$$

Proof. By induction on L.

- Base case ($L = 1$): It follows from Lemma 8.
- Induction case ($L = l + 1$): We prove the following:

  $$(\boldsymbol{K}, \pi \models \varphi_1 \,\mathcal{U}\, \Box\varphi_2) \Leftrightarrow \mathrm{UStable}_{l+1}(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, 0)$$

  Let d_{l+1} be d used in $\mathrm{UStable}_{l+1}(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, 0)$ such that $d_{l+1}(0) = 0$, $d_{l+1}(i)$ is an arbitrary natural number for $i = 1, \ldots, l+1$ and $d_{l+1}(l+2) = \infty$. The induction hypothesis is as follows:

  $$(\boldsymbol{K}, \pi \models \varphi_1 \,\mathcal{U}\, \Box\varphi_2) \Leftrightarrow \mathrm{UStable}_l(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, 0)$$

  Let d_l be d used in $\mathrm{UStable}_l(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, 0)$ such that $d_l(0) = 0$, $d_l(i)$ is an arbitrary natural number for $i = 1, \ldots, l$ and $d_l(l+1) = \infty$. Because $d_{l+1}(i)$ is an arbitrary natural number for $i = 1, \ldots, l+1$, we suppose that $d_{l+1}(1) = d_l(1)$ and $d_{l+1}(i+1) = d_l(i)$ for $i = 1, \ldots, l$. Because π is any path of $\boldsymbol{K}$, π can be replaced with $\pi^{d_l(1)}$. If so, we have the following as an instance of the induction hypothesis:

  $$(\boldsymbol{K}, \pi^{d_l(1)} \models \varphi_1 \,\mathcal{U}\, \Box\varphi_2) \Leftrightarrow \mathrm{UStable}_l(\boldsymbol{K}, \pi^{d_l(1)}, \varphi_1, \varphi_2, 0)$$

  From Definition 5, $\mathrm{UStable}_l(\boldsymbol{K}, \pi^{d_l(1)}, \varphi_1, \varphi_2, 0)$ is $\mathrm{UStable}_{l+1}(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, 1)$ because $d_l(0) = d_{l+1}(0) = 0$, $d_l(1) = d_{l+1}(1)$, and $d_l(i) = d_{l+1}(i+1)$ for $i = 1, \ldots, l$, and $d_l(l+1) = d_{l+1}(l+2) = \infty$. Therefore, the induction hypothesis instance can be rephrased as follows:

  $$(\boldsymbol{K}, \pi^{d_{l+1}(1)} \models \varphi_1 \,\mathcal{U}\, \Box\varphi_2) \Leftrightarrow \mathrm{UStable}_{l+1}(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, 1)$$

  From Definition 5, $\mathrm{UStable}_{l+1}(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, 0)$ is

  $$\begin{aligned}
  &[(\boldsymbol{K}, \pi^{(d_{l+1}(0), d_{l+1}(1))} \models \Box\varphi_1) \\
  &\quad \Rightarrow \mathrm{UStable}_{l+1}(\boldsymbol{K}, \pi, \varphi_1, \varphi_2, 1)] \wedge \\
  &[(\boldsymbol{K}, \pi^{(d_{l+1}(0), d_{l+1}(1))} \not\models \Box\varphi_1) \\
  &\quad \Rightarrow (\boldsymbol{K}, \pi^{(d_{l+1}(0), d_{l+1}(1)} \models \varphi_1 \,\mathcal{U}\, \Box\varphi_2) \wedge \\
  &\qquad (\boldsymbol{K}, \pi^{d_{l+1}(1)} \models \Box\varphi_2)]
  \end{aligned}$$

  which is

  $$\begin{aligned}
  &[(\boldsymbol{K}, \pi^{(d_{l+1}(0), d_{l+1}(1))} \models \varphi_1 \,\mathcal{U}\, \Box\varphi_2) \\
  &\quad \Rightarrow (\boldsymbol{K}, \pi^{d_{l+1}(1)} \models \varphi_1 \,\mathcal{U}\, \Box\varphi_2)] \wedge \\
  &[(\boldsymbol{K}, \pi^{(d_{l+1}(0), d_{l+1}(1))} \not\models \varphi_1 \,\mathcal{U}\, \Box\varphi_2) \\
  &\quad \Rightarrow (\boldsymbol{K}, \pi^{(d_{l+1}(0), d_{l+1}(1))} \models \varphi_1 \,\mathcal{U}\, \Box\varphi_2) \wedge \\
  &\qquad (\boldsymbol{K}, \pi^{d_{l+1}(1)} \models \Box\varphi_2)]
  \end{aligned}$$

  because of the induction hypothesis instance. From Lemma 8, this is equivalent to $\boldsymbol{K}, \pi \models \varphi_1 \,\mathcal{U}\, \Box\varphi_2$.

□

VI. A Divide & Conquer Approach to Until Stable Model Checking Algorithm

An algorithm can be constructed based on Theorem 2, which is shown as Algorithm 2. For each initial state $s_0 \in \boldsymbol{I}$, the reachable state space from s_0 is divided into $L+1$ layers, generating multiple sub-state spaces, and conducting model checking experiments for each sub-state space. Theorem 2 makes it possible to check $\boldsymbol{K} \models \varphi_1 \,\mathcal{U}\, \Box\varphi_2$ in a stratified way in that for each layer $l \in \{1, \ldots, L+1\}$, we can check $\boldsymbol{K}, s, d(l) \models \varphi$ for each $s \in \{\pi(d(l-1)) \mid \pi \in \boldsymbol{P}^{d(l-1)}_{(\boldsymbol{K}, s_0)}\}$, where $d(0)$ is 0, $d(x)$ is a non-zero natural number for $x = 1, \ldots, L$, $d(L+1)$ is ∞, and φ is $\Box\varphi_1$, or $\Box\varphi_2$, or $\varphi_1 \,\mathcal{U}\, \Box\varphi_2$.

$\boldsymbol{CxS}$, $\boldsymbol{CxS'}$, $\boldsymbol{NCxS}$, and $\boldsymbol{NCxS'}$ are variables to which sets of states are set. Initially, $\boldsymbol{CxS}$ contains all initial states in $\boldsymbol{I}$ at line 1 while $\boldsymbol{NCxS}$ is set to an empty set at line 2. For each layer $l = 0, 1, \ldots, L$ in the first **forall** loop, we need to do as follows. Firstly, $\boldsymbol{CxS'}$ and $\boldsymbol{NCxS'}$ that are used to collect states at each layer are set to an empty set at lines 4 and 5, respectively. Secondly, the code fragment at lines 6 – 14 checks $\Box\varphi_1$ for each path that starts with each state in $\boldsymbol{CxS}$. If the path satisfies the formula, the last state of the path is added to $\boldsymbol{CxS'}$ at line 9. Otherwise, we check whether the path satisfies $\varphi_1 \,\mathcal{U}\, \Box\varphi_2$ at line 11. If so, the last state of the path is added to $\boldsymbol{NCxS'}$ at line 12. Otherwise, the path is a counterexample and Algorithm 2 returns Failure at line 14. Thirdly, the code fragment at lines 15 – 20 checks $\Box\varphi_2$ for each path that starts with each state in $\boldsymbol{NCxS}$. If the path satisfies the formula, the last state of the path is added to $\boldsymbol{NCxS'}$ at line 18. Otherwise, the path is a counterexample and Algorithm 2 returns Failure at line 20. Finally, $\boldsymbol{CxS'}$ and $\boldsymbol{NCxS'}$ are assigned to $\boldsymbol{CxS}$ and $\boldsymbol{NCxS}$ for the next layer at lines 21 and 22, respectively.

Just after the first **forall** loop in Algorithm 2, $\boldsymbol{CxS}$ and $\boldsymbol{NCxS}$ contains the sets of states located at bottom of the Lth layer by checking $\Box\varphi_1$, $\varphi_1 \,\mathcal{U}\, \varphi_2$, and $\Box\varphi_2$ for some paths obtained from intermediate layers (1st to Lth layers). For the final layer $L+1$, we check $\varphi_1 \,\mathcal{U}\, \Box\varphi_2$ for each path that starts with each state in $\boldsymbol{CxS}$ in the code fragment at lines 23 – 26. Meanwhile, we check $\Box\varphi_2$ for each path that starts with each state in $\boldsymbol{NCxS}$ in the code fragment at lines 27 – 30. If there is a path that does not satisfy the formula concerned, Algorithm 2 returns Failure. Otherwise, Algorithm 2 returns Success at the end.

VII. Related Work

SAT/SMT-based bounded model checking (BMC) is an effective technique to mitigate the state space explosion problem in model checking. BMC can find a flaw located within some reasonably shallow depth k for each initial state by formalizing the verification problem into an equisatisfiable conjunctive normal form (CNF) formula that can be analyzed by a SAT/SMT solver. An extension of SAT/SMT-based BMC to model check concurrent programs is Lazy Sequentialization (Lazy-CSeq) [14]. Given a concurrent program P together

Algorithm 2: A divide & conquer approach to until stable model checking

input : $\boldsymbol{K}$ – a Kripke structure
φ_1, φ_2 – state propositions
L – a non-zero natural number
d – a function such that $d(x)$ is a non-zero natural number for $x = 1, \ldots, L$
output: Success ($\boldsymbol{K} \models \varphi_1 \ \mathcal{U} \ \Box\varphi_2$) or Failure ($\boldsymbol{K} \not\models \varphi_1 \ \mathcal{U} \ \Box\varphi_2$)

$\boldsymbol{CxS} \leftarrow \boldsymbol{I}$
$\boldsymbol{NCxS} \leftarrow \emptyset$
forall $l \in \{1, \ldots, L\}$ **do**
  $\boldsymbol{CxS'} \leftarrow \{\}$
  $\boldsymbol{NCxS'} \leftarrow \{\}$
  forall $s \in \boldsymbol{CxS}$ **do**
    forall $\pi \in \boldsymbol{P}^{d(l)}_{(\boldsymbol{K},s)}$ **do**
      if $\boldsymbol{K}, \pi \models \Box\varphi_1$ **then**
        $\boldsymbol{CxS'} \leftarrow \boldsymbol{CxS'} \cup \{\pi(d(l))\}$
      else
        if $\boldsymbol{K}, \pi \models \varphi_1 \ \mathcal{U} \ \Box\varphi_2$ **then**
          $\boldsymbol{NCxS'} \leftarrow \boldsymbol{NCxS'} \cup \{\pi(d(l))\}$
        else
          return Failure
  forall $s \in \boldsymbol{NCxS}$ **do**
    forall $\pi \in \boldsymbol{P}^{d(l)}_{(\boldsymbol{K},s)}$ **do**
      if $\boldsymbol{K}, \pi \models \Box\varphi_2$ **then**
        $\boldsymbol{NCxS'} \leftarrow \boldsymbol{NCxS'} \cup \{\pi(d(l))\}$
      else
        return Failure
  $\boldsymbol{CxS} \leftarrow \boldsymbol{CxS'}$
  $\boldsymbol{NCxS} \leftarrow \boldsymbol{NCxS'}$
forall $s \in \boldsymbol{CxS}$ **do**
  forall $\pi \in \boldsymbol{P}_{(\boldsymbol{K},s)}$ **do**
    if $\boldsymbol{K}, \pi \not\models \varphi_1 \ \mathcal{U} \ \Box\varphi_2$ **then**
      return Failure
forall $s \in \boldsymbol{NCxS}$ **do**
  forall $\pi \in \boldsymbol{P}_{(\boldsymbol{K},s)}$ **do**
    if $\boldsymbol{K}, \pi \not\models \Box\varphi_2$ **then**
      return Failure
return Success

with two parameters u and r that are the loop unwinding bound and the number of round-robin schedules, respectively, they first generate an intermediate bounded program P_u by unwinding all loops and inlining all function calls in P with u as a bound except for those used for creating threads. P_u then is transformed into a sequential program $Q_{u,r}$ that simulates all behaviors of P_u within r round-robin schedules. $Q_{u,r}$ is then transformed into a propositional formula that can be analyzed by a SAT/SMT solver. When the size of the system under test is large, the propositional formula becomes complex and the performance of the SAT/SMT solver is degraded or the model checking may become infeasible. To make it possible to conduct model checking experiments. They decompose the set of execution traces of concurrent programs into symbolic subsets [15] so that the single formula is divided into multiple smaller propositional sub-formulas, which then are possibly analyzed by the SAT/SMT solver independently. Their technique is able to deal with safety properties, while our technique is able to deal with until and until stable properties, a class of liveness properties.

VIII. Conclusion

We have described the divide & conquer approach to until and until stable model checking in which for each property, we have proved a theorem that the proposed technique is correct and designed an algorithm based on the theorem to support the technique. As one piece of our future work, we will build a tool supporting the proposed technique and conduct case studies in real-time systems, particularly with RT-Maude, demonstrating that the proposed technique and tool are useful.

References

[1] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, Eds., *Handbook of Model Checking*. Berlin, Heidelberg: Springer, 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-10575-8

[2] R. E. Bryant and C. Meinel, *Ordered Binary Decision Diagrams*. Boston, MA: Springer US, 2002, pp. 285–307. [Online]. Available: https://doi.org/10.1007/978-1-4615-0817-5_11

[3] E. M. Clarke, O. Grumberg, M. Minea, and D. A. Peled, "State space reduction using partial order techniques," *Int. J. Softw. Tools Technol. Transf.*, vol. 2, no. 3, pp. 279–287, 1999.

[4] E. M. Clarke, O. Grumberg, and D. E. Long, "Model checking and abstraction," *ACM Trans. Program. Lang. Syst.*, vol. 16, no. 5, pp. 1512–1542, 1994.

[5] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement for symbolic model checking," *J. ACM*, vol. 50, no. 5, pp. 752–794, 2003.

[6] J. Meseguer, M. Palomino, and N. Martí-Oliet, "Equational abstractions," *Theor. Comput. Sci.*, vol. 403, no. 2-3, pp. 239–264, 2008.

[7] Y. Phyo, C. M. Do, and K. Ogata, "A divide & conquer approach to leads-to model checking," *The Computer Journal*, 2021. [Online]. Available: https://doi.org/10.1093/comjnl/bxaa183

[8] ——, "A divide & conquer approach to conditional stable model checking," in *18th ICTAC*, 2021, pp. 105–111. [Online]. Available: https://doi.org/10.1007/978-3-030-85315-0_7

[9] M. N. Aung, Y. Phyo, C. M. Do, and K. Ogata, "A divide & conquer approach to eventual checking," *Mathematics*, vol. 9, p. 368, 2021. [Online]. Available: https://doi.org/10.3390/math9040368

[10] P. C. Ölveczky, "Real-time maude and its applications," in *Rewriting Logic and Its Applications*, S. Escobar, Ed. Cham: Springer International Publishing, 2014, pp. 42–79.

[11] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, Eds., *All About Maude*, ser. LNCS. Springer, 2007, vol. 4350.

[12] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott, "Full maude: Extending core maude," 01 2007, pp. 559–597.

[13] P. C. Ölveczky, M. Keaton, J. Meseguer, C. L. Talcott, and S. Zabele, "Specification and analysis of the aer/nca active network protocol suite in real-time maude," in *Proceedings of the 4th International Conference on Fundamental Approaches to Software Engineering*, ser. FASE '01. Berlin, Heidelberg: Springer-Verlag, 2001, p. 333–348.

[14] O. Inverso, E. Tomasco, B. Fischer, S. La Torre, and G. Parlato, "Bounded verification of multi-threaded programs via lazy sequentialization," *ACM Trans. Program. Lang. Syst.*, vol. 44, no. 1, dec 2021.

[15] O. Inverso and C. Trubiani, "Parallel and distributed bounded model checking of multi-threaded programs," in *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 202–216.