A Path-based Equivalence Checking Method for Petri Net based Models of Programs

Soumyadip Bandyopadhyay, Dipankar Sarkar, Kunal Banerjee and Chittaranjan Mandal

Indian Institute of Technology, Kharagpur, India

Keywords: Translation Validation, Equivalence Checking, PRES+ Model, FSMD Model.

Abstract: Programs are often subjected to significant optimizing and parallelizing transformations. It is therefore important to model parallel behaviours and formally verify the equivalence of their functionalities. In this work, the untimed PRES+ model (Petri net based Representation of Embedded Systems) encompassing data processing is used to model parallel behaviours. Being value based with inherent scope of capturing parallelism, PRES+ models depict such data dependencies more directly; accordingly, they are likely to be more convenient as the intermediate representations (IRs) of both the source and the transformed codes for translation validation than strictly sequential variable-based IRs like Finite State Machines with Datapath (FSMDs) (which are essentially sequential control data-flow graphs (CDFGs)). In this work, a path based equivalence checking method for PRES+ models is presented.

1 INTRODUCTION

Recent advancement of multi core and multi processor systems has enabled incorporation of concurrent applications in embedded software systems through extensive optimizing transformations for better time performance and resource utilization (Gupta et al., 2003). If such optimizations are carried out by untrusted compilers, they can result in software bugs. Hence, it is important to verify whether the optimized code faithfully represents the intended functionality.

A comprehensive list of models proposed to represent programming systems and their validation can be found in (Edwards et al., 1999; Akl, 1997; Milner, 1989). Petri nets have long been popular modeling paradigm for concurrent behaviours. The untimed one safe PRES+ model (Petri net based Representation for Embedded Systems) (Cortés et al., 2003) enhances the classical Petri net model to capture natural concurrency and well defined semantics of computations over integers, reals and general data structures. Analyses of dependencies among the operations in a program lie at the core of many optimizing and parallelizing transformations. Being value based with inherent scope of capturing parallelism, PRES+ models depict such data dependencies more directly; accordingly, they are likely to be more convenient as the intermediate representations (IRs) of both the source and the transformed codes for translation validation than strictly sequential variable-based IRs like all types of control data-flow graphs, communicating sequential processes, etc.

Behavioural verification involves demonstrating the input-output equivalence of all computations represented by the original behavioural description with those of the transformed behavioural description. From the success of path based equivalence checking of CDFG models (essentially FSMDs) recorded in (Banerjee et al., 2014), it is perceived that a similar approach is worth pursuing for PRES+ models. Path structures in PRES+ models, however, are far more complex than those in CDFG models due to the presence of parallel threads of computations in the former. The major contributions of the present paper are as follows:

1. A formal definition of computations of untimed PRES+ models is provided.
2. An algorithm for path construction of PRES+ models is provided and formally treated.
3. An algorithm for path based validation of PRES+ models is provided and formally treated.

Organization: The rest of the paper is organized as follows. Section 3 describes an overview through a motivating example. Section 4 formally introduces the untimed PRES+ model, its computational semantics and also the notion of computational equivalence between two PRES+ models. The concept of cut-
points and paths in a PRES+ model is introduced and a path construction algorithm is also given in section 5; an equivalence checking algorithm is given in section 6. Experiments on some benchmarks can be found in section 7. Section 8 states the related work on translation validation. The paper is concluded in section 9.

2 FRAMEWORK

Figure 1 depicts the actual framework of the present work. A software program is compiled using some compiler transformation techniques and an optimized intermediate code is produced. Validation of such transformations is an undecidable problem. On the other hand, if we are able to establish the computational equivalence between the original and the transformed programs, we can claim that the transformations applied for the specific run is correct. Therefore, our main target is to develop an equivalence checker for translation validation.

For program analysis, it is necessary to translate any program to its equivalent formal model representation. As the main target of this work is to validate code optimizing and several parallelizing transformations, a parallel model of computation (MoC) is necessary. In this work, the PRES+ model, whose underlying structure is a one-safe Petri net model with token holding values, is selected as the parallel MoC. Therefore, PRES+ models are constructed from both the original and the transformed program. The automated PRES+ construction method from high level language is reported in (Cortes et al., 2000). Here, our main task is to devise a PRES+ equivalence checker which takes two PRES+ models as inputs and returns either “yes” or “no” as its output. If the equivalence checker gives a “yes” response, then the two programs are equivalent, i.e., particular transformations which are carried out by the compiler are correct; if it gives a “no” response, then the two programs may not be equivalent. Hence, our method is sound but not complete and it may give a false negative result. The basic steps of the equivalence checking procedure are as follows: (1) In the first step, a PRES+ model is partitioned into several fragments which are called paths; the paths are obtained by cutting a loop in at least one cut-point which is adapted from (Floyd, 1967); any computation of the model can now be represented as a concatenation of these paths. (2) Next, checking whether for all paths in the PRES+ model N₀, say, which corresponds to the source program, there exists a path in the PRES+ model N₁, say, which corresponds to the transformed program such that the two paths are equivalent, i.e., their data computations and conditions of execution are identical and their input and output places have correspondence. (3) Repeat steps 1 and 2 with N₀ and N₁ interchanged. The major challenges of this work are as follows:

1. Path construction procedure for a PRES+ model
2. Equivalence checking method for PRES+ models.

3 METHODOLOGY

Before describing the formal notion of path based equivalence checking mechanism between two PRES+ models, in this section, we underline through an example some of the relevant issues which arise during development of an equivalence checker for PRES+ models.

Example 1. Figure 2 (a) represents an initial program which computes \([100] + [100] + [100]\). Figure 2(b) and 2(c) pertain to programs transformed using loop swapping transformation and thread level parallelizing transformations, respectively. Figures 3 (a), 3(b) and 3(c) depict the PRES+ models corresponding to the programs given in Figures 2(a), 2(b) and 2(c), respectively. Automated model construction from high level program to its equivalent PRES+ model is reported in (Cortes et al., 2000).

Let us now address the issues in finding equivalence between the PRES+ models of Figures 3(a) and 3(c). In Figure 3(a), the fragments p₁.(p₂)₂.p₅ computes the first term \([100] + [100]\) and the fragment
A Path-based Equivalence Checking Method for Petri Net based Models of Programs

$p_2 \cdot (p_4)^m \cdot p_6$ computes the second term \((100+1)\). Correspondingly, in Figure 3(c), $p_1 \cdot (p_2)^n \cdot p_1$ computes the first term and $p_2 \cdot (p_3)^m \cdot p_5$ computes the second term \((n \neq m)\). In general, in a computation, the number of traversals of a loop in a PRES+ model depends upon the input token values. Since equivalence has to be established for all computations, the notion of finite computation paths is used so that any computation can be captured in terms of these paths. To do so, we cut the loops by introducing cut-points so that every loop is cut in at least one cut-point; each path will originate from a set of cut-points and extend up to a cut-point without having any intermediary cut-points. In Figure 3(a), suppose the cut-points are $p_1, p_2$ (as in-ports), $p_3, p_4$ (for cutting the loops), $p_5$ (for the out-port) and $p_5, p_6$ (extra cut-points). A path is represented as a sequence of maximally parallelizable transitions; hence the corresponding cut-points are also depicted in Figure 3(a) using inverted dotted triangular boxes, such as $\alpha_2 \equiv \{v_1\}$ and the computation $p_5$ can be represented as $(\alpha_1 \parallel \alpha_2, \alpha_3 \parallel \alpha_4 \parallel \alpha_5 \parallel \alpha_6, \alpha_7)$. Thus, introducing extra cut-points judiciously is an important step which will be discussed in detail in the subsequent sections.

Similarly, in Figure 3(c), the paths are as shown by inverted dotted triangular boxes with cut-points \(\{p_1, p_2, p_3, p_4, p_5, p_6\}\). Two entities are used to characterize a path $\alpha$: (1) the condition of execution $R_\alpha$, and (2) the data transformation $r_\alpha$; these are computed by the forward substitution method. For the paths $\alpha_1$ and $\alpha_3$ in Figure 3(a), $R_{\alpha_1}(v_{p_1}) : “true”$ and $R_{\alpha_3}(v_{p_2}) : v_{p_1} \cdot 7 \leq 100$; and the data transformations are $r_{\alpha_1}(v_{p_1}) = 1$ and $r_{\alpha_3}(v_{p_2}) = v_{p_3} + 1$.

Let the in-port association $f_{in} : \text{in}P_1 \times \text{in}P_1$ be \{(p_1, p_1), (p_2, p_1)\}; let out-port association $f_{out} : \text{out}P_0 \leftrightarrow \text{out}P_1$ be $p_1 \leftrightarrow p_6$. For each path of Figure 3(a), the equivalent path of Figure 3(c) is obtained. Specifically, we obtain $\alpha_1 \simeq \beta_1$, $\alpha_2 \simeq \beta_1$, $\alpha_3 \simeq \beta_2$, $\alpha_4 \simeq \beta_3$, $\alpha_5 \simeq \beta_4$, $\alpha_6 \simeq \beta_5$ and $\alpha_7 \simeq \beta_6$; also, their input and output places have correspondence. Since all the paths of the original behaviour have some equivalent paths in the transformed behaviour, and vice versa, the models are asserted to be equivalent. We can also establish the equivalence between the models of Figures 3(a) and 3(b) identically.

Path Extension: Code motion transformations move code segments beyond the basic block boundaries; consequently, some paths of one model may be found to have no equivalent paths in the other model. Such paths will have to be extended through its subsequent path(s) till paths equivalent to the resulting concatenated path(s) are obtained. The idea of path extension is similar to that of path based FSMD equivalence checking mechanism (Banerjee et al., 2014). Intricacies, however, arise due to the presence of paths parallel to the path being extended. This situation is presented in section 6 through Example 5.

4 THE PRES+ MODEL AND ITS COMPUTATION

A PRES+ model is a 6–tuple $N = (P, T, I, O, \text{in}P, \text{out}P)$, where the members are de-
fined as follows. \( P \): a finite non-empty set of places. A place \( p \) is capable of holding a token having a value \( v_p \) from a domain \( D_p \). A token value may be of type Boolean, integer, etc., or a user-defined type of any complexity (for instance, a structure or a set). \( T \) is a finite non-empty set of transitions; the relation \( I \subseteq P \times T \) is a flow relation from places to transitions; a place \( p \in P \) is to be an input place of a transition \( t \) if \((p,t) \in I\); \( \kappa_t \) denotes the input places of \( t \). The relation \( O \subseteq T \times P \) is a flow relation from transitions to places; a place \( p \in P \) is said to be an output place of a transition \( t \) if \((t,p) \in O\); \( \kappa_t \) denotes the output places of \( t \). A place \( p \in P \) is said to be an input-port iff \((t,p) \notin O\), for all \( t \in T \). Likewise, a place \( p \in P \) is said to be an out-port iff \((p,t) \notin I\), for all \( t \in T \). The set \( \text{in} P \subseteq P \) is a non-empty set of in-ports and the set \( \text{out} P \subseteq P \) is a non-empty set of out-ports. The pre-set \( p \) (post-set \( p' \)) of a place \( p \) comprises all the transitions of \( p \) which is an output (input) place. A function \( f_t \) and a guard condition \( g_t \) are associated with a transition \( t \). The function \( f_t \) captures the functional transformation that takes place on the token values in \( t \) to produce the same token value at all the post-places of \( t \). The model is deterministic and completely specified; that is, for any set \( P \) of places, \((t)\) for any two transitions \( t, t' \in \text{P} \), if \( \kappa_t \cap \kappa_t' \neq \emptyset \), then \( g(t) \land g(t') = \text{false} \), and (ii) \( \forall t \in \text{P} \), \( g(t) = \text{true} \).

A marking \( M \) is an ordered 2-tuple of the form \((P_M, \text{val} M)\) where, \( P_M \) is a subset of the places of \( M \) where tokens are present and \( \text{val} M : P \rightarrow \text{DP} \) is a mapping from places to token values. The token value \( \text{val} M(p) \) in \( p \) for the marking \( M \) is also denoted as \( v^p \), where \( p \in \text{P}_M \); otherwise it is undefined, denoted as \( \text{val} M \). A marking \( M_0 \) is an initial marking with \( P_{M_0} = \text{in} P \). In a PRES+ model, a transition \( t \in T \) is bound for a given marking \( M \); \((P_M, \text{val} M)\) iff all its input places are marked, i.e., \( t \subseteq P_M \). A bound transition \( t \in T \) for a given marking \( M \) is enabled iff \( g_t(v^M_t, v^M) \) holds, where \( v^M_t = \{p_1, \ldots, p_n\} \). The set of enabled transitions for a marking \( M \) is denoted as \( T_M \). All the enabled transitions are assumed to fire simultaneously. A marking \( M^+ \) is said to be a successor of the marking \( M \) if \( M^+ \) contains all the post-places of the enabled transitions of \( M \) and also all the places of \( M \) whose post-transitions are not enabled; symbolically, \( P_{M^+} = \{p \mid p \in \kappa_t \} \) and \( t \in T_{M} \cup \{p \mid p \in P_M \text{ and } p \notin T_{M} \} \); for any marking \( M \) and its successor marking \( M^+ \), for any place \( p \in \kappa_t \), where \( t \in T_{M^+} \) having \( \kappa_t = \{p_1, \ldots, p_n\} \) and associated with function \( f_t, v^M = f_t(v^M, v^M, \ldots, v^M) \); for any place \( p \in \kappa_t \), where \( t \notin T_{M^+} \), \( v^M = v^M \). Specifically, for an assignment statement of a high level language of the form \( x := y + c/d + s + t \), the transition \( t \) will have \( \kappa_t = \{p_1, p_2, p_3\} \), \( \kappa_t = \{p\} \) and \( f_t \) will be maintained as \( v_{p_1} + v_{p_2}/v_{p_3} + 4 \).

Let \( p_1 < p_2 < \ldots < p_n \) be an ordering over the places \( P = \{p_1, \ldots, p_n\} \). Let \( \text{val} M \) represent the vector \( \langle v^M_{p_1}, v^M_{p_2}, \ldots, v^M_{p_n} \rangle \) of values associated with the places restricted to the subset \( P_0 \) of places. We refer to \( \text{val} M \) as the value vector for the marking \( M \) and \( \text{val} M'(P_M) \) to represent its restriction \( \langle v^M_{p_1}, v^M_{p_2}, \ldots, v^M_{p_n} \rangle \). It is to be noted that the transitions may also have delay and deadline time parameters; models having these features are called timed PRES+ models. We deal with only untimed PRES+ models. Also, we consider one-safe PRES+ models whose structures ensure that at any point a place may hold at most one token. Henceforth, by a PRES+ model we only mean a one-safe untimed PRES+ model.

**Definition 1** (Successor Relation Between Two Transitions). A transition \( t_i \) succeeds a transition \( t_j \), denoted as \( t_i \succ t_j \), if (i) \( t_i \in T \) or (ii) \( t_i \in T \) and \( t_j \notin T \) and \( \forall t_k \in T_M, (t_k \neq t_i \land t_k \geq t_i \land t_k \geq t_j) \Rightarrow t_k \) \( \forall t_i \in T \), \( t_i \neq t_j \), \( t_i \succ t_j \) is used as a shorthand for \( \forall t_i \in T \), \( t_i \neq t_j \), \( t_i \succ t_j \).

**Definition 2** (Set of Maximally Parallelizable Transitions). Two transitions \( t_i \) and \( t_j \) are said to be parallelizable, denoted as \( t_i \prec t_j \), if (i) \( t_i \neq t_j \) and \( t_j \neq t_i \) and (ii) \( \forall t_k \in T_M, (t_k \neq t_i \land t_k \geq t_i \land t_k \geq t_j) \Rightarrow t_k \prec t_i \). A set \( T \) is \{t_1, t_2, \ldots, t_k\} of transitions is said to be parallelizable if \( \forall t_i, t_j \in T, t_i \neq t_j \), \( t_i \prec t_j \) holds. The set \( T \) is said to be maximally parallelizable if there is no set \( T' \) of parallelizable transitions which contains \( T \).

**Definition 3** (Computation in a PRES+ Model). In a PRES+ model \( N \) a computation \( \mu_{N,p} \) of an out-port \( p \) is a sequence \( \langle T_1, T_2, \ldots, T_i \rangle \) of sets of maximally parallelizable transitions where, \( T_i \subseteq \text{in} P, p \in T_i \) and \( T_i \subseteq \text{P}_M \) in the marking \( M \) and \( T_{i+1} \subseteq \text{P}_M \) in the marking \( M_{i+1} \), then \( M_{i+1} = M_i \), for all \( i, 1 < i < l \). When we need to refer explicitly to the the initial value vector \( \text{val} M(t_1) \), we represent the computation \( \mu_{N,p} \) as an ordered pair \( \langle \langle T_1, T_1, \ldots, T_i \rangle, \text{val} M(t_1) \rangle \).

If there are \( k \) out-ports, then for each initial marking \( M_0 \), there are at most \( k \) computations, one for each out-port. (We drop the suffix(es) of \( \mu \) when they are clear from the context. Thus, more specifically, when there is no other PRES+ model we use the symbol \( \mu_p \).)

There are two entities associated with a computation \( \mu_p \) of an out-port \( p \):

1. The **condition** \( \mu_p(\text{val} M(t_1)) \) of \( \mu_p \) on the initial token values at \( T_1 \) under which \( \mu_p \) takes place.
2. The **data transformation** \( \mu_p(\text{val} M(t_1)) \) of \( \mu_p \) which provides the token value in the out-port \( p \) after \( \mu_p \) is completed.
Two PRES+ models $N_0$ and $N_1$ will not be functionally equivalent unless they are cardinality-equivalent (Cortés et al., 2003), or more specifically, input-output cardinality-equivalent, that is, their respective in-ports and out-ports are bijective; the corresponding bijections are denoted as $f_{in}: \text{in}P_0 \leftrightarrow \text{in}P_1$ and $f_{out}: \text{out}P_0 \leftrightarrow \text{out}P_1$.

Let $N_0 = \langle P_0, T_0, I_0, O_0, \text{in}P_0, \text{out}P_0 \rangle$ and $N_1 = \langle P_1, T_1, I_1, O_1, \text{in}P_1, \text{out}P_1 \rangle$ be two cardinality-equivalent PRES+ models with in-port bijection $f_{in}$ and out-port bijection $f_{out}$. We now define the notions of computations of an out-port, containment of PRES+ models and the computational equivalence notions of computations of an out-port, containment of PRES+ models.

**Definition 4** (Equivalence of PRES+ Computations). Let $\mu_{0,p}$ be a computation of an out-port of $N_0$ of the form $(T_{0,1}, T_{0,2}, \ldots, T_{0,n_p})$ starting with an initial marking $M_{0,0} \supseteq T_{0,1}$: let $\mu_{1,\text{out}(p)}$ be a computation of the out-port $f_{\text{out}}(p)$ of $N_1$ of the form $(T_{1,1}, T_{1,2}, \ldots, T_{1,n_{out}(p)})$ starting with the initial marking $M_{1,0} \supseteq T_{1,1}$, where $\forall \mu \in \text{in}P_0: \mu = \text{val}_{M_{0,0}}(p) = \text{val}_{M_{1,0}}(f_{\text{out}}(p))$, $\forall p \in \text{in}P_0$. The computations $\mu_{0,p}$ and $\mu_{1,\text{out}(p)}$ are said to be equivalent (represented as $\mu_{0,p} \equiv \mu_{1,\text{out}(p)}$), if $R_{\mu_{0,p}}(\text{val}_{M_{0,0}}(T_{0,1})) \equiv R_{\mu_{1,\text{out}(p)}}(\text{val}_{M_{1,0}}(T_{1,1}))$ and $r_{\mu_{0,p}}(\text{val}_{M_{0,0}}(T_{0,1})) = r_{\mu_{1,\text{out}(p)}}(\text{val}_{M_{1,0}}(T_{1,1}))$.

**Definition 5** (Containment of PRES+ Models). A PRES+ model $N_0$ is said to be contained in a PRES+ model $N_1$, represented as $N_0 \subseteq N_1$, if, $\forall p \in \text{out}P_0$, for every computation $\mu_{0,p}$ of the out-port of $N_0$ starting with an initial marking $M_{0,0} \supseteq T_{0,1}$, there exists a computation $\mu_{1,\text{out}(p)}$ of the out-port of $N_1$ starting with an initial marking $M_{1,0} \supseteq T_{1,1}$, where $\forall \mu \in \text{in}P_0: \mu = \text{val}_{M_{0,0}}(p) = \text{val}_{M_{1,0}}(f_{\text{out}}(p))$, $\forall p \in \text{in}P_0$. Let $\mu_{0,p} \equiv \mu_{1,\text{out}(p)}$.$\mu_{0,p}$.

**Definition 6** (Computational Equivalence of PRES+ Models). The PRES+ models $N_0$ and $N_1$ are said to be computationally equivalent if $N_0 \equiv N_1$ and $N_1 \equiv N_0$.

## 5 PATHS OF A PRES+ MODEL

It is to be noted that the notion of paths and concatenated paths has already been covered in Example 1. In this section, we describe the formal notion of a path of a PRES+ model.

**Definition 7** (Back Edge). An edge $(t, p)$ from a transition $t$ to a place $p \in t^\circ$ is said to be a back edge with respect to an arbitrary DFS traversal, if $p$ is an ancestor of $t$ in that traversal of the PRES+ model.

**Definition 8** (Static Cut-point). A place $p$ is designated as a static cut-point with respect to an arbitrary DFS traversal if (i) $p$ is an in-port, or (ii) $p$ is an out-port or (iii) there is an edge $(t, p)$ which is a back edge with respect to that DFS traversal.

**Definition 9** (Path in a PRES+ Model). A finite path $\alpha$ in a PRES+ model from a set $T_1$ of transitions to a transition $t_i$ is a finite sequence of distinct sets of parallelizable transitions of the form $\langle T_1 = \{t_1, t_2, \ldots, t_k\}, T_2 = \{t_{k+1}, t_{k+2}, \ldots, t_{l-1}\}, \ldots, T_n = \{t_l\} \rangle$ satisfying the following properties:

1. All the members of $\circ T_1$ are cut-points.
2. All the members of $T_{n-1}^\circ$ are cut-points.
3. There is no cut-point in $T_m^\circ$, $1 \leq m < n$.
4. $\forall 1 \leq i \leq n, \forall p \in T_{i-1}^\circ$ if $p$ is not a cut-point, then $\exists k, 1 \leq k \leq i-1, p \in T_{k}^\circ$; thus, any pre-place of a transition which is not a cut-point must be a post-place of some preceding transition in the path.
5. There do not exist two transitions $t_i$ and $t_l$ in $\alpha$ such that $t_i \cap t_l \neq \emptyset$.
6. $\forall 1 \leq i \leq n, T_i$ is maximally parallelizable within the path, i.e., $\forall t_i \in T_i$ in the path such that $l \neq i$, $T_l \cup \{t_i\}$ is not parallelizable.

The set $\circ T_1$ of places is called the pre-set (preplaces) of the path $\alpha$, denoted as $\circ \alpha$; similarly, the post-set (post-places) $\alpha'$ of the path $\alpha$ is $T_n^\circ$. We can synonymously denote a path $\alpha = \langle T_1, T_2, \ldots, T_n \rangle$ as the sequence $\langle \circ T_1, \circ T_2, \ldots, \circ T_n \rangle$ of the sets of places from the place(s) $\circ T_1$ to the place(s) $T_n^\circ$.

In the following example, we show the need of extra cut-points such that any computation can be represented in terms of paths.

**Example 2.** In Figure 3(a), suppose the cut-points are $p_1, p_2$ (as in-ports), $p_3, p_4$ (for cutting the loops), $p_7$ (for the out-port); hence, the corresponding paths are as follows: $\alpha_1 = \langle \{t_1\} \rangle, \alpha_2 = \langle \{t_2\} \rangle, \alpha_3 = \langle \{t_3\} \rangle, \alpha_4 = \langle \{t_4\} \rangle$ and $\alpha_5 = \{\{t_4, t_5\}, \{t_6\}\}$. Let a computation $\mu_{p_7}$ of the out-port $p_7$, depicted as a sequence of maximally parallelizable transitions, be $\langle \{t_1, t_2\}, \{\{t_1, t_3\}^m, \{t_1, t_4\}^n, \{t_1, t_5\}^j, \{t_6\}\rangle$. The computation, however, cannot be obtained as a sequence of concatenation of paths from the set $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\}$. Instead, suppose the cut-points are $p_1, p_2, p_3, p_4, p_5, p_6$ and $p_7$; the corresponding paths are also depicted in Figure 3(a) using inverted dotted triangular boxes, such as $\alpha_2 = \langle \{t_1\} \rangle$. Now, $\mu_{p_7}$ can be represented as $\langle \alpha_1 \parallel \alpha_2 \parallel \alpha_3 \parallel \alpha_4 \parallel \alpha_5 \parallel \alpha_6 \parallel \alpha_7 \parallel \alpha_8 \rangle$.

If a path starts from the set of in-port cut-points, then the computation of the path starts with an initial marking $M_0$. Example 1 underlines the need for
introducing further cut-points and the notion of parallel paths and their concatenation for capturing computations. For the former, a notion of token tracking execution is necessary which essentially captures all computations of the model with the token values abstracted out and every loop traversed exactly once. Thus, a token tracking execution starts with an initial marking comprising tokens in the in-ports and tracks the progress of the tokens through the successor markings avoiding repetitions of subsequences of markings. If a given marking involves a token holding place with more than one outgoing transition, then firing of such transitions will be mutually exclusive of each other; hence there may be more than one alternative set of successor markings all of which are covered in a DFS manner by the token tracking execution mechanism. Note that the number of times a loop is executed varies from one execution to another depending upon the input. Hence if a given marking involves more than one place with at least one place having a back edge leading to itself, then the execution falls under a degenerate case whereupon dynamic cut-points have to be introduced exhaustively in all the places of the markings as captured by the following definition.

**Definition 10 (Dynamic Cut-point).** A place is designated as a dynamic cut-point if during a token tracking execution of the model (with static cut-points already incorporated), it occurs in a marking containing at least one cut-point. If the token tracking execution falls under the degenerate case, all the places occurring in the subsequent markings are also marked as dynamic cut-points until a marking is reached with a single post-transition.

The designation of dynamic cut-points is described through Example 3. The definition of path (Definition 9) is modified with “cut-points” read as both static and dynamic cut-points.

### 5.1 Path Construction Algorithm

We now describe the dynamic cut-point designation and the path construction algorithm through Example 3 where all the intricacies are covered. The pseudo code of the path construction algorithm is given below. The functional modules along with complexity and correctness analysis are given in (Bandyopadhyay, ) where, it is also shown that the complexity of the path construction algorithm is $O((|T|^3)|P|(|T|^2))$ which is further shown to reduce to $O(|T|^2)$, where $|T|$ is the cardinality of the set $T$ of transitions. Towards correctness of the algorithm, it is shown that all functional modules terminate, paths constructed by this algorithm satisfy the properties of paths (Definition 9) and the paths constructed by this algorithm give a path cover. It is to be noted that designation of dynamic cut-points and the path construction procedure goes in hand in hand.

1. Input: A PRES+ model $N$; Output: Set $Q$ of all paths.  
2. $M_b \leftarrow \text{inP}, Q \leftarrow \emptyset, T_{sh} \leftarrow \emptyset$ /* $M_b$ : marking at hand — initialized to in-ports; set $Q$ of all paths — initialized empty; $T_{sh}$ : transition sequence at hand — initially empty */
3. $T = \text{ComputeAllSetsOfConcurrentTransitions}(M_b,N);$ /* it takes $M_b$ and forms all possible sets of concurrent transitions that are bound to $M_b$ */
4. $\forall T \in T$
   - $Q \leftarrow Q \cup \text{obtainAllThePaths}(T_{sh},M_b,T,N);$ /* obtainAllThePaths appends the set $T$ of enabled transitions to the transition sequence $T_{sh}$ at hand. Token tracking execution and designation of dynamic cut-points is carried out by this module. For each cut-point, it calls the recursive function constructOnePath which constructs the path from a single cut-point to the set of cut-points by backward cone of foci method (using $T_{sh}$) — proceed beyond $T$ (recursively) */
5. Return $Q$

**Example 3.** Consider the model given in Figure 3(a). The token tracking execution starts with the initial marking $\{p_1,p_2\}$. After firing of $t_1$ and $t_2$, marking becomes $\{p_3,p_4\}$ with a back edge leading to $p_3$ and another leading to $p_4$. As $p_3$ and $p_4$ are cut-points, the paths are $\langle t_1 \rangle$ and $\langle t_2 \rangle$ which are constructed using backward cone of foci method. Since, the cardinality of the current marking $\{p_3,p_4\}$ is greater than one, the situation falls under degenerate case. Both $p_3$ and $p_4$ have two out-transitions each, i.e., $\{t_3,t_4\}$ and $\{t_5,t_6\}$. Therefore, four alternative sets of enabled transitions are obtained from the given marking, namely, $\{t_3,t_5\}, \{t_5,t_6\}, \{t_4,t_5\}$ and $\{t_4,t_6\}$. These four alternatives are explored in a DFS manner. For the set $\{t_3,t_5\}$, the successor marking becomes $\{p_5,p_6\}$ indicating a loop since the marking $\{p_3,p_6\}$ is repeated. Hence this DFS branch is not pursued. As the marking $\{p_3,p_6\}$ do not have a single post-transition, $p_6$ is designated as a dynamic cut-point. Therefore, the paths are $\langle t_3 \rangle$ and $\langle t_5 \rangle$. Similarly, when the set $\{t_4,t_6\}$ is processed, the place $p_5$ is designated as a dynamic cut-point. The paths are $\langle t_4 \rangle$ and $\langle t_6 \rangle$. For the set $\{t_4,t_5\}$, the successor marking becomes $\{p_5,p_6\}$. As $p_5^* = p_6^* = t_7$, at this point the token tracking execution ceases to exist in the degenerate case. Finally, $\{p_7\}$ is reached after firing of the
transition τ. The path is ⟨{τ}⟩. Therefore, the set of dynamic cut-points is computed as \{p_5, p_3\} and the set of paths is \{(t_1), \{(t_2), \{(t_4)\}, \{(t_5)\}, \{(t_6)\}, \{(t_7)\}\}. □

5.2 Characterization of a Path

We associate with a path α two entities namely, Rα, the condition of execution of the path α, and rα, the data transformation along the path α. For any computation µ of the form ⟨T_1, T_2, ...⟩ of the path α with \(T_i \subseteq P_M\), say, the condition Rα depicts the condition that must be satisfied by val_M(α) so that α is executed. The data transformation rα depicts the token value obtained in α after computation µ. Thus, the places in α contain the value rα(val_M(α)) after execution of the path α.

Example 4. Figure 4(a) depicts a PRES+ model having p_0, p_1, and p_5 as cut-points following the cut-point introduction rules given in the previous section. So the corresponding paths are α_1 = ⟨{t_1}⟩, α_2 = ⟨{t_2}, {t_3}, {t_4}⟩ and α_3 = ⟨{t_1}, {t_5}, {t_6}⟩, respectively. Figure 4(b) depicts how the data transformation (rα_3) and the condition of execution (Rα_3) for the path α_3 are computed. A forward traversal along the forward direction of the edges of the path α_3 from p_1 to p_4 is used for this purpose. We may use backward traversal (along the edges in the reverse direction) also as an alternative. In Figure 4(b), let the token value at p_1 be v_{p_1} and the condition be true. The token value at both p_2 and p_3 after t_1 fires is v_{p_2} = v_{p_3} = f_1(v_{p_1}) and the condition is true since t_1 fires unconditionally. After firing of t_3, the token value at p_4 becomes v_{p_4} = f_2(v_{p_2}, v_{p_3}) = f_2(f_1(v_{p_1}), f_1(v_{p_1})) and condition still remains true. When the condition c_4 associated with the transition t_4 is satisfied by v_{p_4}, t_4 fires. After firing of t_4, the token value at p_1 becomes v_{p_1} = f_3(v_{p_4}) = f_3(f_2(f_1(v_{p_1})), f_2(f_1(v_{p_1}))) which is identical to the data transformation rα_3 and the condition of execution Rα_3 is c_1(f_2(f_1(v_{p_1})), f_2(f_1(v_{p_1}))). □

The following theorem captures the uniqueness of the set of paths obtained from the set of (static and dynamic) cut-points.

Theorem 1. For any PRES+ model N, for the set of cut-points obtained by Definition 8 and the token tracking execution of N, the set of paths covering all the transitions is unique.

The proof sketch of the theorem is given below and the detailed proof is given in (Bandyopadhyay, ). Let there be two distinct sets Q_1 and Q_2 of paths where each of the sets covers all the transitions of the given PRES+ model N. Let α = ⟨T_1, T_2, ..., T_n⟩ be a path such that α ∈ Q_1 ∩ Q_2. We argue that any member T_i of α, 1 ≤ i ≤ n, represents the only way to group the transitions of T_i into a maximally parallelizable set and hence conclude that α must be in Q_2 as well. We prove it by induction on i.

Similar to the parallelizable transitions, we can also define parallelizable paths in the same manner.

**Definition 11 (Concatenated Path).** A path α is said to be a concatenated path obtained by concatenation of a path α to a set Q_0 = {α_1, ..., α_k} of parallelizable paths if α ∩ α_i ≠ Ø, 1 ≤ i ≤ k. The path α is denoted as (α_1 || ··· || α_k), where . stands for concatenation operation. The intermediary cut-points \(\bigcup_{1≤i≤k} α_i^c\) ∩ α lose their cut-point designation so that the concatenated path α does not have any intermediary cut-points.

Note that a concatenated path follows the properties of a path. The characterization of a concatenated path is given in (Bandyopadhyay, ).

**Definition 12 (Path Cover).** A finite set of paths Π = \{α_0, α_1, ..., α_k\} is said to be a path cover of a PRES+ model N if any computation µ of any out-port of N can be represented as a sequence of concatenations of parallelizable paths from Π.

In Example 3 of Example 1, it is noted that the set \{α_1, α_2, α_3, α_4, α_5\} of paths which are obtained only from the static cut-points is not a path cover. Whereas, the set \{α_0, α_1, α_2, α_4, α_5, α_6\} of paths which are obtained from both static and dynamic cut-points is a path cover.

**Theorem 2.** Let C be a set of cut-points obtained by Definition 8 and a token tracking execution of a PRES+ model N. The set of paths corresponding to the set C is a path cover of N.

The proof sketch of the theorem is given below and is detailed in (Bandyopadhyay, ). Let µ_p be a computation of an out-port p of the form ⟨T_1, T_2, ..., T_l⟩ where, \(T_i \subseteq \text{inP}, p \in T_i\), T_p \subseteq P_M, 1 ≤ i < l,
where $M_i$ is a marking and $M_{i+1} = M_i^+$, the successor marking of $M_i$, for all $i$, $1 \leq i < l$. The sequence $\mu_i$ can be represented as the sequence $(\mathcal{T}_1, \ldots, \mathcal{T}_l, \mathcal{T}_{l+1}, \ldots, \mathcal{T}_{m+1}, \ldots, \mathcal{T}_l)$, where $\mathcal{T}_j^i$, $1 \leq j \leq m$ and $\mathcal{T}_l^i$ are all members of $C$ (cut-points) and there are no other transitions in the above sequence whose output places are members of $C$.

6 VERIFICATION METHOD

In this section, we describe the formalism of equivalence checking procedure between two out-port cardinality equivalent PRES+ models.

**Definition 13 (Path Equivalence).** Let $N_0$ and $N_1$ be two out-port cardinality-equivalent PRES+ models. A path $\alpha$ of $N_0$ is said to be computationally equivalent to a path $\beta$ of $N_1$, denoted as $\alpha \simeq \beta$, if their data transformation functions are same and their conditions of execution are equivalent, i.e., $\alpha_0 = \beta_0$ and $\alpha_\delta = \beta_\delta$.

**Definition 14 (Corresponding Transitions).** Let $N_0 = (P_0, T_0, I_0, O_0, \text{in}P_0, \text{out}P_0)$ and $N_1 = (P_1, T_1, I_1, O_1, \text{in}P_1, \text{out}P_1)$ be two out-port cardinality-equivalent PRES+ models having the in-port relation $f_{\text{in}} \subseteq \text{in}P_0 \times \text{in}P_1$ and the out-port bijection $f_{\text{out}} : \text{out}P_0 \leftrightarrow \text{out}P_1$. A transition $t$ of $N_0$ corresponds to a transition $t'$ of $N_1$, if

1. $t' \in \text{out}P_1 \Rightarrow (t')^o \in \text{out}P_1$ and $f_{\text{out}}(t^o) = (t')^o$
2. $\exists \alpha, \beta \in \Pi_1$ such that $\alpha \simeq \beta$, $t = \text{last} (\alpha)$ and $t' = \text{last} (\beta)$ where last($\alpha$) = last($\beta$) is the last transition of $\alpha(\beta)$. The set of corresponding transitions is denoted as $\eta_t$;

**Definition 15 (Corresponding Places).** Two places $p$ of $N_0$ and $p'$ of $N_1$ are said to be corresponding if $(p, p') \in f_{\text{in}}(p)$ or $p \in t^o$ and $p' \in (t')^o$, where $(t, t') \in \eta_\delta$. The set of corresponding places is denoted as $\eta_\delta$.

The basic step for path-based equivalence checking is to find a path $\beta$ of $N_1$ for any path $\alpha$ in a path cover $\Pi_0$ of $N_0$ such that $\alpha \simeq \beta$. For the path $\alpha$, the mechanism of selecting the subset of candidate paths of $N_1$ for checking equivalence with $\alpha$ is as follows. If there is a place $p \in \alpha$ such that $(p, p') \in f_{\text{in}}(p)$ where, $p \in \text{in}P_0$ and $p' \in \text{in}P_1$ then the paths from $p'$ in $N_1$ are candidate paths of $\alpha$. Otherwise, the set $\eta_\delta$ of corresponding transitions is used to choose paths of $N_1$ for examining the equivalence. More specifically, for the latter case, the paths leading to $\alpha'$ are identified; let $\alpha'$ be such a path, i.e., $(\alpha')^o \in \alpha$; a transition $t'$ of $N_1$ is found such that $(\text{last}(\alpha'), t') \in \eta_\delta$. Let $T$ be the set of all such transitions $t'$ of $N_1$. Any path $\beta$ satisfying the set equality $T^o = \beta^o$ will be a candidate for checking equivalence with $\alpha$.

Theorem 3. A PRES+ model $N_0$ is contained in another PRES+ model $N_1$, denoted as $N_0 \subseteq N_1$, if there exists a finite path cover $\Pi_0 = \{\alpha_0, 0, \alpha_1, 1, \ldots, \alpha_{l-1}\}$ of $N_0$ for which there exists a set $\Pi_1 = \{\beta_0, 0, \beta_1, 1, \ldots, \beta_{l-1}\}$ $\forall\alpha_0 \subseteq \beta_1$. The proof of the theorem is soundness of equivalence checking algorithm.

6.1 Broad Outline of the Equivalence Checking Algorithm

The basic method for checking equivalence of two PRES+ models consists of the following steps:

1. Introduce static and dynamic cutpoints and hence construct the paths of $N_0$ and $N_1$.
2. Construct the initial path covers $\Pi_0$ of $N_0$ and $\Pi_1$ of $N_1$, comprising paths from a set of cutpoints to another cutpoint without having any intermediate cutpoint. Let $\Pi_0 = \{\alpha_0, 0, \alpha_1, 1, \ldots, \alpha_{l-1}\}$ and $\Pi_1 = \{\beta_0, 0, \beta_1, 1, \ldots, \beta_{l-1}\}$.
3. Show that $\forall \alpha_0 \subseteq \beta_1$. There exists a path $\beta_{1, i}$ of $N_1$ such that $\alpha_{l-1} \simeq \beta_{1, i}$.
4. Let $\Pi_{1,1}^l \subseteq \Pi_1$ be the paths of $N_1$ which have already been found to be equivalent to some paths in $N_0$. $\forall \beta_{1, k} \subseteq \Pi_{1,1}^l$, find its equivalent path of $N_0$.

Step 3 may fail because of code motion transformations where the code segments move beyond the basic block boundaries. In this situation, some paths $\alpha_{l-1} \subseteq \Pi_0$ have no equivalent paths in $N_1$. In such a case either $\alpha_0$, or one of its candidate paths is to be extended till equivalence of the resulting concatenated path(s) are obtained. The idea of path extension is similar to that of path based FSMD equivalence checking mechanism (Banerjee et al., 2014).

Intricacies, however, arise due to presence of paths parallel to the path being extended. The mechanism is illustrated through Example 5 (rather than presenting a formal algorithm). All the functional modules as well as complexity and correctness of the equivalence checking algorithm are presented in (Bandyopadhyay, ).

**Example 5.** Fig 5(a) depicts the PRES+ model $N_0$ for some program. Let the code corresponding to the transition $t_4$ of $N_0$ be moved ahead of $t_3$ and the transitions $t_3$ and $t_5$ be combined into a single transition $t_6$ resulting in a program whose model $N_1$ is depicted in Fig 5(b). In Fig 5(a), the initial path cover
\( \Pi_0 \) of \( \mathcal{N}_0 \) is \( \{ \alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4 \} \) and in Fig 6(b), the initial path cover \( \Pi_i \) of \( \mathcal{N}_i \) is \( \{ \beta_0, \beta_1, \beta_2, \beta_3 \} \). (Note that the computation \( \mu_{0, \Pi_0} \) can be represented as \( (\{ \alpha_0, \alpha_1, \alpha_2 \} | \alpha_3) \alpha_4 \). Similarly, \( \mu_{1, \Pi_1} \) can be represented as \( (\{ \beta_0, \beta_1, \beta_2, \beta_3 \} \beta_4) \). Establishing \( \alpha_3 \simeq \beta_0 \) and \( \alpha_3 \simeq \beta_1 \) is straightforward. For \( \alpha_4 \), the path \( \beta_3 \) is the candidate path because \( \beta_3 \) is the only path some of whose pre-places have correspondence with \( (all) \) the pre-places of \( \alpha_2 \). However, as \( |\beta_3| = 3 < |\alpha_2| = 5 \), an extension of \( \alpha_2 \) is required through all its post-paths. The path \( \alpha_4 \) is the only post path of \( \alpha_2 \). The pre-paths of \( \alpha_4 \) contain \( \alpha_3 \), in addition to \( \alpha_2 \) (which is being extended). The path \( \alpha_3 \) is equivalent to \( \beta_2 \), which is also a pre-path of \( \alpha_2 \)’s candidate path \( \beta_3 \). So, the path \( \alpha_3 \) is extended through the path \( \alpha_4 \) to obtain the path \( \alpha_2, \alpha_3, \alpha_4 \) which is found to be equivalent with \( \beta_3 \). Since \( \alpha_3 \) of all the paths of \( \mathcal{N}_0 \) or their extensions have some equivalent paths in \( \mathcal{N}_1 \), and vice versa, the models \( \mathcal{N}_0 \) and \( \mathcal{N}_1 \) are asserted to be equivalent. □

7 RESULTS

The path construction and the equivalence checking procedures have been implemented in C on a 3.0-GHz Intel(R) Core(TM)2 Duo CPU machine with 2-GB RAM and satisfactorily tested on both sequential (Gupta et al., 2003) and parallel benchmarks. The translation is carried out by one HLS (high level synthesis) compiler, i.e., SPARK (Gupta et al., 2003) and two level thread parallel compilers PLuTo and Par4All. For checking equivalence between two paths the SMT solver has been used (Z3, ). For each of these benchmarks, the original behaviour (in C) is fed to all of the above mentioned compilers to obtain the transformed behaviours (again, in C). In all the cases, the PRES+ models are obtained from the corresponding C code manually. To nullify human errors, each of these models is checked for validity using the CPN tool (Jensen et al., 2007).

### Table 1: Results for several sequential benchmarks.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Trig.</th>
<th>Trasp.</th>
<th>FSMD Eqv. Chk Time (sec) ( [\text{ours}] )</th>
<th>Eqv. Chk Time (sec) ( [\text{Banerjee et al. 2014}] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINMAX</td>
<td>36</td>
<td>36</td>
<td>0.2231</td>
<td>0.2395</td>
</tr>
<tr>
<td>SUMDIGITS</td>
<td>11</td>
<td>10</td>
<td>0.12465</td>
<td>0.07838</td>
</tr>
<tr>
<td>PERFECT</td>
<td>19</td>
<td>14</td>
<td>0.73414</td>
<td>0.69190</td>
</tr>
<tr>
<td>GCD</td>
<td>31</td>
<td>17</td>
<td>0.17267</td>
<td>0.09890</td>
</tr>
<tr>
<td>TLC</td>
<td>39</td>
<td>26</td>
<td>0.75410</td>
<td>0.59801</td>
</tr>
<tr>
<td>DCT</td>
<td>25</td>
<td>18</td>
<td>0.10562</td>
<td>0.04980</td>
</tr>
<tr>
<td>LCM</td>
<td>32</td>
<td>28</td>
<td>0.12687</td>
<td>0.07180</td>
</tr>
<tr>
<td>LRU</td>
<td>42</td>
<td>39</td>
<td>0.52767</td>
<td>0.25980</td>
</tr>
<tr>
<td>PRIMEFAC</td>
<td>12</td>
<td>11</td>
<td>0.92767</td>
<td>0.70980</td>
</tr>
<tr>
<td>MINMAX</td>
<td>35</td>
<td>24</td>
<td>NA</td>
<td>0.07180</td>
</tr>
</tbody>
</table>

### Table 2: Results for several parallel benchmarks.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Original PRES+</th>
<th>Transformed PRES+</th>
<th>Exec. Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINMAX</td>
<td>35</td>
<td>25</td>
<td>0.18721</td>
</tr>
<tr>
<td>LUP</td>
<td>55</td>
<td>50</td>
<td>0.3111</td>
</tr>
<tr>
<td>DEKKER</td>
<td>34</td>
<td>32</td>
<td>0.27131</td>
</tr>
<tr>
<td>PATTERSON</td>
<td>33</td>
<td>28</td>
<td>0.1159</td>
</tr>
</tbody>
</table>

Table 2 shows the equivalence checking time for the parallelizing compilers PLuTo and Par4All. The FSMD equivalence checking method fails to validate these transformations because thread level parallelism...
is not supported by FSMD models. In course of this experiment our equivalence checker has identified a bug of the PLuTo compiler (Bandyopadhyay, ) (possibly due to faulty usage of a variable name in the source program).

8 RELATED WORKS

Translation validation, whereby each individual translation is followed by a validation phase to establish the behavioural equivalence of the source code and the target code, was introduced by Pnueli et al. in (Pnueli et al., 1998) and were demonstrated by Necula in (Necula, 2000) and Rinard et al. (Rinard and Diniz, 1999). This method is further enhanced by Kundu et al. (Kundu et al., 2008) to verify the high-level synthesis tool SPARK capturing parallel execution of statements and Vafeiadis et al. (V. Vafeiadis, 2015) to verify C11 compiler. A bisimulation method for concurrent programs is reported in Milner et al. (Milner, 1989). A major limitation of these methods (Necula, 2000; Kundu et al., 2008; V. Vafeiadis, 2015; Milner, 1989) is that they can verify only structure preserving transformations and fail for schedulers that alter the control structure of a program. To alleviate this shortcoming, a path based equivalence checker for the FSMD model is proposed for sophisticated uniform and non-uniform code motions and code motions across loops (Karfa et al., 2012; Banerjee et al., 2014). They, however, are presently unable to handle loop swapping transformations and also thread-level parallelizing transformations mainly because FSMDs, being a sequential model of computation, cannot capture parallel behaviours straightaway; modeling concurrent behaviours via CDFGs is significantly more complex due to all possible interleavings of the parallel operations. In (S Bandyopadhyay, 2015; Bandyopadhyay et al., 2015), some issues for translation validation of concurrent programs are addressed.

9 CONCLUSION

A formal notion of computation of an untimed PRES+ model is presented. The concept of finite paths capturing computations on the PRES+ model has been incorporated based on a notion of dynamic cut-points. The path construction and path based equivalence checking methods are described through two examples. The implementation is satisfactorily tested on a set of fifteen benchmark problems encompassing the various speculative and non speculative code optimization techniques (Karfa et al., 2012) as well as thread level loop parallelizing transformations for scalar programs.

The limitations of the present work is that it cannot handle loop-shifting, software pipelining based transformations as well as several loop transformations for array handling programs. Being a value based model, it captures data-path more vividly. In course of code transformation, only data movement takes place. If we consider only those portions of the model where the exact code transformations have taken place, the present method can be fast and scalable. Hence, scalability analysis for this method is one of our future goals. Enhancing the equivalence checking procedure to encompass the limitations mentioned above seems to be a promising future endeavor; investigating alternate proof based verification techniques, such as (Lengauer, 2011), can be useful in this regard.

ACKNOWLEDGEMENTS

This work is supported by TCS research fellowship of Soumyadip Bandyopadhyay and DST Project No: SB/EMEQ–281/2013.

REFERENCES


