Synthesis of Parametric Embedded Real-Time Systems

Pao-Ann Hsiung
Institute of Information Science, Academia Sinica, Taipei, Taiwan, ROC.
E-mail: hpa@computer.org

Abstract

Synthesis means restricting a specification behavior such that the synthesized behavior satisfies some given property. Previous work on the synthesis of embedded real-time system have not considered *parameters*. In the world of embedded systems, parameters are of paramount importance for system design because tradeoffs among various system features and functionalities have to be made. We define and solve the *parametric embedded real-time system synthesis* problem by integrating parametric analysis techniques into embedded real-time system synthesis. An example is given to illustrate the feasibility of our approach.
Keywords: embedded real-time system synthesis, parametric analysis, parametric synthesis, formal design

1 Introduction

Currently, a technological gap exists between system analyzers and system designers because the former do not take real-world constraints into consideration while verifying systems and the latter often lack preciseness and completeness in designing systems. Researchers are trying to bridge this gap through formal synthesis techniques [4, 5, 6, 11] and through practical analysis techniques [2, 7, 8, 10, 9, 15]. Our effort at incorporating system parameters into embedded real-time system synthesis is also directed towards this goal. On one hand, parametric analysis techniques have been proposed for real-time systems [15]. On the other hand, synthesis methods have been proposed for real-time systems [5, 12, 16] through extensions of Ramadge & Wonham's classical work [13] on supervisor synthesis. Our work consists of integrating these two classes of techniques such that embedded real-time system can be synthesized for systems with static parameters.

In general, system parameters include system component costs and performance readings such as response time, throughput, utilization, power (percentage throughput per unit utilization), fault-tolerance, scalability, reliability, and others. These parameters become all the more important in embedded systems, because the behavior of an embedded system is constantly under the influence of an environment. Synthesis that does not consider the values of such parameters may not give the best system design result.

Parametric analysis of real-time systems solves the problem of finding a general relationship that valuations for a set of parameters must satisfy in order for a system to meet a given property [15]. Parameters appear in a system description and as a quantification suffix in a property specification. Previous work [15] first modeled a system by *Parametric Timed Automata* (PTA), specified a system property by *parametric timed computational tree logic* (PTCTL), and then derived a condition on parameter valuations, which is a boolean combination of semi-linear expressions.

Synthesis for embedded control systems was mainly performed in the discrete time domain, with a large portion of classical work done by Ramadge and Wonham [13, 14]. Around 1994, when timed automata was proposed as a dense-time model for real-time systems [3], discrete synthesis was extended to dense real-time systems [4, 12, 17] as well as to hybrid systems [16]. Recently, the same technique was further extended to multimedia scheduler synthesis [1]. Given a dense real-time system modeled by timed automata and a (temporal) property given as a formula in *Timed Computation Tree Logic* (TCTL) [2, 8], a restriction of system behavior is synthesized for satisfying the property. This is called *embedded real-time system* (ERTS) synthesis problem.

In previous work on synthesis [1, 4, 5, 12], discrete variables and constants appear in both system description and property specification, but parameters were not allowed. We extend the existing work by allowing parameters to appear in system description and property specification such that embedded real-time system can be synthesized for parametric systems, too.

This article is organized as follows. Section 2 formulates and models the *Parametric Embedded Real-Time Systems* (PERTS) synthesis problem. Section 3 proposes a method for solving PERTS synthesis problem. Section 4 illustrates the proposed method through an application example. Section 5 concludes the article.

2 Problem Formulation

The problem that we are modeling and solving here consists of two main issues:

1. **Parameter Valuation**: Given a system, a specification, and a set of parameters, find a general condition on the parameter values such that the system satisfies the specification.

2. **Embedded Real-Time System Synthesis**: Given a system, a synthesis mask, and a specification, find a restriction of the system behavior, interpreted under the given mask, such that it satisfies the specification.

Given a system, a synthesis mask, a specification, and a set of parameters, both of the above two issues must be solved. A synthesis mask is a boolean vector that specifies which transitions are synthesizable and which are not. These two issues may at first appear to be orthogonal problems due to the former being a static problem (parameter values do not change with time), while the latter a dynamic problem (restricts the dynamic behavior of a system). But, they are in fact intricately involved with each other. The following are two ways in which they affect each other.

- *Static affects dynamic*: Corresponding to different static parameter valuations, a system description will have different interpretations, which results in different behavior synthesized.

- *Dynamic affects static*: When the dynamic behavior of a system is restricted in some way, certain static parameter valuations that worked originally may no longer work now, where *work* means allowing a system to satisfy a given property specification.

Due to the above relationships between the two issues, we need to find the *most general condition* on parameter valuations while imposing the *minimum restriction* on a system behavior to satisfy a given specification. Before formulating our problem, some models and definitions are necessary.

Our target is a *real-time system*, that is, a system whose correctness depends on satisfying certain temporal constraints. Here, a real-time system is formally modeled by *Parametric Timed Automaton* (PTA) model [15]. The PTA model basically extends *Timed Automata* (TA) [3] with parameters in the system description. Timed automata are automata with time or clock variables, which model temporal characteristics and temporal specifications. In the following, the set of integers and non-negative real numbers are denoted by $\mathcal{N}$ and $\mathcal{R}_{\geq 0}$, respectively. *Mode predicates* are used to specify transition triggering conditions and mode invariants and are defined as follows.

Definition 1 *:* **Mode Predicate**
Given a set C of clock variables, a set D of discrete variables, and a set H of parameters, the syntax of a *mode predicate* η over C, D, and H is defined as: $\eta := \mathit{false} \mid x \sim c \mid x - y \sim c \mid d \sim c \mid \Sigma_i a_i \alpha_i \sim c \mid \eta_1 \wedge \eta_2 \mid \neg\eta_1$, where $x, y \in C$, $\sim \in \{\leq, <, =, \geq, >\}$, $c, a_i \in \mathcal{N}$, $\alpha_i \in H$, $d \in D$, and η_1, η_2 are mode predicates.

Let $B(C, D, H)$ be the set of all mode predicates over C, D, and H. A PTA is composed of various *modes* interconnected by *transitions*. Variables are distinguished into *clock* and *discrete*, where variables of the former type increment at a uniform rate and can be reset on a transition, while variables of the latter type change values only when assigned a new value on a transition.

Definition 2 *:* **Parametric Timed Automaton**
A *Parametric Timed Automaton* (PTA) is a tuple $\mathcal{A} = (M, m^0, C, D, H, \chi, E, \tau, \rho)$ such that: M is a finite set of modes, $m^0 \in M$ is the initial mode, C is a set of clock variables, D is a set of discrete variables, H is a set of static parameters, $\chi : M \mapsto B(C, D, H)$ is an *invariance* function that labels each mode with a condition true in that mode, $E \subseteq M \times M$ is a set of transitions, $\tau : E \mapsto B(C, D, H)$ defines the transition triggering conditions, and $\rho : E \mapsto 2^{C \cup (D \times \mathcal{N})}$ is an *assignment* function that maps each transition to a set of assignments such as resetting some clock variables and setting some discrete variables to specific integer values.

Definition 3 *:* **Interpretation**
An *interpretation*, $\mathcal{I}$, of a PTA $\mathcal{A} = (M, m^0, C, D, H, \chi, E, \tau, \rho)$ is a mapping from the set of parameters H to the set of integers $\mathcal{N}$. A PTA $\mathcal{A}$ interpreted under an interpretation $\mathcal{I}$ will be denoted as $\mathcal{A}|_{\mathcal{I}}$.

Definition 4 *:* **State**
Given a PTA $\mathcal{A} = (M, m^0, C, D, H, \chi, E, \tau, \rho)$, a *state* s of $\mathcal{A}$ is defined as a mapping from $C \cup D$ to $\mathcal{R}_{\geq 0} \cup \mathcal{N}$ such that

- $\forall x \in C, s(x) \in \mathcal{R}_{\geq 0}$ is the reading of clock x in s, and
- $\forall d \in D, s(d) \in \mathcal{N}$ is the value of d in s.

Definition 5 *:* **Satisfaction of Mode Predicates**
The *satisfaction* of mode predicates by a state s under interpretation $\mathcal{I}$, written as $s \models_{\mathcal{I}} \eta$, is defined by the following rules:

- $s \not\models_{\mathcal{I}} \mathit{false}$;
- $s \models_{\mathcal{I}} x - y \sim c$ iff $s(x) - s(y) \sim c$;
- $s \models_{\mathcal{I}} x \sim c$ iff $s(x) \sim c$;
- $s \models_{\mathcal{I}} d \sim c$ iff $s(d) \sim c$;
- $s \models_{\mathcal{I}} \sum a_i \alpha_i \sim c$ 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 $s \not\models_{\mathcal{I}} \eta_1$.

Given a PTA $\mathcal{A} = (M, m^0, C, D, H, \chi, E, \tau, \rho)$, an interpretation $\mathcal{I}$ for H, and a state s, let s^M be the mode in M such that $s \models_{\mathcal{I}} \chi(s^M)$. If there is no mode $m \in M$ such that $s \models_{\mathcal{I}} \chi(m)$, then s^M is undefined.

Definition 6 *:* **Mode Transition**

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

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

Also, given a state s and a $\delta \in \mathcal{R}_{\geq 0}$, let $s + \delta$ be the state that agrees with s in every aspect except for all $x \in C$, $s(x) + \delta = (s + \delta)(x)$.

Definition 7 *:* **Interpreted PTA**

A PTA $\mathcal{A} = (M, m^0, C, D, H, \chi, E, \tau, \rho)$ is said to be interpreted under some interpretation $\mathcal{I}$, when all the mode invariants and trigerring conditions on transitions have their parameters interpreted under $\mathcal{I}$, i.e., $\forall \alpha \in H$, α is replaced by $\mathcal{I}(\alpha) \in \mathcal{N}$.

When $\mathcal{A}$ is interpreted under $\mathcal{I}$, it is denoted by $\mathcal{A}|_{\mathcal{I}}$. Let $\mathcal{V}_{H,\phi}$ be the set of all possible valuations of the parameters in H and of those appearing in a PTCTL formula ϕ.

Definition 8 *:* **A Feasible s-run**

Given a state s of PTA $\mathcal{A} = (M, m^0, C, D, H, \chi, E, \tau, \rho)$ under interpretation $\mathcal{I}$, a computation of $\mathcal{A}$ starting at s is called a *feasible s-run* which can be represented by an infinite sequence $((s_1, t_1), (s_2, t_2), \ldots\ldots)$ such that

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

We denote by $Run(\mathcal{A}|_{\mathcal{I}})$ the set of all feasible s-runs of a PTA $\mathcal{A}$ interpreted under $\mathcal{I}$.

Timed Computation Tree Logic (TCTL) [8] is extended to include static parameters in the automata and parameters in TCTL formula, we call this extension *Parametric Timed Computation Tree Logic* (PTCTL). The syntax of a PTCTL formula, ϕ, used for analyzing models described by a PTA $\mathcal{A} = (M, m^0, C, D, H, \chi, E, \tau, \rho)$ is defined as follows.

Definition 9 *:* **Parametric TCTL (PTCTL)**

A PTCTL formula ϕ is defined to have the following syntax.

$$\phi ::= \eta \mid \exists \phi_1 \mathcal{U}_{\sim\theta} \phi_2 \mid \forall \phi_1 \mathcal{U}_{\sim\theta} \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg \phi_1 \quad (1)$$

where η is a mode predicate, ϕ_1, ϕ_2 are PTCTL formulae and θ is either an integer constant in $\mathcal{N}$ or a parameter in H. Note that the parameter subscripts of modal formulae can also be used as parameters in PTA.

Definition 10 *:* **Synthesis Mask**

A *synthesis mask* μ_c maps each transition of a PTA into a boolean value, such that *true* (1) denotes a synthesizable transition, and *false* (0) denotes an unsynthesizable transition.

$$\mu_c : E \rightarrow \{0, 1\} \quad (2)$$

where E is a set of transitions of some given PTA. Often, a synthesis mask is simply given as a boolean vector, $\mu_c(E) = \langle \mu_c(e_0), \mu_c(e_1), \ldots, \mu_c(e_{|E|}) \rangle$, assuming some sequential order for all the transitions in a PTA.

Given the above formal definitions and notations, our target problem is thus formulated as follows.

Definition 11 *:* **PERTS Synthesis** (**PSynth**(S, μ_c, ϕ)): Given a real-time system modeled by a parametric timed automaton S, a synthesis mask μ_c, and a parametric timed computation tree logic formula ϕ, the **parametric embedded real-time system synthesis** problem is to find a general condition V_ϕ on the parameter valuations and a synthesized parametric timed automaton S_C such that the following conditions are satisfied.

1. the synthesized system S_C, interpreted under V_ϕ, satisfies ϕ, that is, $S_C \models_{V_\phi} \phi$, and
2. there is no state $s \in Q_S$ such that s is reachable in $S|_v$, $s \notin Q_{S_C}$ and $s \models_v \phi$ for some parameter valuation $v \in \mathcal{V}_{H,\phi}$, where $S|_v$ is the interpretation of S under v, Q_S and Q_{S_C} are the state-spaces of S and S_C, respectively.

3 PERTS Synthesis

As mentioned at the start of Section 2, we have two goals in solving the parametric controller synthesis problem: (1) derive most general conditions on parameter valuations, and (2) construct a minimally restricted synthesized system satisfying a given property. These two goals are inter-related. To achieve these two goals simultaneously, we propose an integrated approach, where the iterative algorithm of embedded real-time system synthesis is the basis and parametric analysis is performed on-the-fly.

3.1 Parametric Analysis

Parametric analysis is a method for finding a general condition on the valuation of parameters appearing in a system and in a specification such that the given system satisfies the given specification [15]. Given a system represented by a PTA $S = \langle M, m_0, E, X, Y, H, L, \chi, \tau, \rho, \psi \rangle$

Table 1: Labeling Algorithm for Parametric Analysis

PAnalyze(S, ϕ)
PTA $S = \langle M, m_0, E, X, Y, H, L, \chi, \tau, \rho, \psi \rangle$;
PTCTL Formula ϕ;
{
  Construct Param. Region Graph $G_{S:\phi} = (R, F)$; (1)
  for each $r \in R$, recursively compute label $L^{\phi}(r)$; (2)
  return $\bigvee_{r_0 \in R_0} L^{\phi}(r_0)$; (3)
  // $R_0 \subseteq R$ is a set of all initial regions in R
}

and a PTCTL specification ϕ, parametric analysis finds a condition $V_{H,\phi}$ on the valuation of parameters in H and of those appearing in ϕ such that $S \models_{V_{H,\phi}} \phi$. The condition $V_{H,\phi}$ is a boolean combination of semi-linear expressions, where each semi-linear expression gives the pattern of values that a parameter could take. A semi-linear expression is a union of a finite number of integer sets like $\{a + b_1 j_1 + b_2 j_2 + \ldots + b_n j_n \mid j_1, \ldots, j_n \in \mathcal{N}\}$ for some $a, b_1, b_2, \ldots, b_n \in \mathcal{N}$.

Parametric analysis is given briefly in Table 1 as a function **PAnalyze**(). The basic steps are explained here, but details can be found in [15]. First, parametric analysis constructs a *parametric region graph*, which is a parametric extension of region graphs proposed by Alur et al. [3]. All the states in a region have the same truth value with respect to model checking and hence can be taken as the basis of parametric analysis. Second, a *conditional path graph* is constructed for each pair of regions in the parametric region graph. A conditional path graph for a pair of regions represents a predicate condition and a semi-linear expression such that a computation run along that path must satisfy the predicate condition and takes time as recorded by the semi-linear expression. Third, through a labeling algorithm each region r is associated with a label $L^{\phi}(r)$ representing $S, r \models_{L^{\phi}(r)} \phi$. The labels are computed recursively. Finally, a disjunction of labels associated with all initial regions is taken as the general parametric condition for the satisfaction of ϕ by system S.

3.2 Predicate Synthesis

Predicate synthesis modifies existing triggering condition predicates on synthesizable transitions and invariant predicates in modes of a given to-be-controlled system such that the resulting modified system satisfies a given specification. More formally, given a system represented by a PTA $S = \langle M, m_0, E, X, Y, H, L, \chi, \tau, \rho, \psi \rangle$ interpreted under a parametric valuation V, a synthesis mask μ_c, and a PTCTL specification ϕ, predicate synthesis constructs a new PTA

Table 2: System Synthesis Algorithm (□ Timed Game)

Synthesize_System$(S|_V, \mu_c, \phi)$
PTA $S = \langle M, m_0, E, X, Y, H, L, \chi, \tau, \rho, \psi \rangle$;
V is an interpretation or parameter valuation;
Synthesis Mask $\mu_c : E \to \{\mathit{true}, \mathit{false}\}$;
PTCTL formula ϕ;
{
  $z_0 = \phi$; (1)
  for $i = 0, 1, \ldots$ { (2)
    $z_{i+1} = z_i \cap \pi(z_i, \phi)$; (3)
    if $(z_{i+1} = z_i)$ **break**; } (4)
  $S_C =$ **Synthesize_Predicate**$(S|_V, \mu_c, z_i)$; (5)
  return S_C; (6)
}

z_i is a set of states, $i \geq 0$, and S_C is a PTA.

S_C by refining $\tau(e)$ for all $e \in E$ with $\mu_c(e) = \mathit{true}$, and by refining $\chi(m)$ for all $m \in M$, such that $S_C, r \models_V \phi$, for all regions r in the parametric region graph of S_C.

The algorithm **Synthesize_System**() given in Table 2 illustrates how a system PTA is *synthesized*. The basic steps are explained here, but details can be found in [4, 5]. First, a *fix-point* is obtained iteratively, which represents a closure of all the regions (collection of states) in which ϕ can be satisfied. The iteration starts with an initial region z_0 which includes all states that satisfy ϕ. In each iteration i, a predecessor operator π is applied to region z_i and a new region z_{i+1} is obtained by an intersection of z_i and $\pi(z_i, \phi)$ (for □ timed game) or by a union of z_i and $\pi(z_i, \phi)$ (for ◇ timed game). Here, a timed game is simply a PTCTL formula. The new region represents a 1-step backward reachable collection of states. Then, using the resulting fix-point z_i, predicates are synthesized by the algorithm **Synthesize_Predicate**() given in Table 3.

The predicate synthesis algorithm is given as a function **Synthesize_Predicate**() in Table 3. Here, the predicates we are concerned with are those appearing in triggering conditions on *synthesizable* transitions and those appearing in invariant conditions of modes. By *predicate synthesis*, we mean the existing predicates in a system S are to be modified into new predicates, called *synthesized predicates*, which guarantee that the modified system S_C satisfies ϕ. This algorithm uses a region, z, called the *fix-point region* which is a closure of regions that satisfy ϕ. First, for all modes m in M that has a state in z, its invariant condition $\chi(m)$ are conjuncted with z so as to restrict the states in each such mode to only those in z. Then, for all out-going synthesizable transition e of each such mode m, its triggering condition $\tau(e)$ is conjuncted with a predicate condition τ_ϕ. This predicate condition τ_ϕ ensures that for each unsyn-

Table 3: Predicate Synthesis Algorithm

Synthesize_Predicate(S, μ_c, z)
PTA $S = \langle M, m_0, E, X, Y, H, L, \chi, \tau, \rho, \psi \rangle$;
Synthesis Mask $\mu_c : E \to \{\mathit{true}, \mathit{false}\}$;
Set of regions z;
{
  for each $m \in M$ s.t. $\exists s \in z, s^M = m$ { (1)
    $\chi(m) = \chi(m) \wedge z$; (2)
    for each out-going trans e of m
      with $\mu_c(e) = \mathit{true}$ { (3)
      $\tau(e) = \tau(e) \wedge \tau'$; (4)
      where $\tau' = \forall e' \in E$
$$\left(\begin{array}{l} (\mu_c(e') = \mathit{false}) \wedge (\tau(e') \Rightarrow z) \\ \wedge\ \exists \text{ a run } \langle (s_0, t_0), (s_1, t_1), \ldots, (s_k, t_k) \rangle \\ \text{such that } e' = (s_0^M, s_1^M), e = (s_{k-1}^M, s_k^M). \end{array} \right)$$
}}
  return S; (5)
}

Table 4: PERTS Synthesis Algorithm (□ Timed Game)

PSynth(S, μ_c, ϕ)
PTA $S = \langle M, m_0, E, X, Y, H, L, \chi, \tau, \rho, \psi \rangle$;
Synthesis Mask $\mu_c : E \to \{\mathit{true}, \mathit{false}\}$;
PTCTL formula ϕ;
{
  Construct Param. Region Graph $G_{S:\phi} = (R, F)$; (1)
  $z_0 = \phi$; (2)
  for $i = 0, 1, \ldots$ { (3)
    OTF_PAnalyze(z_i, ϕ); (4)
    $z_{i+1} = z_i \cap \pi(z_i, \phi)$; (5)
    if $(z_{i+1} = z_i)$ **break**; } (6)
  if $(R_0 \neq \emptyset)$ {
    // $R_0 \subseteq z_i$ is a set of initial regions of S (7)
    $V_{H,\phi} = \bigvee_{r_0 \in R_0} L^\phi(r_0)$; (8)
    $S_C =$ **Synthesize_Predicate**$(S|_{V_{H,\phi}}, \mu_c, z_i)$; (9)
    return $(V_{H,\phi}, S_C)$; } (10)
  else return (*false*, NULL); // No solution (11)
}

z_i is a parametric region, $i \geq 0$,
$V_{H,\phi} \subseteq \mathcal{V}_{H,\phi}$ is a condition on parameter values, S_C: PTA.

thesizable transition f (*i.e.*, $\mu_c(f) = \mathit{false}$), the behavior of a run from f to e does not leave the fix-point region z.

3.3 PERTS Synthesis Algorithm

We propose a solution to the PERTS synthesis problem by an integration of the *iterative synthesis* algorithm (described in Table 2 and the *parametric analysis* algorithm (described in Table 1. The basis is the synthesis algorithm and parametric analysis is performed *on-the-fly*. Parametric regions are labeled as and when they are included into the fix-point set of regions derived in each iteration. Here, *labeling regions* means regions are associated with conditions on parameter valuations such that computation runs starting from those regions satisfy a given specification under the associated conditions.

The algorithm for this approach is given as function **PSynth**() in Table 4, where a system PTA S is given along with a synthesis mask μ_c and a PTCTL property ϕ. On-the-fly parametric analysis, as represented by **OTF_PAnalyze**() in Table 5, is performed in each iteration (Step (4) of Table 4). A restricted behavior is synthesized in Step (9) using the fix-point region z_i. In the rest of this subsection, we will state the completeness and termination of the iterative algorithm. Proofs are omitted due to page-limits.

Lemma 1 *A feasible computation run cannot simultaneously belong to two sets of runs of S interpreted under two contradictory parameter valuations.*

Theorem 1 **Completeness** *There does not exist a state $s \in Q_S$ and a parameter valuation $v \in \mathcal{V}_{H,\phi}$ such that s is reachable in $S|_v$, $s \notin Q_{S_C}$, and $s \models_v \phi$, where Q_S and Q_{S_C} are respectively the state spaces of a system S and its corresponding controlled system S_C derived using the iterative parametric controller synthesis algorithm.*

Table 5: On-the-Fly Parametric Analysis Algorithm

OTF_PAnalyze(Z, ϕ)
Set of Parametric Regions Z;
PTCTL Formula ϕ;
{
  for each $r \in Z$, recursively compute label $L^\phi(r)$; (1)
}

Theorem 2 **Termination** *The iterative parametric controller synthesis algorithm is guaranteed to terminate.*

4 Application Experiment

The proposed PERTS synthesis method is applied to an example: a ◇ timed-game. This example illustrates how the proposed method for parametric embedded real-time system synthesis works and how syntheses of trigger predicates on a transition correspond to prunings of a parametric region graph. A system PTA for this example is given in Fig. 1 and

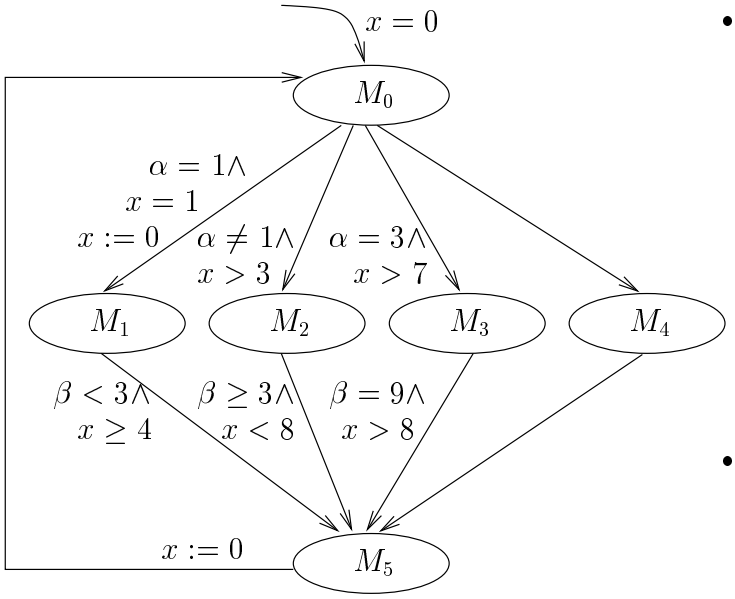

Figure 1: **System PTA S for Example 1**

the PTCTL specification for this example is as follows.

$$\phi = \exists \Diamond_{<\sigma} (M_5 \wedge x \geq 5 \wedge x \leq 7) \tag{3}$$

where σ is a parameter, M_5 is a mode name, and x is a clock variable. For simplicity, it is assumed that all transitions are synthesizable.

The PERTS synthesis algorithm given in Section 3.3 was used to synthesize a system PTA S illustrated in Fig. 1 such that it satisfied the PTCTL specification ϕ given in Equation (3). A parametric region graph $G_{S:\phi} = (R, F)$ was constructed, whose *pruned* version is shown in Fig. 2. Just as a complete parametric region graph depicts the behavior of a system PTA S, a *pruned* parametric region graph depicts the behavior of a *controlled* system PTA S_C. A pruning in the graph is depicted as a dashed bold arrow and represents a control synthesis (refinement) of trigger predicates on some transition. A parametric region is represented by an oval in Fig. 2. Each oval is labeled by a region name (r_i), a mode name (M_k), a clock interval ($[u, v)$), and an optional parametric condition ($\alpha \ldots$). A clock interval corresponds to the valuation of clock x in the system PTA description (Fig. 1). For ease of illustration, a parametric condition is given at the top of a sub-graph, instead of repeating it in each region of the sub-graph.

Details of the iterations in our proposed **PSynth()** method (Table 4) are as follows.

- *Initialization*: In *Step* (2), regions satisfying ϕ are included in initial fix-point, $z_0 = \{r_1, r_2, \ldots, r_9\}$.

- *Iteration* $i = 0$: In *Step* (4), regions in z_0 are labelled with parametric conditions as follows
  $L^\phi(r_1) = (\alpha = 1 \wedge \beta < 3 \wedge 1 \leq \sigma \leq 3)$,
  $L^\phi(r_2) = (\alpha = 1 \wedge \beta < 3 \wedge 1 \leq \sigma \leq 2)$,
  $L^\phi(r_3) = (\alpha = 1 \wedge \beta < 3 \wedge \sigma = 1)$,
  $L^\phi(r_4) = (\alpha \neq 1 \wedge \beta \geq 3 \wedge 1 \leq \sigma \leq 3)$,
  $L^\phi(r_5) = (\alpha \neq 1 \wedge \beta \geq 3 \wedge 1 \leq \sigma \leq 2)$,
  $L^\phi(r_6) = (\alpha \neq 1 \wedge \beta \geq 3 \wedge \sigma = 1)$,
  $L^\phi(r_7) = (1 \leq \sigma \leq 2)$,
  $L^\phi(r_8) = (1 \leq \sigma \leq 2)$, and
  $L^\phi(r_9) = (\sigma = 1)$.
  In *Step* (5), the next fix-point region space is computed as $z_1 = \{r_1, \ldots, r_9, r_{10}, \ldots, r_{18}\}$.

- *Iteration* $i = 1$: In *Step* (4), new regions in z_1 are labelled with parametric conditions as follows
  $L^\phi(r_{10}) = L^\phi(r_1)$, $L^\phi(r_{11}) = L^\phi(r_2)$,
  $L^\phi(r_{12}) = L^\phi(r_3)$, $L^\phi(r_{13}) = L^\phi(r_4)$,
  $L^\phi(r_{14}) = L^\phi(r_5)$, $L^\phi(r_{15}) = L^\phi(r_6)$,
  $L^\phi(r_{16}) = L^\phi(r_7)$, $L^\phi(r_{17}) = L^\phi(r_8)$,
  $L^\phi(r_{18}) = L^\phi(r_9)$.
  In *Step* (5), the next fix-point region space is computed as $z_2 = \{r_1, \ldots, r_{18}, r_{19}, \ldots, r_{27}\}$.

- *Iteration* $i = 2$: In *Step* (4), new regions in z_2 are labelled with parametric conditions as follows
  $L^\phi(r_{19}) = (\alpha = 1 \wedge \beta < 3 \wedge 2 \leq \sigma \leq 4)$,
  $L^\phi(r_{20}) = (\alpha \neq 1 \wedge \beta \geq 3 \wedge 2 \leq \sigma \leq 4)$,
  $L^\phi(r_{21}) = L^\phi(r_{13})$, $L^\phi(r_{22}) = L^\phi(r_{14})$,
  $L^\phi(r_{23}) = L^\phi(r_{15})$, $L^\phi(r_{24}) = 2 \leq \sigma \leq 4$,
  $L^\phi(r_{25}) = L^\phi(r_{16})$, $L^\phi(r_{26}) = L^\phi(r_{17})$,
  $L^\phi(r_{27}) = L^\phi(r_{18})$.
  In *Step* (5), the next fix-point region space is computed as $z_3 = \{r_1, \ldots, r_{27}, r_{28}, \ldots, r_{32}\}$.

- *Iteration* $i = 3$: In *Step* (4), new regions in z_3 are labelled with parametric conditions as follows
  $L^\phi(r_{28}) = (\alpha = 1 \wedge \beta < 3 \wedge 3 \leq \sigma \leq 5)$,
  $L^\phi(r_{29}) = (\alpha \neq 1 \wedge \beta \geq 3 \wedge 3 \leq \sigma \leq 5)$,
  $L^\phi(r_{30}) = L^\phi(r_{20})$, $L^\phi(r_{31}) = 3 \leq \sigma \leq 5$,
  $L^\phi(r_{32}) = L^\phi(r_{24})$.
  In *Step* (5), the next fix-point region space is computed as $z_4 = \{r_1, \ldots, r_{32}, r_{33}, \ldots, r_{36}\}$.

- *Iteration* $i = 4$: In *Step* (4), new regions in z_4 are labelled with parametric conditions as follows
  $L^\phi(r_{33}) = (\alpha = 1 \wedge \beta < 3 \wedge 4 \leq \sigma \leq 6)$,
  $L^\phi(r_{34}) = L^\phi(r_{29})$, $L^\phi(r_{35}) = 4 \leq \sigma \leq 6$,
  $L^\phi(r_{36}) = L^\phi(r_{31})$.
  In *Step* (5), the next fix-point region space is computed as $z_5 = \{r_1, \ldots, r_{36}, r_{37}, \ldots, r_{40}\}$.

- *Iteration* $i = 5$: In *Step* (4), new regions in z_5 are labelled with parametric conditions as follows

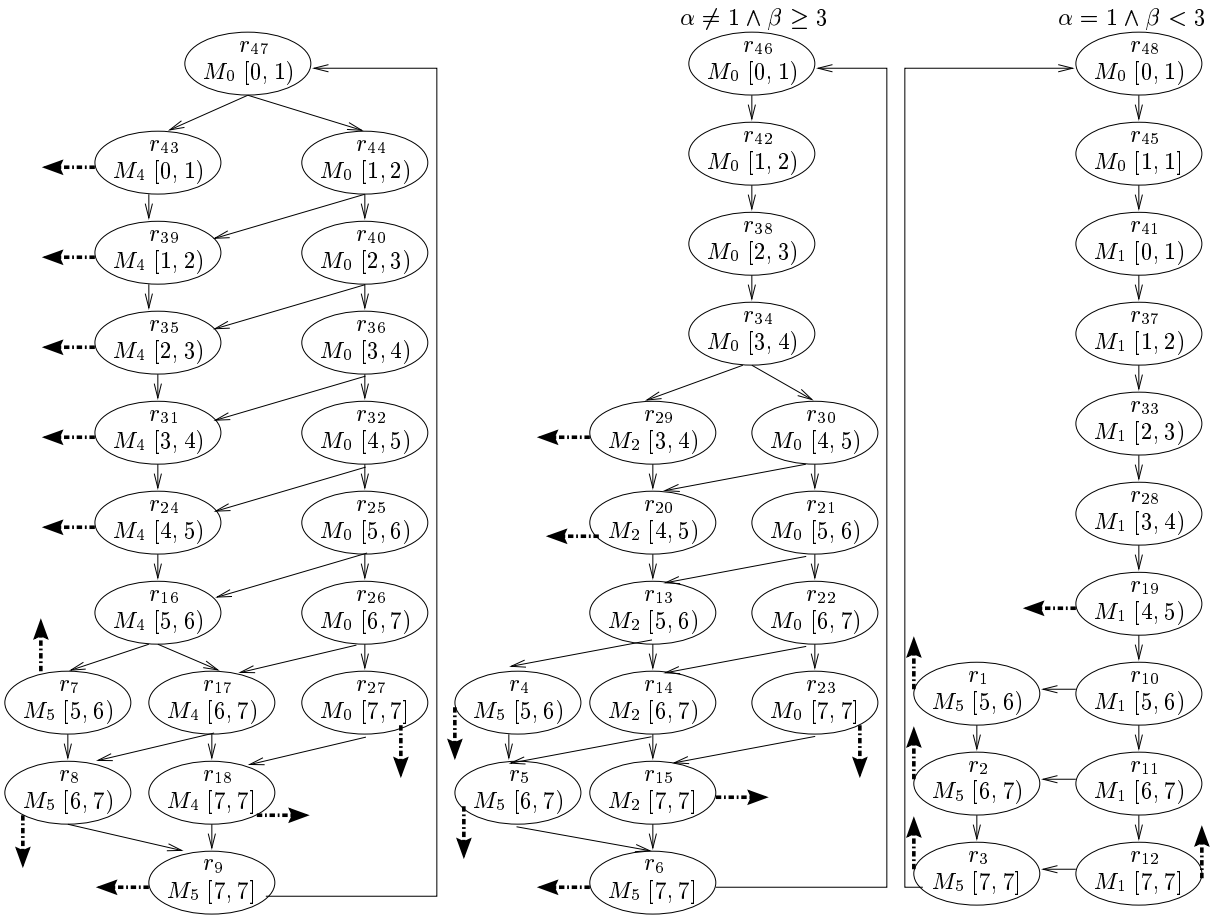

Figure 2: **Pruned Parametric Region Graph for Example 1**

$L^\phi(r_{37}) = (\alpha = 1 \wedge \beta < 3 \wedge 5 \leq \sigma \leq 7)$,
$L^\phi(r_{38}) = (\alpha \neq 1 \wedge \beta \geq 3 \wedge 4 \leq \sigma \leq 6)$,
$L^\phi(r_{39}) = 5 \leq \sigma \leq 7$,
$L^\phi(r_{40}) = L^\phi(r_{35})$.
In *Step* (5), the next fix-point region space is computed as $z_6 = \{r_1, \ldots, r_{40}, r_{41}, \ldots, r_{44}\}$.

- *Iteration* $i = 6$: In *Step* (4), new regions in z_6 are labelled with parametric conditions as follows
  $L^\phi(r_{41}) = (\alpha = 1 \wedge \beta < 3 \wedge 6 \leq \sigma \leq 8)$,
  $L^\phi(r_{42}) = (\alpha \neq 1 \wedge \beta \geq 3 \wedge 5 \leq \sigma \leq 7)$,
  $L^\phi(r_{43}) = 6 \leq \sigma \leq 8$,
  $L^\phi(r_{44}) = L^\phi(r_{39})$.
  In *Step* (5), the next fix-point region space is computed as $z_7 = \{r_1, \ldots, r_{44}, r_{45}, \ldots, r_{47}\}$.

- *Iteration* $i = 7$: In *Step* (4), new regions in z_7 are labelled with parametric conditions as follows
  $L^\phi(r_{45}) = (\alpha = 1 \wedge \beta < 3 \wedge 6 \leq \sigma \leq 8)$,
  $L^\phi(r_{46}) = (\alpha \neq 1 \wedge \beta \geq 3 \wedge 6 \leq \sigma \leq 8)$,
  $L^\phi(r_{47}) = L^\phi(r_{43})$
  In *Step* (5), the next fix-point region space is computed as $z_8 = \{r_1, \ldots, r_{47}, r_{48}\}$.

- *Iteration* $i = 8$: In *Step* (4), new regions in z_8 are labelled with parametric conditions as follows
  $L^\phi(r_{48}) = (\alpha = 1 \wedge \beta < 3 \wedge 7 \leq \sigma \leq 9)$. In *Step* (5), the next fix-point region space is computed as $z_9 = \{r_1, \ldots, r_{48}\}$, which is the same as z_8. Hence, the loop terminates in *Step* (6).

- *Step* (7): The set of initial regions is $R_0 = \{r_{46}, r_{48}, r_{47}\}$.

- *Step* (8): The general parametric condition is obtained as:

$$\begin{array}{rcl} V_{H,\phi} & = & L^\phi(r_{46}) \vee L^\phi(r_{48}) \vee L^\phi(r_{47}) \\ & = & (\alpha \neq 1 \wedge \beta \geq 3 \wedge 6 \leq \sigma \leq 8) \vee \\ & & (\alpha = 1 \wedge \beta < 3 \wedge 7 \leq \sigma \leq 9) \vee \\ & & (6 \leq \sigma \leq 8) \end{array} \quad (4)$$

- *Step* (9): The system synthesized is illustrated in Fig. 3.

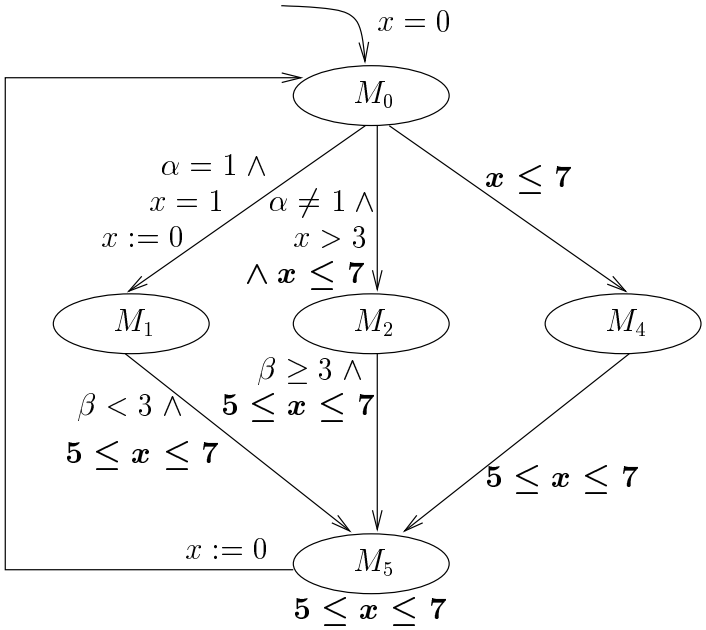

Figure 3: **Synthesized PTA S_C for Example 1**

- *Step* (10): $(V_{H,\phi}, S_C)$ is the final output result.

5 Conclusion

An embedded real-time system synthesis problem was extended to include *static parameters* in system descriptions and *quantitative parameters* in specifications. The parametric embedded real-time system (PERTS) synthesis problem was presented and solved. A PERTS synthesis algorithm based on iterative system synthesis was proposed to solve the problem. The completeness and soundness of the algorithm were validated analytically. An example experiment was conducted to show the feasibility of our PERTS synthesis algorithm. Using our approach, system engineers can now take system parameters into consideration while designing embedded real-time systems. The formal approach also allows correct verified designs.

References

[1] K. Altisen, G. Gobler, A. Pneuli, J. Sifakis, S. Tripakis, and S. Yovine. A framework for scheduler synthesis. In *Real-Time System Symposium (RTSS'99)*. IEEE Computer Society Press, 1999.

[2] R. Alur, C. Courcoubetis, N. Halbwachs, and D. Dill. Modeling checking for real-time systems. In *Proc. IEEE Logics in Computer Science*, 1990.

[3] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183 – 235, 1994.

[4] E. Asarin, O. Maler, and A. Pneuli. Symbolic controller synthesis for discrete and timed systems. In P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems II*, volume 999, pages 1 – 20. Lecture Notes in Computer Science, Springer Verlag, 1995.

[5] E. Asarin, O. Maler, A. Pneuli, and J. Sifakis. Controller synthesis for timed automata. In *Proc. System Structure and Control*. IFAC, Elsevier, July 1998.

[6] V. Carchiolo, M. Malgeri, and G. Mangioni. Hardware software synthesis of formal specifications in codesign of embedded systems. *ACM Transactions on Design Automation of Electronic Systems*, 5(3):399–432, July 2000.

[7] J.-M. Fu, T.-Y. Lee, P.-A. Hsiung, and S.-J. Chen. Hardware-software timing coverification of distributed embedded systems. *IEICE Trans. on Information and Systems*, E83-D(9):1731–1740, September 2000.

[8] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *Proc. IEEE Logics in Computer Science*, 1992.

[9] P.-A. Hsiung. Embedded software verification in hardware-software codesign. *Journal of Systems Architecture — the Euromicro Journal*, 46(15):1435–1450, November 2000.

[10] P.-A. Hsiung. Hardware-software timing coverification of concurrent embedded real-time systems. *IEE Proceedings on Computers and Digital Techniques*, 147(2):81–90, March 2000.

[11] P.-A. Hsiung. POSE: A parallel object-oriented synthesis environment. *ACM Transactions on Design Automation of Electronic Systems*, 6(1):to appear, January 2001.

[12] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In E.W. Mayr and C. Puech, editors, *12th Annual Symposium on Theoretical Aspects of Computer Science (STACS'95)*, volume 900, pages 229 – 242. Lecture Notes in Computer Science, Springer Verlag, March 1995.

[13] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal of Control and Optimization*, 25:206 – 230, 1987.

[14] P.J. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77:81 – 98, 1989.

[15] F. Wang and P.-A. Hsiung. Parametric analysis of computer systems. In Michael Johnson, editor, *International Conference on Algebraic Methodology and Software Technology (AMAST'97)*, volume 1349, pages 539 – 553. Lecture Notes in Computer Science, Springer Verlag, December 1997.

[16] H. Wong-Toi. The synthesis of controllers for linear hybrid automata. In *Procs. CDC'97*, 1997.

[17] H. Wong-Toi and G. Hoffman. The control of dense real-time discrete event systems. Technical Report STAN-CS-92-1411, Stanford University, 1992.