Formal Methods in System Design, 17, 39–60, 2000

Parametric Analysis of Computer Systems*

FARN WANG farn@iis.sinica.edu.tw
Institute of Information Science, Academia Sinica, Taipei, Taiwan 115, ROC

Received September 15, 1997; Revised June 3, 1999

Abstract. A general parametric analysis problem which allows the use of parameter variables in both the real-time automata and the specifications is proposed and solved. The analysis algorithm is much simpler and can run more efficiently in average cases than can previous works.

Keywords: parameters, real-time systems, verification, model-checking

1. Introduction

Successful real-world project management relies on the satisfaction of various timing and nontiming restraints which may compete with each other for resources. Examples of such restraints include timely responses, budget, domestic or international regulations, system configurations, environments, compatibilities, etc. In this work, we define and algorithmically solve the *parametric analysis problem* of computer systems which allows for the formal description of system behaviors and design requirements with various timing and nontiming parameter variables and requires general conditions for all solutions to those parameter variables.

The design of our problem was influenced by previous work of Alur et al. [4] and Wang [12] which will be discussed briefly later. Our parametric analysis problem is presented in two parts: an automaton with nontiming parameter variables and a specification with both timing and nontiming parameter variables. The following example is adapted from the railroad crossing example and shows how such a platform can be useful.

Example (Railroad Gate Controller). The popular railroad crossing example consists of a train monitor and gate-controller. In figure 1, we give a parametric version of the automaton descriptions of the monitor and controller, respectively. The ovals represent meta-states while the arcs represent transitions. For each transition, we label its triggering condition and the clocks which are to be reset to zero on its firing. The global state space can be calculated as the Cartesian-product of local state spaces.

The safety requirement is that whenever a train is at the crossing, the gate must be in the D mode (gate is down). The more money you spend on monitoring, the more precisely

*The paper has been accepted for publication in Proceedings of the 6th AMAST, December 1997, Sydney, Australia.

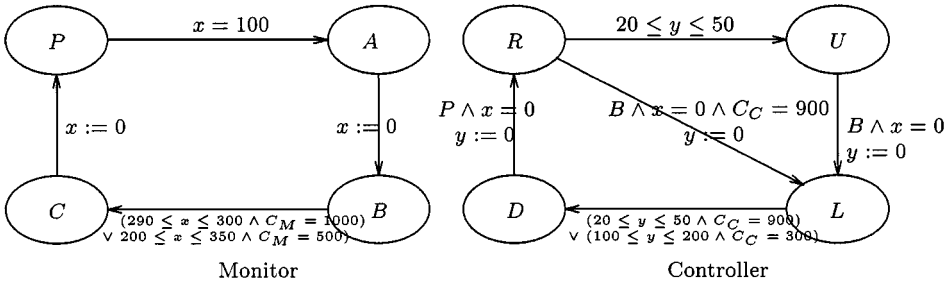

Figure 1. Railroad gate controller example.

you can tell how far away an approaching train is. Suppose we now have two types of monitoring, one costs 1000 dollars and can tell if a train will arrive at the crossing in 290 to 300 seconds; the other type costs 500 and can tell if a train will arrive at the crossing in 200 to 350 seconds.

We also have two gate-controller types. One costs 900 dollars and can lower the gate in 20 to 50 seconds and skip the U mode (gate is up) when a train is coming to the crossing and the controller is in the R mode (gate-Raising mode). The other type costs 300 dollars and can lower the gate in 100 to 200 seconds but cannot skip the U mode once the controller is in the R mode.

Suppose now the design of a rail-road crossing gate-controller is subjected to the following budget constraint : the cost of the monitor ($\$_M$) and that of controller ($\$_C$) together cannot exceed 1500 dollars. We want to make sure that, under this constraint, the safety requirement can still be satisfied. This can be expressed in our logic PCTL as $\$_M + \$_C \leq 1500 \wedge \forall\Box(C \rightarrow D)$. Here $\forall\Box$ is a modal operator from CTL [7, 8] which means that for all computations henceforth, the following statement must be true.

Our system behavior descriptions are given in *statically parametric automata* (SPA), and our specifications are given in *parametric computation tree logic* (PCTL). The outcomes of our algorithm are Boolean expressions, whose literals are linear inequalities on the parameter variables and can be further processed using standard techniques like simplex, simulated annealing, etc. to extract useful design feedback.

In the remainder of the introduction, we shall first briefly discuss work related to this subject and then outline the rest of the paper.

1.1. Related work

In the earliest works [7, 8], researchers used finite-state automata to describe system behavior and check if they satisfied specifications given in branching-time temporal logic CTL. Such a framework is usually called *model-checking*. A CTL (*Computation Tree Logic*) formulus is composed of binary propositions ($p, q, \ldots$), Boolean operators ($\neg, \vee, \wedge$), and branching-time modal operators ($\exists\mathcal{U}, \exists\bigcirc, \forall\mathcal{U}, \forall\bigcirc$). $\exists$ means "*there exists*" a computation. $\forall$ means

"*for all*" computations. $\mathcal{U}$ means something is true "until" something else is true. $\bigcirc$ means "*next state*." For example, $\exists p \mathcal{U} q$ says that there exists a computation in which p is true until q is true. Since there is no notion of real-time (clock time), only ordering among events is considered.

The following shorthand are generally accepted besides the usual ones in Boolean algebra. $\exists\Diamond\phi_1$ stands for $\exists true\ \mathcal{U}\phi_1$; $\forall\Box\phi_1$ for $\neg\exists\Diamond\neg\phi_1$; $\forall\Diamond\phi_1$ for $\forall true\ \mathcal{U}\phi_1$; and $\exists\Box\phi_1$ for $\neg\forall\Diamond\neg\phi_1$. Intuitively, $\Diamond$ means "*eventually*" while $\Box$ means "*henceforth*."

CTL model-checking has been used to prove the correctness of concurrent systems such as circuits and communication protocols. In 1990, the platform was extended by Alur et al. to the *Timed* CTL (TCTL) *model-checking problem* to verify dense-time systems equipped with resettable clocks [1]. Alur et al. also solved the problem in the same paper using an innovative state space partitioning scheme.

In [9], the problem of deciding the earliest and latest times a target state can appear in the computation of a timed automaton was discussed. However, they did not derive the general conditions for parameter variables.

In 1993, Alur et al. studied on the reachability problem of real-time automata with parameter variables [4]. In particular, they established that in general, the problem has no algorithm when three clocks are compared with parameter variables in the automata [4]. This observation greatly influences the design of our platform.

In 1995, Wang proposed another platform which extends the TCTL model-checking problem to allow for timing parameter variables in TCTL formulae [12]. His algorithm gives back Boolean conditions whose literals are linear equalities for the timing parameter variables. He also showed that his parametric timing analysis problem is PSPACE-hard while his analysis algorithm has double-exponential time complexity. In comparision, our work provides a generalized framework for analysis of both timing and static parameter variables. Also, our algorithm employs cycle-reduction in the dynamic programming style, which is easier to understand, implement, and analyze. Moreover, since the cycles are implicitly represented in succinct semilinear expressions,[1] our algorithm can run more efficiently with respect to time and space complexities in most cases.

Henzinger et al.'s HyTech system developed at Cornell also has parametric analysis power [4, 10]. However in this framework, the authors did not identify a decidable class for the parametric analysis problem, and their procedure is not guaranteed to terminate. In comparison, our framework has an algorithm which can generate a semilinear description of the working solutions for the parameter variables.

1.2. Outline

Section 2 presents our system behavior description language: the *Statically Parametric Automaton* (SPA). Section 3 defines the *Parametric Computation Tree Logic* (PCTL) and the *Parametric Analysis Problem*. Section 4 presents the algorithm, proves its correctness, and analyzes its complexity. Section 5 concludes the paper.

We also adopt $\mathcal{N}$ and $\mathcal{R}^+$ as the sets of nonnegative integers and nonnegative real numbers respectively. Given real numbers $\delta \leq \delta'$, (δ, δ'), $(\delta, \delta']$, $[\delta, \delta')$, and $[\delta, \delta']$ traditionally denote *intervals* with left and right endpoints δ, δ', respectively. A left (right) square (round)

bracket means the corresponding endpoint is included (not included). For any $\delta \in \mathcal{R}^+$, (δ, δ), $(\delta, \delta]$, $[\delta, \delta)$ are all *empty* intervals because δ is not included. $[\delta, \delta]$ is a *point* interval because it contains the sole element δ. Since intervals are also sets, we shall adopt the traditional operations on sets for our intervals. For example, given two intervals λ, λ', we may require that $\lambda \subseteq \lambda'$ and calculate $\lambda' - \lambda$.

2. Statically parametric automata (SPA)

In an SPA, we may combine propositions, timing inequalities for clock readings, and linear inequalities of parameter variables to write the invariance and transition conditions. Such a combination is called a *state predicate* and is defined formally in the following. Given a set P of atomic propositions, a set C of clocks, and a set H of parameter variables, a *state predicate* η of P, C, and H, has the following syntax rules:

$$\eta ::= \mathit{false} \mid p \mid x - y \sim c \mid x \sim c \mid \sum a_i \alpha_i \sim c \mid \eta_1 \vee \eta_2 \mid \neg \eta_1$$

where $p \in P$, $x, y \in C$, $a_i, c \in \mathcal{N}$, $\alpha_i \in H$, $\sim \in \{\leq, <, =, \geq, >\}$, and η_1, η_2 are state predicates. Notationally, we let $B(P, C, H)$ be the set of all state predicates on P, C, and H. Note that the parameter variables considered in H are static because their values do not change with time during each computation of an automaton. A state predicate with only $\sum a_i \alpha_i \sim c$ type literals is said to be *static*.

Definition 1 (Statically Parametric Automata). A *Statically Parametric Automaton* (SPA) is a tuple $(Q, q_0, P, C, H, \chi, E, \rho, \tau)$ with the following restrictions.

- Q is a finite set of meta-states.
- $q_0 \in Q$ is the initial meta-state.
- P is a set of atomic propositions.
- C is a set of clocks.
- H is a set of parameter variables.
- $\chi : Q \mapsto B(P, C, H)$ is a function that labels each meta-state with a condition true in that meta-state.
- $E \subseteq Q \times Q$ is the set of transitions.
- $\rho : E \mapsto 2^C$ defines the set of clocks to be reset during each transition.
- $\tau : E \mapsto B(P, C, H)$ defines the transition triggering conditions.

An SPA starts execution at its meta-state q_0. We shall assume that initially, all clocks read zero. In between meta-state transitions, all clocks increment their readings at a uniform rate. The transitions of the SPA may be fired when the triggering conditon is satisfied. Depending on the interpretation of the parameter variables, an SPA may exhibit different behaviors. During a transition from meta-state q_i to q_j, for each $x \in \rho(q_i, q_j)$, the reading of x will be reset to zero. There are state predicates with parameter variables labeled on the states as well as on the transitions. These parameters may also appear in the specifications of the same analysis problem instance.

Definition 2 (State). A *state* s of SPA $A = (Q, q_0, P, C, H, \chi, E, \rho, \tau)$ is a mapping from $P \cup C$ to $\{true, false\} \cup \mathcal{R}^+$ such that for each $p \in P$, $s(p) \in \{true, false\}$ and for each $x \in C$, $s(x) \in \mathcal{R}^+$, where $\mathcal{R}^+$ is the set of nonnegative real numbers.

The same SPA may generate different computations depending on the interpretation of its parameter variables. An *interpretation*, $\mathcal{I}$, for H is a mapping from $\mathcal{N} \cup H$ to $\mathcal{N}$ such that for all $c \in \mathcal{N}$, $\mathcal{I}(c) = c$. An SPA $A = (Q, q_0, P, C, H, \chi, E, \rho, \tau)$ is said to be interpreted with respect to $\mathcal{I}$ when the parameter variables of all the state predicates in A are interpreted according to $\mathcal{I}$.

Definition 3 (Satisfaction of interpreted state predicates by a state). State predicate η is satisfied by state s under interpretation $\mathcal{I}$, written as $s \models_{\mathcal{I}} \eta$, iff

- $s \not\models_{\mathcal{I}} false$;
- $s \models_{\mathcal{I}} p$ iff $s(p) = true$;
- $s \models_{\mathcal{I}} x - y \sim c$ iff $s(x) - s(y) \sim c$;
- $s \models_{\mathcal{I}} x \sim c$ iff $s(x) \sim c$;
- $s \models_{\mathcal{I}} \sum a_i \alpha_i \sim c$ iff $\sum a_i \mathcal{I}(\alpha_i) \sim c$;
- $s \models_{\mathcal{I}} \eta_1 \vee \eta_2$ iff $s \models_{\mathcal{I}} \eta_1$ or $s \models_{\mathcal{I}} \eta_2$; and
- $s \models_{\mathcal{I}} \neg\eta_1$ iff $s \not\models_{\mathcal{I}} \eta_1$.

Now we will define the computation of SPA. For convenience, we adopt the following conventions.

An SPA $A = (Q, q_0, P, C, H, \chi, E, \rho, \tau)$ is *unambiguous* iff for all states s, there is at most one $q \in Q$ such that for some I, $s \models_{\mathcal{I}} \chi(q)$. Ambiguous SPA's can be made unambiguous by incorporating meta-state names as propositional conjuncts in the conjunctive normal forms of the $\chi(\ )$-state predicate of each meta-state. For convenience, from now on, we shall only talk about unambiguous SPA's. When we say an SPA, we mean an unambiguous SPA.

Given an SPA $A = (Q, q_0, P, C, H, \chi, E, \rho, \tau)$, an interpretation $\mathcal{I}$ for H, and a state s, we let s^Q be the meta-state in Q such that $s \models_{\mathcal{I}} \chi(s^Q)$. If there is no meta-state $q \in Q$ such that $s \models_{\mathcal{I}} \chi(q)$, then s^Q is undefined.

Given two states s, s', there is a *meta-state transition* from s to s' in A under interpretation $\mathcal{I}$, in symbols $s \rightarrow_{\mathcal{I}} s'$, iff

- s^Q, s'^Q are both defined,
- $(s^Q, s'^Q) \in E$,
- $s \models_{\mathcal{I}} \tau(s^Q, s'^Q)$, and
- $\forall x \in C((x \in \rho(s^Q, s'^Q) \Rightarrow s'(x) = 0) \wedge (x \notin \rho(s^Q, s'^Q) \Rightarrow s'(x) = s(x)))$.

Also, given a state s and a $\delta \in \mathcal{R}^+$, we let $s + \delta$ be the state that agrees with s in every aspect except that for all $x \in C$, $s(x) + \delta = (s + \delta)(x)$.

Definition 4 (s-run of interpreted SPA). Given a state s of SPA $A = (Q, q_0, P, C, H, \chi, E, \rho, \tau)$ and an interpretation $\mathcal{I}$, a computation of A starting at s is called an *s-run* and is a sequence $((s_1, t_1), (s_2, t_2), \ldots)$ of pairs such that

- $s = s_1$; and
- for each $t \in \mathcal{R}^+$, there is an $i \in \mathcal{N}$ such that $t_i \geq t$; and
- for each integer $i \geq 1$, s_i^Q is defined and for each real $0 \leq \delta \leq t_{i+1} - t_i$, $s_i + \delta \models_{\mathcal{I}} \chi(s_i^Q)$; and
- for each $i \geq 1$, A goes from s_i to s_{i+1} because of
  - a meta-state transition, i.e. $t_i = t_{i+1} \wedge s_i \rightarrow_{\mathcal{I}} s_{i+1}$; or
  - time passage, i.e. $t_i < t_{i+1} \wedge s_i + t_{i+1} - t_i = s_{i+1}$.

3. PCTL and parametric analysis problem

Parametric Computation Tree Logic (PCTL) is used to specify the design requirements and is defined with respect to a given SPA. Suppose we are given an SPA $A = (Q, q_0, P, C, H, \chi, E, \rho, \tau)$. A PCTL formulus ϕ for A has the following syntax rules:

$$\phi ::= \eta \mid \phi_1 \vee \phi_2 \mid \neg\phi_1 \mid \exists\phi_1 \mathcal{U}_{\sim\theta} \phi_2 \mid \forall\phi_1 \mathcal{U}_{\sim\theta} \phi_2$$

Here, η is a state predicate in $B(P, C, H)$, ϕ_1 and ϕ_2 are PCTL formulae, and θ is an element in $\mathcal{N} \cup H$.

Note that the parameter variable subscripts of modal formulae can also be used as parameter variables in SPA. Also, we adopt the following standard shorthand: *true* for $\neg$*false*, $\phi_1 \wedge \phi_2$ for $\neg((\neg\phi_1) \vee (\neg\phi_2))$, $\phi_1 \rightarrow \phi_2$ for $(\neg\phi_1) \vee \phi_2$, $\exists\Diamond_{\sim\theta}\phi_1$ for $\exists$*true* $\mathcal{U}_{\sim\theta}\phi_1$, $\forall\Box_{\sim\theta}\phi_1$ for $\neg\exists\Diamond_{\sim\theta}\neg\phi_1$, $\forall\Diamond_{\sim\theta}\phi_1$ for $\forall$*true* $\mathcal{U}_{\sim\theta}\phi_1$, and $\exists\Box_{\sim\theta}\phi_1$ for $\neg\forall\Diamond_{\sim\theta}\neg\phi_1$.

Depending on the interpretation, a PCTL formulus may impose different requirements. We use the notation $s \models_{\mathcal{I}} \phi$ to mean that ϕ is satisfied at state s in A under interpretation $\mathcal{I}$. The satisfaction relation is defined inductively as follows.

- If ϕ is a state predicate, then $s \models_{\mathcal{I}} \phi$ iff ϕ is satisfied by s as a state predicate under $\mathcal{I}$.
- $s \models_{\mathcal{I}} \phi_1 \vee \phi_2$ iff either $s \models_{\mathcal{I}} \phi_1$ or $s \models_{\mathcal{I}} \phi_2$.
- $s \models_{\mathcal{I}} \neg\phi_1$ iff $s \not\models_{\mathcal{I}} \phi_1$.
- $s \models_{\mathcal{I}} (\exists\phi_1 \mathcal{U}_{\sim\theta} \phi_2)$ iff there are an s-run $= ((s_1, t_1), (s_2, t_2), \ldots)$ in A, an $i \geq 1$, and a nonempty interval $\langle\delta_1, \delta_2\rangle \subseteq [0, t_{i+1} - t_i]$ with "$\langle$" $\in$ {"(", "["} and "$\rangle$" $\in$ {")", "]"}, such that
  - for all $\delta \in \langle\delta_1, \delta_2\rangle$, $t_i + \delta \sim t_1 + \mathcal{I}(\theta)$ and $s_i + \delta \models_{\mathcal{I}} \phi_2$, and
  - for all $0 \leq j \leq i$ and $\delta \in [0, t_{j+1} - t_j]$, if $t_j + \delta \in [0, \delta_1] - \langle\delta_1, \delta_2\rangle$, then $s_j + \delta \models_{\mathcal{I}} \phi_1$.
- $s \models_{\mathcal{I}} (\forall\phi_1 \mathcal{U}_{\sim\theta} \phi_2)$ iff for every s-run $= ((s_1, t_1), (s_2, t_2), \ldots)$ in A, there are an $i \geq 1$ and a nonempty interval $\langle\delta_1, \delta_2\rangle \subseteq [0, t_{i+1} - t_i]$ with "$\langle$" $\in$ {"(", "["} and "$\rangle$" $\in$ {")", "]"}, such that
  - for all $\delta \in \langle\delta_1, \delta_2\rangle$, $t_i + \delta \sim t_1 + \mathcal{I}(\theta)$ and $s_i + \delta \models_{\mathcal{I}} \phi_2$, and
  - for all $0 \leq j \leq i$ and $\delta \in [0, t_{j+1} - t_j]$, if $t_j + \delta \in [0, \delta_1] - \langle\delta_1, \delta_2\rangle$, then $s_j + \delta \models_{\mathcal{I}} \phi_1$.

Note that in our definitions for $\exists\mathcal{U}_{\sim\theta}$ and $\forall\mathcal{U}_{\sim\theta}$, we require that ϕ_2 be satisfied by a nonempty interval of states instead of just a point state as in the case of classic temporal logic.

This change is purely for dense-time semantics and is justified as follows. Note that our computation is defined as a dense line of mappings from propositions to truth values and from clocks to reals. For example, we may have a formulus like $\exists(x \leq 1)\mathcal{U}(x > 1)$, in which $\phi_2 = x > 1$. Since there is really not a "smallest" real bigger than 1, there is really not a state in any run which "first" satisfies $x > 1$ either. Thus a formulus like $\exists(x \leq 1)\mathcal{U}(x > 1)$ is unsatisfiable if we define the satisfaction of ϕ_2 by a point state. This may be counter-intuitive for some readers. Thus we choose to define the semantics of $\mathcal{U}$-formulae with respect to intervals.

Given an SPA A, a PCTL formulus ϕ, and an interpretation $\mathcal{I}$ for H, we say that A is a *model* of ϕ under $\mathcal{I}$, written as $A \models_{\mathcal{I}} \phi$, iff $s \models_{\mathcal{I}} \phi$ for all states s such that $s^Q = q_0$.

We will now formally define our problem.

Definition 5 (Statically Parametric Analysis Problem). Given an SPA A and a specification (PCTL formulus) ϕ, the *parametric analysis problem instance* for A and ϕ, denoted PAP(A, ϕ), is formally defined as the problem of deriving the general condition of all interpretation $\mathcal{I}$ such that $A \models_{\mathcal{I}} \phi$. $\mathcal{I}$ is called a *solution* to PAP(A, ϕ) iff $A \models_{\mathcal{I}} \phi$.

We will show that such conditions are always expressible as Boolean combinations of linear inequalities of parameter variables.

4. Parametric analysis

In this section, we shall develop new data-structures, the *parametric region graph* and the *conditional path graph*, for solving the parametric analysis problem. The parametric region graph is similar to the region graph defined in [1], but it contains parametric information. A region is a subset of the state space in which all states exhibit the same behavior with respect to the given SPA and PCTL formulus.

Given a parametric analysis problem for A and ϕ, a modal subformulus ϕ_1 of ϕ, and the parametric region graph with region sets V, the *conditional path graph* for ϕ_1 is a fully connected graph of V whose arcs are labeled with sets of pairs of the form (π, T), where π is a static state predicate and T is an integer set. For convenience, we call such pairs *conditional time expressions* (CTE). Alternatively, we can say that the conditional path graph $J[\phi_1]$ for ϕ_1 is a mapping from $V \times V$ to the power set of CTE's. For any $v, v' \in V$, if $(\pi, T) \in J[\phi_1](v, v')$, then for all interpretation $\mathcal{I}$, $t \in T$, and $s \in v$, if π is satisfied by $\mathcal{I}$, then there is a finite s-run of time t ending at an $s' \in v'$ such that ϕ_1 is satisfied all the way through the run except at s'. In Subsection 4.2, we shall show that all our modal formulus evaluations can be decomposed to the computation of conditional time expressions.

The kernel of this section is a Kleene's closure procedure which computes the conditional path graph. Its computation utilizes the following four types of integer set manipulations.

- $T_1 \cup T_2$ means $\{a \mid a \in T_1 \text{ or } a \in T_2\}$.
- $T_1 + T_2$ means $\{a_1 + a_2 \mid a_1 \in T_1; a_2 \in T_2\}$.
- T_1* means $\{0\} \cup \bigcup_{i \in \mathcal{N}} \sum_{1 \leq j \leq i} T_1$, where $\sum_{1 \leq j \leq i} T_1$ means the addition of i consecutive T_1.
- $\bar{T}_1$ is the complement of T_1, i.e., $\{a_1 \mid a_1 \in \mathcal{N}; a_1 \notin T_1\}$.

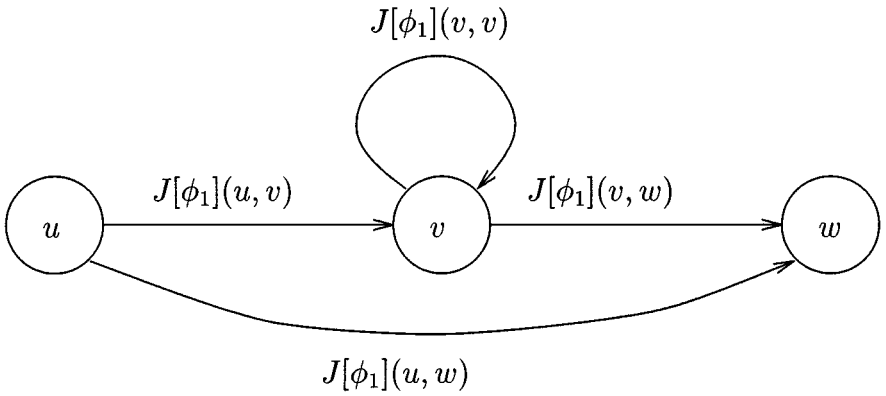

Figure 2. Central operation in our Kleene's closure algorithm.

It can be shown that all integer sets resulting from such manipulations in our algorithm are semilinear. Semilinear expressions are convenient notations for expressing infinite integer sets that are constructed from regular structures (finite-state automata). They are closed under the four manipulations. There are also algorithms to compute the manipulation results. Specifically, we know that all semilinear expressions can be represented as the union of a finite number of sets like $a + c*$. Such a special form is called the *periodical normal form* (PNF). It is not difficult to prove that given operands in PNF, the results of the four manipulations can all be transformed back into PNF. Due to space limitations, we shall skip the details here.

The idea behind our algorithm for computing the conditional path graph is a vertex bypassing scheme. Suppose we have three regions u, v, w whose connections in the conditional path graph are those shown in figure 2. Then it is clear that by bypassing region v, we obtained that $J[\phi_1](u, w)$ should be a superset of $H[\phi_1](u, v, w)$ equal to

$$\left\{ \left(\pi_1 \wedge \pi_2 \wedge \bigwedge_{(\pi_3, T_3) \in D} \pi_3,\ T_1 + T_2 + \sum_{(\pi_3, T_3) \in D} T_3 * \right) \;\middle|\; \begin{array}{l} (\pi_1, T_1) \in J[\phi_1](u, v); \\ (\pi_2, T_2) \in J[\phi_1](v, w); \\ D \subseteq J[\phi_1](v, v) \end{array} \right\}$$

Intuitively, this means that any path from u to w through v can be a concatenation of a path from u to v, any cycles through v, and a path from v to w. The calculation of sets like this will be the kernel operation of our algorithm. Our conditional path graph construction algorithm utilizes a Kleene's closure framework to calculate all the arc labels.

In Subsection 4.1, we extend the regions graph concepts in [1] and define the parametric region graph. In Subsection 4.2, we define the conditional path graph and present an algorithm to compute it. In Subsections 4.3, we present our labelling algorithm for the parametric analysis problem and prove its correctness. In Subsection 4.4, we briefly analyze our algorithm's complexity.

4.1. Parametric region graph

The brilliant concept of region graphs were originally proposed by Alur et al. [1] for use in verifying dense-time systems. A region graph partitions its system state space into finitely many behavior-equivalent subspaces. Our parametric region graphs extend region graphs and contain information about parameter variable restrictions. Besides parameter variables, each parametric region graph has an additional clock κ which is reset to zero once its reading reaches one. κ is not used in the user-given SPA and is added when we construct regions for the convenience of parametric timing analysis. It functions as a ticking indicator for evaluating timed modal formulae of PCTL. The κ reading is always between 0 and 1; that is, for every state s, $0 \leq s(\kappa) \leq 1$.

The *timing constants* in an SPA A are the integer constants c that appear in conditions such as $x - y \sim c$ and $x \sim c$ in A. The timing constants in a PCTL formulus ϕ are the integer constants c that appear in subformulae like $x - y \sim c$, $x \sim c$, $\exists\phi_1 \mathcal{U}_{\sim c}\phi_2$, and $\forall\phi_1 \mathcal{U}_{\sim c}\phi_2$. Let $K_{A:\phi}$ be the largest timing constant used in both A and ϕ for the given parametric analysis problem instance.

For each $\delta \in \mathcal{R}^+$, we define *fract*(δ) as the fractional part of δ, i.e., *fract*$(\delta) = \delta - \lfloor\delta\rfloor$.

Definition 6 (Regions). Given an SPA $A = (Q, q_0, P, C, H, \chi, E, \rho, \tau)$ and a PCTL formulus ϕ for A, two states s, s' of A, $s \cong_{A:\phi} s'$ (i.e., s and s' are equivalent with respect to A and ϕ) iff the following conditions are satisfied.

- For each $p \in P$, $s(p) = s'(p)$.
- For each $x - y \sim c$ used in A or ϕ, $s(x) - s(y) \sim c$ iff $s'(x) - s'(y) \sim c$.
- For each $x \in C$, if either $s(x) \leq K_{A:\phi}$ or $s'(x) \leq K_{A:\phi}$, then $\lfloor s(x)\rfloor = \lfloor s'(x)\rfloor$.
- For every $x, y \in C \cup \{0, \kappa\}$, *fract*$(s(x)) \leq$ *fract*$(s(y))$ iff *fract*$(s'(x)) \leq$ *fract*$(s'(y))$, where $s(0) = s'(0) = 0$.

$[s]$ denotes the equivalent class of A's states, with respect to relation $\cong_{A:\phi}$, to which s belongs, and it is called a *region*.

Note that because of our assumption of unambiguous SPA's, we know that for all $s' \in [s]$, $s'^Q = s^Q$. Using the above definition, the parametric region graph is defined as follows.

Definition 7 (Parametric Region Graph (PR-graph)). The *Parametric Region Graph* (PR-graph) for an SPA $A = (Q, q_0, P, C, H, \chi, E, \rho, \tau)$ and a PCTL formulus ϕ is a directed graph $G_{A:\phi} = (V, F)$ such that the vertex set V is the set of all regions, and the arc set F consists of the following two types of arcs.

- An arc (v, v') may represent *meta-state transitions* in A. That is, for every $s \in v$, there is an $s' \in v'$ such that $s \to s'$.
- An arc (v, v') may be a *time arc* and may represent passage of time in the same meta-state. Formally, for every $s \in v$, there is an $s' \in v'$ such that
  - $s + \delta = s'$ for some $\delta \in \mathcal{R}^+$;

– there is neither $\dot{s}$ nor $\dot{\delta} \in \mathcal{R}^+$, $0 < \dot{\delta} < \delta$, s.t. $[\dot{s}] \neq v$, $[\dot{s}] \neq v'$, $s + \dot{\delta} = \dot{s}$, and $\dot{s} + \delta - \dot{\delta} = s'$.

We shall adopt the following notations for convenience of discussion. Given a region v, $v \models \mathit{fract}(\kappa) = 0$ iff for all $s \in v$, $s(\kappa)$ is an integer. Given an arc (v, v'), $\epsilon(v, v') = \uparrow$ if $v \not\models \mathit{fract}(\kappa) = 0 \wedge v' \models \mathit{fract}(\kappa) = 0$; $\epsilon(v, v') = \downarrow$ if $v \models \mathit{fract}(\kappa) = 0 \wedge v' \not\models \mathit{fract}(\kappa) = 0$; $\epsilon(v, v') = 0$ otherwise.

Also, we write $v \models_{\mathcal{I}} \phi_1$ for some PCTL formulus ϕ_1 when for all $s \in v (s \models_{\mathcal{I}} \phi_1)$. Remember that earlier (page 5), we let s^Q be the meta-state in Q such that $s \models_{\mathcal{I}} \chi(s^Q)$. Similarly, we let v^Q be the meta-state such that for all $s \in v$, $v^Q = s^Q$.

Since regions have enough informations to determine the truth values of all propositions and clock inequalities used in a parametric analysis problem, we can define the mapping from state predicates to static state predicates through a region. Formally, given a region v and a state predicate η, we write $v(\eta)$ for the static predicate constructed according to the following rules.

- $v(\mathit{false})$ is *false*.
- $v(p)$ is *true* iff $\forall s \in v(s(p) = \mathit{true})$.
- $v(x - y \sim c)$ is *true* iff $\forall s \in v(s(x) - s(y) \sim c)$.
- $v(x \sim c)$ is *true* iff $\forall s \in v(s(x) \sim c)$.
- $v(\sum a_i \alpha_i \sim c)$ is $\sum a_i \alpha_i \sim c$.
- $v(\eta_1 \vee \eta_2) = v(\eta_1) \vee v(\eta_2)$.
- $v(\neg \eta_1) = \neg v(\eta_1)$.

For convenience, we let $\langle \kappa \rangle v$ be the region in a PR-graph that agrees with v in every aspect except that for all $s' \in \langle \kappa \rangle v$, $s'(\kappa) = 0$.

4.2. Conditional path analysis

To compute the parametric condition for a parametric modal formulus like $\exists \phi_1 \mathcal{U}_{\sim\theta} \phi_2$ in a region, instead of analyzing all the cycle time patterns as in [12], we can decompose the formulus into a Boolean combination of path conditions and then compute the path conditions. For example, suppose that under interpretation $\mathcal{I}$, we know there exists a path $v_1 v_2 \ldots v_n$ of time 5. Then a sufficient condition for all states in v_1 to satisfy $\exists \phi_1 \mathcal{U}_{\leq\theta} \phi_2$ is that $\mathcal{I}(\theta) \geq 5 \wedge v_n \models_{\mathcal{I}} \phi_2 \wedge \bigwedge_{1 \leq i < n} v_i \models_{\mathcal{I}} \phi_1$. Now we define our second new data structure, the *conditional path graph*, in preparation for presenting of the algorithm.

Definition 8 (Conditional path graph). Suppose we are given a region graph $G_{A:\phi} = (V, F)$ and a subformulus ϕ_1 of ϕ. The *conditional path graph* for ϕ_1, denoted $J[\phi_1](\ )$, is a mapping from $V \times V$ to the power set of conditional time expressions with the following restrictions. For all $v, v' \in V$, if $(\pi, T) \in J[\phi_1](v, v')$, then for all interpretations $\mathcal{I}$ satisfying π, $t \in T$, and $s \in v$, there is a finite s-run of time t ending at an $s' \in v'$ such that ϕ_1 is satisfied all the way through the run except at s'.

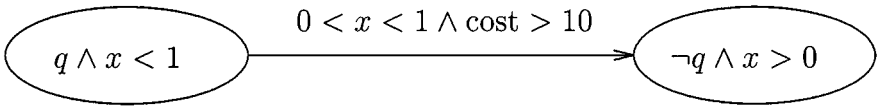

Figure 3. A simple automaton to illustrate of the algorithm.

The procedure for computing $J[\phi_1](\ )$ is presented in the following.

KClosure$[\phi_1](V, F)$
/* It is assumed that for all regions $v \in V$, we know the static state predicate condition $L[\phi_1](v)$ which satisfies ϕ_1 at v. */
{
(1) For each $(v, w) \notin F$, $J[\phi_1](v, w) := \emptyset$;
(2) For each $(v, w) \in F$, if $\epsilon(v, w) = \uparrow$, then let
$J[\phi_1](v, w) := \{(L[\phi_1](v) \wedge v(\chi(v^Q)) \wedge w(\chi(w^Q)) \wedge v(\tau(v^Q, w^Q)), 1)\}$;
else let
$J[\phi_1](v, w) := \{(L[\phi_1](v) \wedge v(\chi(v^Q)) \wedge w(\chi(w^Q)) \wedge v(\tau(v^Q, w^Q)), 0)\}$.
(3) For $i := 0$ to $|V|$, do
(1) Iteratively for each $v \in V$, do {
(1) for each $u, w \in V$, let $J[\phi_1](u, w) := J[\phi_1](u, w) \cup H[\phi_1](u, v, w)$;
}
}
}

Example (A simple automaton). In figure 3, we show a simple automaton with two meta-states. Suppose the specification in PCTL is

$$\text{cost} < 100 \wedge q \wedge x = 0 \wedge \forall\Diamond_{\leq\theta}\neg q$$

Then, the region graph is that shown in figure 4. Note that $L[q](v_0) = L[q](v_1) = \mathit{true}$ and $L[q](v_2) = L[q](v_3) = L[q](v_4) = L[q](v_5) = \mathit{false}$. After running algorithm KClosure

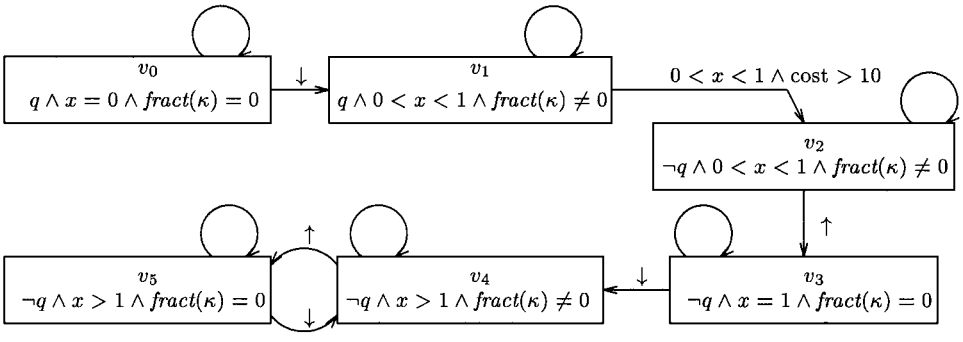

Figure 4. A simple region graph to illustrate the algorithm.

Table 1. Computation of KClosure[ ]( ).

$(true, 0) \in J[true](v_0, v_0)$	$(true, 0) \in J[true](v_1, v_1)$	
$(true, 0) \in J[true](v_2, v_2)$	$(true, 0) \in J[true](v_3, v_3)$	Row 1
$(true, 0) \in J[true](v_4, v_4)$	$(true, 0) \in J[true](v_5, v_5)$	
$(true, 0) \in J[q](v_0, v_0)$	$(true, 0) \in J[q](v_1, v_1)$	
$(false, 0) \in J[q](v_2, v_2)$	$(false, 0) \in J[q](v_3, v_3)$	Row 2
$(false, 0) \in J[q](v_4, v_4)$	$(false, 0) \in J[q](v_5, v_5)$	
$(true, 0) \in J[\neg q](v_0, v_0)$	$(true, 0) \in J[\neg q](v_1, v_1)$	
$(false, 0) \in J[\neg q](v_2, v_2)$	$(false, 0) \in J[\neg q](v_3, v_3)$	Row 3
$(false, 0) \in J[\neg q](v_4, v_4)$	$(false, 0) \in J[\neg q](v_5, v_5)$	
$(true, 0) \in J[true](v_0, v_1)$	$(\text{cost} > 10, 0) \in J[true](v_1, v_2)$	
$(true, 1) \in J[true](v_2, v_3)$	$(true, 0) \in J[true](v_3, v_4)$	Row 4
$(true, 1) \in J[true](v_4, v_5)$	$(true, 0) \in J[true](v_5, v_4)$	
$(true, 0) \in J[q](v_0, v_1)$	$(\text{cost} > 10, 0) \in J[q](v_1, v_2)$	
$(false, 1) \in J[q](v_2, v_3)$	$(false, 0) \in J[q](v_3, v_4)$	Row 5
$(false, 1) \in J[q](v_4, v_5)$	$(false, 0) \in J[q](v_5, v_4)$	
$\vdots$	$\vdots$	$\vdots$
$(\text{cost} > 10, 0) \in J[true](v_0, v_2)$	$(\text{cost} > 10, 1) \in J[true](v_1, v_3)$	
$(true, 1) \in J[true](v_2, v_4)$	$(true, 1) \in J[true](v_3, v_5)$	Row 7
$(true, 1*) \in J[true](v_4, v_4)$	$(true, 1*) \in J[true](v_5, v_5)$	
$\vdots$	$\vdots$	$\vdots$

$[q]( )$ on the region graph, we find that the computation of membership relations is that shown in Table 1. In the table, we group the formulae into rows with horizontal lines to make it more readable. The first five rows are obtained for length one paths through statement (2) in algorithm KClosure[ ]( ). The seventh row is obtained with the transitivity law through statement (3) in algorithm KClosure[ ]( ). In the last second line, because of the time 1 self-loops on regions v_4, v_5, we can deduce that $(true, 1*) \in J[true](v_4, v_4)$, which means that we can cycle through region v_4 forever.

Lemma 1 establishes the correctness of KClosure$[\phi_1]( )$.

Lemma 1. *Suppose we are given two regions v, v' in a region graph (V, F), the labeling funciton $L[\phi_1]( )$ for a PCTL formulus ϕ_1 on (V, F), an interpretation $\mathcal{I}$, and a natural number $t \in \mathcal{N}$. After running algorithm KClosure$[\phi_1](V, F)$, there is a computation from a state in v to a state in v' under interpretation $\mathcal{I}$ of t time units (κ's reading changes from noninteger to integer for t times) such that ϕ_1 is satisfied in all but the last states during the computation iff there is a $(\pi, T) \in J[\phi_1](v, v')$ such that $\mathcal{I} \models \pi$ and $t \in T$.*

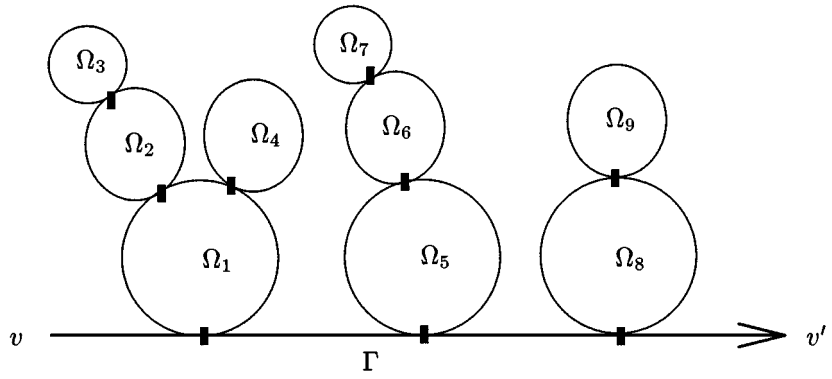

Figure 5. A typical cactus structure of path.

Proof: The proof is presented for two directions of the "iff" statement.
($\Rightarrow$) We assume there is a computation from a state in v to a state in v' under interpretation $\mathcal{I}$ with time t such that ϕ_1 is maintained until the last state. By mapping each state along the computation to its region, we can project the computation in dense time onto a finite discrete path in the region graph. According to the proof of Lemma 3 in [12], every path can be decomposed into a cactus structure of a simple path and some simple cycles, as shown in figure 5. In the cactus structure, each cycle can be traversed many times to compose the path. We can define the height of a cycle Ω, from the simple path in a cactus structure, to be the number of cycles which connect Ω to the simple path in the cactus structure. For example, in the cactus structure shown in figure 5, the heights of Ω_1, Ω_5, and Ω_8 are one, the heights of Ω_2, Ω_4, Ω_6, and Ω_9 are two, and the heights of Ω_3 and Ω_7 are three. Also, we can show that for every path, there is a way to decompose it into a cactus structure whose greatest cycle height is less than $|V|$. (Examine the connecting nodes shown as little black rectangles between cycles in figure 5. There can only be $|V|-1$ such connecting nodes. If there is a simple cycle Ω with height $>|V|$, then the sequence of connecting nodes along the simple cycles connecting Ω to Γ must have two identical connecting nodes. This repetition of connecting nodes makes it possible to collapse the height.)

We can show by induction on the heights of simple cycles in cactus structures that, after the i'th iteration of "For"-loop in KClosure[ϕ_1]( ), this direction of the lemma is true for all computation path constructed with cycles of height $\leq i$. Thus this direction of the lemma is proven.
($\Leftarrow$) This direction can be proven with an induction on i in the "For"-loop. For the induction step, we have to check that every time we perform statement $J[\phi_1](u, w) := J[\phi_1](u, w) \cup H[\phi_1](u, v, w)$; the new elements added into $J[\phi_1](u, w)$ all correspond to valid computations. This finishes the whole proof. □

4.3. Labeling algorithm

Once the conditional path graph has been constructed for ϕ_1 using KClosure[ϕ_1]( ), we can then turn to the labeling algorithm to calculate the parametric conditions for the modal

formulae properly containing ϕ_1. However, there is still one thing which we should define clearly before presenting our labeling algorithm, that is: "How should we connect the conditional time expressions (as pairs of parametric conditions and parametric semilinear expressions) in the arc labels to parametric conditions ?" Suppose, we want to examine if from v to v', there is a run with time $\geq\theta$. To do this, we define semilinear conditions in the form of $T \sim \theta$ with semilinear expressions T in PNF, and the (numerical or variable) parameter θ is calculated according to the following rewriting rules.

- $a + c* \sim \theta \Rightarrow a + cj \sim \theta$ where j is a new integer variable never used before.
- $T_1 \cup T_2 \sim \theta \Rightarrow (T_1 \sim \theta) \vee (T_2 \sim \theta)$.

Note that since we assume that the operands are in PNF, we do not have to pay attention to the case of $+, *, —$. Then, the condition that there is a run with time $\geq\theta$ from v to v' can be calculated as $\bigvee_{(\pi,T)\in J[\phi_1](v,v')} \pi \wedge T \geq \theta$.

In the following, we present the labeling algorithm for $L[\phi](v)$ in the traditional inductive case analysis of formulus ϕ.

Label(A, ϕ) {
(1) construct the PR-graph $G_{A:\phi} = (V, F)$;
(2) for each $v \in V$, recursively compute $L[\phi](v)$;
}

$L[\phi_i](v)$ {
switch(ϕ_i) {

case (*false*), $L[\,false](v) := false$;

case (p) where $p \in P$, $L^p(v) :=$ *true* if $v \models p$, else $L^p(v) :=$ *false*;

case $(x - y \sim c)$, if either x or y is zero in v, evaluate $x - y \sim c$ as in the next case;
else $x - y \sim c$ is evaluated to the same value as it is in any region u such that $(u, v) \in F$.

case $(x \sim c)$, $L[\phi_i](v) :=$ *true* if $v \models (\phi_i)$, else $L[\phi_i](v) :=$ *false*;

case $(\sum a_i\alpha_i \sim d)$, $L[\Sigma a_i\alpha_i \sim d](v) := \sum a_i\alpha_i \sim d$;

case $(\phi_j \vee \phi_k)$, $L[\phi_j \vee \phi_k](v) := L[\phi_j](v) \vee L[\phi_k](v)$;

case $(\neg\phi_j)$, $L[\neg\phi_j](v) := \neg L[\phi_j](v)$;

case $(\exists\Box_{\geq 0}\phi_j)$, {
(1) KClosure$[\phi_j](V, F)$;
(2) $L[\exists\Box_{\geq 0}\phi_j](v) := \bigvee_{u\in V} \begin{pmatrix} \left(\bigvee_{(\pi,T)\in J[\phi_j](\langle\kappa\rangle v,u)} \pi\right) \\ \wedge\left(\bigvee_{(\pi,T)\in J[\phi_j](u,u)} (\pi \wedge T > 0)\right) \end{pmatrix}$
}

case $(\exists\phi_j\mathcal{U}_{\geq\theta}\phi_k)$, {
(1) KClosure$[\phi_j](V, F)$;

(2) $L[\exists\phi_j\mathcal{U}_{\geq\theta}\phi_k](v) := \bigvee_{u\in V}\begin{pmatrix} L[\phi_k](u)\wedge L[\exists\Box_{\geq 0}true](u)\\ \wedge\bigvee_{(\pi,T)\in J[\phi_j](\langle\kappa\rangle v,u)}(\pi\wedge T\geq\theta)\end{pmatrix}$

}

case $(\exists\phi_j\mathcal{U}_{>\theta}\phi_k)$ {
(1) KClosure$[\phi_j](V,F)$;
(2) $L[\exists\phi_j\mathcal{U}_{>\theta}\phi_k](v)$ be
$\bigvee_{u\in V}\begin{pmatrix}\left(\bigvee_{(\pi,T)\in J[\phi_j](\langle\kappa\rangle v,u)}(\pi\wedge(T>\theta\vee(T=\theta\wedge u\models fract(\kappa)\neq 0)))\right)\\ \wedge L[\phi_k](u)\wedge L[\exists\Box_{\geq 0}true](u)\end{pmatrix}$

}

case $(\exists\phi_j\mathcal{U}_{\leq\theta}\phi_k)$ {
(1) KClosure$[\phi_j](V,F)$;
(2) $L[\exists\phi_j\mathcal{U}_{\leq\theta}\phi_k](v)$ be
$\bigvee_{u\in V}\begin{pmatrix}\left(\bigvee_{(\pi,T)\in J[\phi_j](\langle\kappa\rangle v,u)}(\pi\wedge(T<\theta\vee(T=\theta\wedge u\models fract(\kappa)=0)))\right)\\ \wedge L[\phi_k](u)\wedge L[\exists\Box_{\geq 0}true](u)\end{pmatrix}$

}

case $(\exists\phi_j\mathcal{U}_{<\theta}\phi_k)$ {
(1) KClosure$[\phi_j](V,F)$;
(2) $L[\exists\phi_j\mathcal{U}_{<\theta}\phi_k](v)$ be
$\bigvee_{u\in V}\left(\bigvee_{(\pi,T)\in J[\phi_j](\langle\kappa\rangle v,u)}(\pi\wedge T<\theta)\wedge L[\phi_k](u)\wedge L[\exists\Box_{\geq 0}true](u)\right)$

}

case $(\exists\phi_j\mathcal{U}_{=\theta}\phi_k)$ {
(1) KClosure$[\phi_j](V,F)$;
(2) $L[\exists\phi_j\mathcal{U}_{=\theta}\phi_k](v)$ be
$\bigvee_{u\in V}\left(\bigvee_{(\pi,T)\in J[\phi_j](\langle\kappa\rangle v,u)}\begin{pmatrix}(\pi\wedge T=\theta)\wedge u\models fract(\kappa)=0\\ \wedge L[\phi_k](u)\wedge L[\exists\Box_{\geq 0}true](u)\end{pmatrix}\right)$

}

case $(\forall\phi_j\mathcal{U}_{\leq\theta}\phi_k)$, {
(1) KClosure$[\phi_j](V,F)$;
(2) $L[\forall\phi_j\mathcal{U}_{\leq\theta}\phi_k](v) := \neg\begin{pmatrix} L[\exists(\neg\phi_k)\mathcal{U}_{\leq\theta}\neg(\phi_j\vee\phi_k)](\langle\kappa\rangle v)\\ \vee\bigvee_{u_1,u_2\in V}\begin{pmatrix} L[\neg\phi_k](u_1)\wedge\epsilon(u_1,u_2)=\downarrow\\ \wedge L[\exists\Box_{\geq 0}true](u_2)\\ \wedge\bigvee_{(\pi,T)\in J[\neg\phi_k](\langle\kappa\rangle v,u_1)}(\pi\wedge T=\theta)\end{pmatrix}\end{pmatrix}$

}

case $(\forall\phi_j\mathcal{U}_{>\theta}\phi_k)$ {
(1) KClosure$[\phi_j](V,F)$;

(2) $L[\forall\phi_j\mathcal{U}_{>\theta}\phi_k](v)$ be

$$\neg\begin{pmatrix} L[\exists\Diamond_{\leq\theta}\neg\phi_j](\langle\kappa\rangle v) \\ \vee\bigvee_{u_1,u_2\in V}\begin{pmatrix} \bigvee_{(\pi,T)\in J_{\phi_j}(\langle\kappa\rangle v,u_1)}(\pi\wedge T=\theta) \\ \wedge L[\phi_j](u_1)\wedge\epsilon(u_1,u_2)=\downarrow \\ \wedge(L[\exists\Box_{\geq 0}\neg\phi_k](u_2)\vee L[\exists(\neg\phi_k)\mathcal{U}_{\geq 0}\neg(\phi_j\vee\phi_k)](u_2))\end{pmatrix}\end{pmatrix}$$

}

case $(\forall\phi_j\mathcal{U}_{\geq\theta}\phi_k)$ {

(1) KClosure$[\phi_j](V,F)$;

(2) $L[\forall\phi_j\mathcal{U}_{\geq\theta}\phi_k](v)$ be

$$\neg\begin{pmatrix} L[\exists\Diamond_{<\theta}\neg\phi_j](\langle\kappa\rangle v) \\ \vee(\theta=0\wedge(L[\exists\Box_{\geq 0}\neg\phi_k](\langle\kappa\rangle v)\vee L[\exists(\neg\phi_k)\mathcal{U}_{\geq 0}\neg(\phi_j\vee\phi_k)](\langle\kappa\rangle v))) \\ \vee\begin{pmatrix}\theta>0\wedge\bigvee_{u_1,u_2\in V}\begin{pmatrix}\bigvee_{(\pi,T)\in J[\phi_j](\langle\kappa\rangle v,u_1)}\begin{pmatrix}\pi\wedge T=\theta-1\\ \wedge L[\phi_j](u_1)\\ \wedge\epsilon(u_1,u_2)=\uparrow\end{pmatrix} \\ \wedge\begin{pmatrix}L[\exists\Box_{\geq 0}\neg\phi_k](u_2)\\ \vee L[\exists(\neg\phi_k)\mathcal{U}_{\geq 0}\neg(\phi_j\vee\phi_k)](u_2)\end{pmatrix}\end{pmatrix}\end{pmatrix}\end{pmatrix}$$

}

case $(\forall\phi_j\mathcal{U}_{<\theta}\phi_k)$ {

(1) KClosure$[\phi_j](V,F)$;

(2) $L[\forall\phi_j\mathcal{U}_{<\theta}\phi_k](v)$ be

$$\neg\begin{pmatrix}\begin{pmatrix}\bigvee_{u_1,u_2\in V}\begin{pmatrix}L[\neg\phi_k](u_1)\wedge\epsilon(u_1,u_2)=\uparrow\wedge L[\exists\Box_{\geq 0}\mathit{true}](u_2)\\ \wedge\bigvee_{(\pi,T)\in J[\neg\phi_j](\langle\kappa\rangle v,u_1)}(\pi\wedge T=\theta-1)\end{pmatrix}\end{pmatrix} \\ \vee L[\exists(\neg\phi_k)\mathcal{U}_{<\theta}\neg(\phi_j\vee\phi_k)](\langle\kappa\rangle v)\end{pmatrix}$$

}

case $(\forall\phi_j\mathcal{U}_{=\theta}\phi_k)$ {

(1) KClosure$[\phi_j](V,F)$;

(2) $L[\forall\phi_j\mathcal{U}_{=\theta}\phi_k](v)$ be

$$\neg\begin{pmatrix} L[\exists\Diamond_{<\theta}\neg\phi_j](\langle\kappa\rangle v) \\ \vee\begin{pmatrix}\theta=0\wedge\begin{pmatrix}\bigvee_{u_1,u_2\in V}\begin{pmatrix}L[\neg\phi_k](u_1)\wedge\epsilon(u_1,u_2)=\downarrow\\ \wedge L[\exists\Box_{\geq 0}\mathit{true}](u_2)\\ \wedge\bigvee_{(\pi,T)\in J[\neg\phi_k](\langle\kappa\rangle v,u_1)}(\pi\wedge T=0)\end{pmatrix} \\ \vee L[\exists(\neg\phi_k)\mathcal{U}_{=0}\neg(\phi_j\vee\phi_k)](\langle\kappa\rangle v)\end{pmatrix}\end{pmatrix} \\ \vee\bigvee_{u_1,u_2\in V}\begin{pmatrix}\theta>0\wedge\bigvee_{(\pi,T)\in J[\phi_j](\langle\kappa\rangle v,u_1)}(\pi\wedge T=\theta-1) \\ \wedge L[\phi_j](u_1)\wedge\epsilon(u_1,u_2)=\uparrow \\ \wedge\begin{pmatrix}\bigvee_{u_3,u_4\in V}\begin{pmatrix}L[\neg\phi_k](u_3)\wedge\epsilon(u_3,u_4)=\downarrow\\ \wedge L[\exists\Box_{\geq 0}\mathit{true}](u_4)\\ \wedge\bigvee_{(\pi,T)\in J[\neg\phi_k](\langle\kappa\rangle u-2,u_3)}\pi\wedge T=0\end{pmatrix} \\ \vee L[\exists(\neg\phi_k)\mathcal{U}_{=0}\neg(\phi_j\vee\phi_k)](u_2)\end{pmatrix}\end{pmatrix}\end{pmatrix}$$

}

}

This algorithm maps pairs of vertices and temporal logic formulae to a Boolean combination of linear inequalities with parameter variables as free variables. Also note that the labeling algorithm relies on the special case of $\exists\Box_{\geq 0}\phi_j$ which essentially says that there is an infinite computation in which ϕ_j is always true. Case analysis for atoms, disjunctions, and negations are straightfoward.

Let us examine two cases to see how the algorihtm works. Formulus $\exists\Box_{\geq 0}\phi_j$ is satisfied at a state s if from s, there is computation along which ϕ_j is always true and clock κ ticks infinitely often. Projecting the computation into the PR-graph, the image of the computation forms a cycle reachable from region $\langle\kappa\rangle[s]$ ($[s]$ is the region for s in the PR-graph) such that along the path to the cycle and in the cycle, ϕ_j is true and κ ticks at least once in the cycle. In the formulus for case $\exists\Box_{\geq 0}\phi_j$:

$$L[\exists\Box_{\geq 0}\phi_j](v) := \bigvee_{u\in V}\begin{pmatrix}\left(\bigvee_{(\pi,T)\in J[\phi_j](\langle\kappa\rangle v,u)}\pi\right)\\ \wedge\left(\bigvee_{(\pi,T)\in J[\phi_j](u,u)}(\pi\wedge T>0)\right)\end{pmatrix}$$

$[s]=v$, the path from u to itself is a cycle in which κ ticks at least once ($T>0$), and the path from $\langle\kappa\rangle v$ to u characterizes the path from $\langle\kappa\rangle[s]$ to the cycle.

For all the $\forall\mathcal{U}$ cases, we find solutions using negations, formulae $\exists\Box_{\geq 0}$, and $\exists\mathcal{U}$. Let us examine case $\forall\phi_j\mathcal{U}_{>\theta}\phi_k$ carefully. It is not satisfied at a state s iff either of the following two situations hold. First, for some path from s, within θ (inclusive) time units, ϕ_j becomes false. This is handled by formulus $L[\exists\Diamond_{\leq\theta}\neg\phi_j](\langle\kappa\rangle v)$. Second, for some path from s after θ time units, either ϕ_k is never fulfilled or ϕ_j becomes false before ϕ_k becomes true. This is handled by the following formulus:

$$\bigvee_{u_1,u_2\in V}\begin{pmatrix}\bigvee_{(\pi,T)\in J[\phi_j](\langle\kappa\rangle v,u_1)}(\pi\wedge T=\theta)\\ \wedge\ L[\phi_j](u_1)\wedge\epsilon(u_1,u_2)=\downarrow\\ \wedge\ (L[\exists\Box_{\geq 0}\neg\phi_k](u_2)\vee L[\exists(\neg\phi_k)\mathcal{U}_{\geq 0}\neg(\phi_j\vee\phi_k)](u_2))\end{pmatrix}$$

Note here that u_2 is the region where we find that ϕ_k is not in time for the downfall of ϕ_j. We require that $\epsilon(u_1,u_2)=\downarrow$ to match the timing requirement of $>\theta$.

In the following example, labeling algorithm is run on a small example to present the idea.

Example (A test run of the labelling algorithm). We illustrate our algorithm on the automaton shown in figure 3 and PCTL specification

$$\text{cost}<100\wedge q\wedge x=0\wedge\forall\Diamond_{\leq\theta}\neg q$$

The region graph is shown in figure 4. Formulus $L[\text{cost}<100\wedge q\wedge x=0\wedge\forall\Diamond_{\leq\theta}\neg q](v)$, after rewriting according to the labeling algorithm, becomes

$$\begin{pmatrix}L[\text{cost}<100](v)\\ \wedge L[q](v)\\ \wedge L[x=0](v)\end{pmatrix}\wedge\neg\begin{pmatrix}L[\exists q\mathcal{U}_{\leq\theta}\neg(true\vee\neg q)](\langle\kappa\rangle v)\\ \vee\bigvee_{u_1,u_2\in V}\begin{pmatrix}L[q](u_1)\wedge\epsilon(u_1,u_2)=\downarrow\\ \wedge L[\exists\Box_{\geq 0}true](u_2)\\ \wedge\bigvee_{(\pi,T)\in J[q](\langle\kappa\rangle v,u_1)}(\pi\wedge T=\theta)\end{pmatrix}\end{pmatrix}$$

It is apparent that only region v_0 satisfies $q \wedge x = 0$. Also, $\exists q \mathcal{U}_{\leq\theta} \neg(true \vee \neg q)$ can be simplified to $\exists q \mathcal{U}_{\leq\theta} false$, which is *false* no matter what value θ has. We can simplify this formulus using $v = v_0$:

$$\begin{pmatrix} L[\text{cost} < 100](v_0) \\ \wedge\ L[q](v_0) \\ \wedge\ L[x = 0](v_0) \end{pmatrix} \wedge \neg \left(\begin{array}{l} L[\exists q \mathcal{U}_{\leq\theta} \neg(true \vee \neg q)](\langle\kappa\rangle v_0) \\ \vee \bigvee_{u_1,u_2 \in V} \begin{pmatrix} L[q](u_1) \wedge \epsilon(u_1, u_2) = \downarrow \\ \wedge\ L[\exists \Box_{\geq 0} true](u_2) \\ \wedge \bigvee_{(\pi,T) \in J[q](\langle\kappa\rangle v_0, u_1)} (\pi \wedge T = \theta) \end{pmatrix} \end{array} \right)$$

$$\equiv \begin{pmatrix} \text{cost} < 100 \\ \wedge\ true \\ \wedge\ true \end{pmatrix} \wedge \neg \left(\begin{array}{l} false \\ \vee \bigvee_{u_1,u_2 \in V} \begin{pmatrix} L[q](u_1) \wedge \epsilon(u_1, u_2) = \downarrow \\ \wedge\ L[\exists \Box_{\geq 0} true](u_2) \\ \wedge \bigvee_{(\pi,T) \in J[q](\langle\kappa\rangle v_0, u_1)} (\pi \wedge T = \theta) \end{pmatrix} \end{array} \right)$$

$$\equiv \text{cost} < 100 \wedge \neg \bigvee_{u_1,u_2 \in V} \begin{pmatrix} L[q](u_1) \wedge \epsilon(u_1, u_2) = \downarrow \\ \wedge\ L[\exists \Box_{\geq 0} true](u_2) \\ \wedge \bigvee_{(\pi,T) \in J[q](\langle\kappa\rangle v_0, u_1)} (\pi \wedge T = \theta) \end{pmatrix}$$

Now we can examine the outer quantification $\bigvee_{u_1,u_2 \in V} \ldots$ which requires that $u_1 \models q$ and $\epsilon(u_1, u_2) = \downarrow$. From the region graph in figure 4, it is apparent that u_1 must be v_0 and that u_2 must be v_1 to make the quantification true. If we follow the labeling algorithm $L(\phi, v)$, we shall find that $L[\exists \Box_{\geq 0} true](v_1) = \text{cost} > 10$. Thus, the manipulation continues as in the following:

$$\text{cost} < 100 \wedge \neg \bigvee_{u_1,u_2 \in V} \begin{pmatrix} L[q](u_1) \wedge \epsilon(u_1, u_2) = \downarrow \\ \wedge\ L[\exists \Box_{\geq 0} true](u_2) \\ \wedge \bigvee_{(\pi,T) \in J[q](\langle\kappa\rangle v_0, u_1)} (\pi \wedge T = \theta) \end{pmatrix}$$

$$\equiv \text{cost} < 100 \wedge \neg \begin{pmatrix} L[q](v_0) \wedge \epsilon(v_0, v_1) = \downarrow \\ \wedge\ L[\exists \Box_{\geq 0} true](v_1) \\ \wedge \bigvee_{(\pi,T) \in J[q](\langle\kappa\rangle v_0, v_0)} (\pi \wedge T = \theta) \end{pmatrix}$$

$$\equiv \text{cost} < 100 \wedge \neg\big(true \wedge true \wedge \text{cost} > 10 \wedge \bigvee_{(\pi,T) \in J[q](\langle\kappa\rangle v_0, v_0)} (\pi \wedge T = \theta)\big)$$

$$\equiv \text{cost} < 100 \wedge \neg\big(\text{cost} > 10 \wedge \bigvee_{(\pi,T) \in J[q](\langle\kappa\rangle v_0, v_0)} (\pi \wedge T = \theta)\big)$$

Running KClosure$[q](v_0, v_0)$ (note $v_0 = \langle\kappa\rangle v_0$), we find that $J[q](v_0, v_0) = \{(true, 0)\}$. Thus, our simplification continues:

$$\begin{aligned} &\text{cost} < 100 \wedge \neg\big(\text{cost} > 10 \wedge \bigvee_{(\pi,T) \in J[q](\langle\kappa\rangle v_0, v_0)} (\pi \wedge T = \theta)\big) \\ &\quad \equiv \text{cost} < 100 \wedge \neg\,(\text{cost} > 10 \wedge (true \wedge 0 = \theta)) \\ &\quad \equiv \text{cost} < 100 \wedge (\text{cost} \leq 10 \vee 0 \neq \theta) \\ &\quad \equiv \text{cost} \leq 10 \vee (\text{cost} < 100 \wedge 0 \neq \theta) \end{aligned}$$

Since the domain of θ is $\mathcal{N}$, $0 \neq \theta$ is equivalent to $\theta \geq 1$. Let us interpret the result formulus.

- When cost $\leq$10 is required, according to figure 3, there will be no computation, and the universal quantification on paths will be automatically satisfied.
- When cost $<100 \wedge \theta \geq 1$ is required, two cases need to be considered:
  - When cost $\leq$10, the specification is automatically satisfied for the above-mentioned reason under the last bullet.
  - When cost $>$10, the sole transition will be triggered and will assert $\neg q$ within one time unit to satisfy the specification.

The following theorem establishes the correctness of our labeling algorithm.

Theorem 1. *Given* PAP(A, ϕ)*, an interpretation* $\mathcal{I}$ *for* H*, and a state* s (*with* $[s]$ *as a region in* $G_{A:\phi}$)*, after executing* $L[\phi_i](v)$ *in our labeling algorithm,* $\mathcal{I}$ *satisfies* $L[\phi_i]([s])$ *iff* $s \models_{\mathcal{I}} \phi_i$.

Proof: Conveniently, a path $v_1 v_2 \ldots v_n$ in the region graph is called a ϕ_j-path with interpretaton $\mathcal{I}$ iff for all $1 \leq i < n$ and $s \in v_i$, $s \models_{\mathcal{I}} \phi_j$. The *time* of a path is the number of arcs along the path with an $\uparrow$ label.

With structural induction on ϕ_i in both directions, we first assume that $\mathcal{I}$ satisfies $L[\phi_i]([s])$ and want to show that $s \models_{\mathcal{I}} \phi_i$. We will prove this by induction on the structure of ϕ_i.

1. The case where ϕ_i is an atomic state predicate is trivial and serves as the assumption basis of induction.
2. Suppose ϕ_i is $\phi_j \vee \phi_k$. By the assumption, $\mathcal{I}$ satisfies $L[\phi_j]([s]) \vee L[\phi_k]([s])$. According to the labeling algorithm, $L[\phi_j \vee \phi_k]([s]) = L[\phi_j]([s]) \vee L[\phi_k]([s])$. According to the inductive hypothesis, we then have either $s \models_{\mathcal{I}} \phi_j$ or $s \models_{\mathcal{I}} \phi_k$, which in turn means that $s \models_{\mathcal{I}} (\phi_j \vee \phi_k)$.
3. Suppose ϕ_i is $\neg\phi_j$. According to our labeling algorithm, $L[\neg\phi_j]([s]) = \neg L[\phi_j]([s])$. Thus, that $\mathcal{I}$ satisfies $L[\neg\phi_j]([s])$ means that $\mathcal{I}$ does not satisfy $L[\phi_j]([s])$. According to the inductive hypothesis, we then have $s \not\models_{\mathcal{I}} \phi_j$, which in turn means $s \models_{\mathcal{I}} \neg\phi_j$.
4. Suppose ϕ_i is $\exists\Box_{\geq 0}\phi_j$. According to our labeling algorithm, satisfaction of condition $L[\exists\Box_{\geq 0}\phi_j]([s])$ by $\mathcal{I}$ means that there is a ϕ_j-cycle of positive time accessible from region u through a ϕ_j-path from $\langle\kappa\rangle[s]$ to u. Note that u can denote all four types of intervals (closed or open on the left or right) since the regions can record if a clock's reading is an integer or not. According to the construction of the CR-graph and our inductive hypothesis, this means that $s \models_{\mathcal{I}} \exists\Box_{\geq 0}\phi_j$ is true, and the case is proven.
5. Suppose ϕ_i is $\exists\phi_j \mathcal{U}_{\geq\theta}\phi_k$. When $\mathcal{I}$ satisfies the condition $L[\exists\phi_j \mathcal{U}_{\geq\theta}\phi_k]([s])$, there exists a ϕ_j-path from $\langle\kappa\rangle[s]$ to u of time $\sim\mathcal{I}(\theta)$, ϕ_k is satisfied at u, and $\exists\Box_{\geq 0}$*true* is true at u. According to the construction of the CR-graph and our inductive hypothesis, this means that $s \models_{\mathcal{I}} \exists\phi_j \mathcal{U}_{\geq\mathcal{I}(\theta)}\phi_k$.

   One special thing to check is the precise timing requirement $\geq\theta$ in $\exists\phi_j \mathcal{U}_{\geq\theta}\phi_k$ for dense-time computation. (Please see page 6) This presents no problem since our regions record whether each clock (including κ) has an integer reading or a noninteger one.
6. The other cases of $\exists\mathcal{U}_{\sim\theta}$ can be proven much as we have done in case 5.

7. Suppose ϕ_i is $\forall\phi_j\mathcal{U}_{\leq\theta}\phi_k$. We work on the negation instead. $\forall\phi_j\mathcal{U}_{\leq\theta}\phi_k$ is false exactly when one of the following two conditions occurs.

   – for some path, no later than θ time units, ϕ_k is never true;
   – for some path, no later than θ time units, ϕ_j becomes false before ϕ_k becomes true.

   Segmenting these paths into regions in CR-graphs, we find that the falsity of $L[\forall\phi_j\mathcal{U}_{\leq\theta}\phi_k]([s])$ leads to the satisfaction of the disjunction of these two conditions.

   The precise timing requirement $\sim\theta$ in $\forall\phi_j\mathcal{U}_{\sim\theta}\phi_k$ for dense-time computation can also be proven much as we have done in case 5.
8. The other cases of $\forall\mathcal{U}_{\sim\theta}$ can be proven much as we have done in case 7.

We will next assume that $s \models_{\mathcal{I}} \phi_i$ and show that $\mathcal{I}$ satisfies $L[\phi_i]([s])$. Again, we can do this by induction on the structure of ϕ_i

1. The case where ϕ_i is an atomic state predicate is trivial and serves as the assumption basis of induction.
2. Suppose ϕ_i is $\phi_j \vee \phi_k$. According to our assumption, $s \models_{\mathcal{I}} \phi_j \vee \phi_k$, which in turn means that $s \models_{\mathcal{I}} \phi_j \vee s \models_{\mathcal{I}} \phi_k$. Accroding to the inductive hypothesis, we then have that either $\mathcal{I}$ satisfies $L[\phi_j]([s])$ or $\mathcal{I}$ satisfies $L[\phi_k]([s])$. But our labeling algorithm calculates $L[\phi_j \vee \phi_k]([s])$ as $L[\phi_j]([s]) \vee L[\phi_k]([s])$. Thus, the case is proven.
3. Suppose ϕ_i is $\neg\phi_j$. According to our assumption, $s \models_{\mathcal{I}} \neg\phi_j$, which in turn means that $s \not\models \phi_j$. According to the inductive hypothesis, we then have that $\mathcal{I}$ does not satisfy $L[\phi_j]([s])$. Then, our labeling algorithm calculates $L[\neg\phi_j]([s])$ as the negation of $L[\phi_j]([s])$.
4. Suppose ϕ_i is $\exists\Box_{\geq 0}\phi_j$. According to the semantics of $\exists\Box_{\geq 0}$, there is an s-run in which $\phi_j^{\mathcal{I}}$ is always true. We can segment the s-run into regions to produce a path in the CR-graph. By the inductive hypothesis, all the regions along the path must satisfy ϕ_j. Examining the formulus constructed correspondingly in our labeling algorihtm, we find that it must also be satisfied. This means that $\mathcal{I}$ satisfies $L[\phi_i]([s])$.
5. Suppose ϕ_i is $\exists\phi_j\mathcal{U}_{\geq\theta}\phi_k$. When $s \models_{\mathcal{I}} \phi_i$ is true, there is an s-run through s_1 such that

   – it takes $\sim\mathcal{I}(\theta)$ time units to go from s to s_1,
   – $\phi_j^{\mathcal{I}}$ is true from s to just before s_1, and
   – ϕ_k is satisfied at s_1 with $\mathcal{I}$.

   This in turn means that there is a path in the CR-graph such that $\mathcal{I}$ satisfies $L[\phi_i]([s])$ according to our inductive hypothesis and CR-graph construction.

   The precise timing requirement $\sim\theta$ in $\forall\phi_j\mathcal{U}_{\sim\theta}\phi_k$ for dense-time computation presents no problem since our regions record whether each clock (including κ) has an integer reading or a noninteger one.
6. The other cases of $\exists\mathcal{U}_{\sim\theta}$ can be proven much as we have done in case 5.
7. Suppose ϕ_i is $\forall\phi_j\mathcal{U}_{\leq\theta}\phi_k$. We work on the negation instead. $\forall\phi_j\mathcal{U}_{\leq\theta}\phi_k$ is false exactly when one of the following three conditions occurs.

   – for some path, before θ time units, ϕ_k is never true;
   – for some path, before θ time units, ϕ_j becomes false before ϕ_k becomes true.

Mapping these two conditions to path conditions on CR-graphs, we find that their satisfaction leads to the falsity of $L[\forall\phi_j\mathcal{U}_{\leq\theta}\phi_k]([s])$. The precise timing requirement $\sim\theta$ in $\forall\phi_j\mathcal{U}_{\sim\theta}\phi_k$ for dense-time computation can also be proven much as we have done in 5.

8. The other cases of $\forall\mathcal{U}_{\sim\theta}$ can be proven much as we have done in case 7.

This ends our proof. □

4.4. Complexity

According to our construction, the number of regions in $G_{A:\phi}$, denoted $|G_{A:\phi}|$, is at most $3|Q|\cdot(K_{A:\phi}+1)^C\cdot(|C|+1)!$, where the coefficient 3 and constant $+1$ reflect the introduction of the ticking indicator κ. The inner loop of KClosure$[\phi_1]($ $)$ will be executed $|G_{A:\phi}|^4$ times. Each iteration takes an amount of time proportional to $|J[\phi_1](u,v)||J[\phi_1](v,w)|2^{|J[\phi_1](v,v)|}$. The conditional path graph arc labels, i.e., $J[\phi_1](u,v)$, roughly correspond to the set of simple paths from u to v although they utilize the succinct representation of semilinear expressions. Thus, according to the complexity analysis in [12], we find that the procedure KClosure$[\phi_1]($ $)$ has complexity doubly exponential to the size of $G_{A:\phi}$ and, thus, triply exponential to the size of the input, assuming a constant time for the manipulation of semilinear expressions.

We will now analyze the complexity of our labeling procedure. Procedure $L($ $)$ invokes KClosure$[\phi_j]($ $)$ at most once. Label(A,ϕ) invokes $L($ $)$ at most $|G_A||\phi|$ times. Thus, the complexity of the algorithm is roughly triply exponential to the size of A and ϕ since polynomials of exponentialities are still exponentialities.

Finally, the PCTL satisfiability problem is undecidable since it is not easier than the TCTL satisfiability problem [1].

5. Conclusion

With the success of CTL-based techniques in automatic verification for computer systems [5, 6, 11], it will be helpful if a formal theory suitable for common real-world projects can be developed. We feel hopeful that the insight obtained and techniques used in this study can be further applied to verification of reactive systems in a more natural and productive way.

Acknowledgments

The authors would like to thank Prof. Tom Henzinger. His suggestion that I use dynamic programming to solve the PTCTL parametric timing analysis problem [12] triggered this research. Also, special thanks go to Dr. Pao-Ann Hsiung who discussed ideas with me and reviewed a preliminary draft of the paper.

Note

1. A semilinear integer set is expressible as the union of a finite number of integer sets like $\{a+b_1j_1+\cdots+b_nj_n \mid j_1,\ldots,j_n\in\mathcal{N}\}$ for some $a,b_1,\ldots,b_n\in\mathcal{N}$.

References

1. R. Alur, C. Courcoubetis, and D.L. Dill, "Model-checking in dense real-time," *Information and Computation* Vol. 104, No. 1, pp. 2–34, 1993.
2. R. Alur and D. Dill, "Automata for modeling real-time systems," in *Automata, Languages and Programming: Proceedings of the 17th ICALP*, 1990. Lecture Notes in Computer Science, Vol. 443, Springer-Verlag, Berlin/New York, pp. 332–335.
3. R. Alur and T.A. Henzinger, "Real-time logics: Complexity and expressiveness," in *Proceedings, 5th IEEE LICS*, 1990, pp. 390–401.
4. R. Alur, T.A. Henzinger, and M.Y. Vardi, "Parametric real-time reasoning," in *Proceedings, 25th ACM STOC*, 1993, pp. 592–601.
5. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang, "Symbolic model checking: 10^{20} states and beyond," in *IEEE LICS*, 1990, pp. 428–439.
6. R.E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, Vol. C-35, No. 8, pp. 677–691, 1986.
7. E. Clarke and E.A. Emerson, "Design and synthesis of synchronization skeletons using branching-time temporal logic," in *Proceedings, Workshop on Logic of Programs*, 1981. LNCS, Vol. 131, Springer-Verlag, pp. 52–71.
8. E. Clarke, E.A. Emerson, and A.P. Sistla, "Automatic verification of finite-state concurrent systems using temporal-logic specifications," *ACM Trans. Programming, Languages, and Systems*, Vol. 8, No. 2, pp. 244–263, 1986.
9. C. Courcoubetis and M. Yannakakis, "Minimum and maximum delay problems in real-time systems," *Formal Methods in System Design*, Vol. 1, pp. 385–415, 1992. Also in Proceedings, 3rd CAV, 1991, Springer-Verlag, LNCS, Vol. 575.
10. T.A. Henzinger, P.-H. Ho, and H. Wong-Toi, "HyTech: The next generation," in *Proceedings of the 16th Annual Real-Time System Symposium*, IEEE Computer Society Press, 1995, pp. 56–65.
11. T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic model checking for real-time systems," in *IEEE LICS*, 1992, pp. 394–406.
12. F. Wang, "Parametric timing analysis for real-time systems," *Information and Computation*, Vol. 130, No. 2, Academic Press, 1996, ISSN 0890-5401, pp. 131–150. Also in Proceedings, 10th IEEE Symposium on Logic in Computer Science, 1995.