Deriving Bisimulation Relations from Path Extension Based Equivalence Checkers

Kunal Banerjee, MIEEE, Dipankar Sarkar and Chittaranjan Mandal, SMIEEE

Abstract—Constructing bisimulation relations between programs as a means of translation validation has been an active field of study. The problem is in general undecidable. Currently available mechanisms suffer from drawbacks such as non-termination and significant restrictions on the structures of programs to be checked. We have developed a path extension based equivalence checking method as an alternative translation validation technique to alleviate these drawbacks. In this work, path extension based equivalence checking of programs (flowcharts) is leveraged to establish a bisimulation relation between a program and its translated version by constructing the relation from the outputs of the equivalence checker.

Index Terms—Translation validation, bisimulation relation, equivalence checking, path extension based equivalence checker.

1 Introduction

Translation validation was proposed by Pnueli et al. in [1] as a new approach for the verification of compilers whereby, each individual translation is followed by a validation phase which verifies that the target code produced correctly implements the source code, rather than proving in advance that the code produced by the compiler is always correct by construction. A related technique was also developed by Rinard [2]. Translation validation for an optimizing compiler by obtaining simulation relations between programs and their translated versions was first demonstrated by Necula in [3]. The procedure broadly consists of two algorithms – an inference algorithm and a checking algorithm. The inference algorithm collects a set of constraints (representing the simulation relation) in a forward scan of the two programs and then the checking algorithm checks the validity of these constraints. This work is enhanced by Kundu et al. [4], [5] to validate the High-Level Synthesis (HLS) process. Unlike Necula's approach, Kundu et al.'s procedure uses a general theorem prover, rather than specialized solvers and simplifiers, and is thus more modular. A limitation of these methods [3], [4], [5] is that they cannot handle non-structure preserving transformations such as those introduced by path based schedulers [6], [7]; in other words, the control structures of the source and the target programs must be identical if one were to apply these methods. This limitation is alleviated to some extent by Li et al. [8] using knowledge of the scheduling mechanism.

An alternative approach for proving equivalence of programs represented as Finite State Machines with Datapath (FSMDs) [9] was proposed by Karfa et al. in [10] which does not require any additional information from the compiler. Further enhancements were reported in [11], [12], [13], so that application of structure altering scheduling and code optimization techniques can be verified. Many of these optimizations, such as [7], are not handled by bisimulation based verification methods mentioned above. A related method for translation validation is proposed in [14] which uses the CASM language [15] to represent the source and the target programs and then uses a specialized solver to establish their equivalence. The validation mechanism, however, needs "witness information" from each pass of the compiler describing the performed transformations. Fundamentally, in these methods, the FSMDs corresponding to the source and the target codes are viewed as a combination of finite set of control paths (with the loops cut at their entry states). It is important to note that we use the term "path" in this work to mean a "control path" in a program unless otherwise stated. This method basically constructs the finite sets P_1 and P_2, say, of paths of the two FSMDs M_1 and M_2, respectively so that any computation of M_1 (M_2) can be captured by concatenation of the members of P_1 (P_2); accordingly, the set P_1 (P_2) is called a (finite) path cover of M_1 (M_2); the method then shows that for each path in P_1, there is an equivalent path in P_2, and vice versa. To take care of optimizing code motions, original paths are extended judiciously in either M_1 or M_2 [11]. The prowess of the path extension based translation validation method is exhibited during verification of the high-level synthesis process of digital circuits [16].

However, transformations such as loop shifting [17] that can be handled by bisimulation based methods of [4], [5] by repeated strengthening of the simulation relation over the related loops of the source and the target codes still elude the path based methods. This process of strengthening the simulation relation iteratively until a fixed point is reached (i.e., the relation becomes strong enough to imply the equivalence), however, may not terminate. On the contrary, a single pass is used to determine the equivalence/non-equivalence for a pair of paths in the path based approaches and hence, the number of paths in the path cover of a program being finite, these methods are guaranteed to ter-

• *K. Banerjee is with Intel Parallel Computing Lab, Bangalore, India. This work, however, was largely done when he was with the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India.*
E-mail: kunal.banerjee@intel.com
• *D. Sarkar, and C. Mandal are with the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India.*
E-mail: ds@cse.iitkgp.ernet.in; chitta@iitkgp.ac.in

minate. Thus, we find that both bisimulation and path extension based approaches have their own merits and demerits, and therefore, both have found application in the field of translation validation of untrusted compilers. However, the conventionality of bisimulation as the approach for equivalence checking raises the natural question of examining whether path extension based equivalence checking yields a bisimulation relation or not.

In this work, we define the notion of simulation/bisimulation relation in the context of the FSMD model. We then show that a bisimulation relation can be constructed whenever two FSMDs are found to be equivalent by path extension based equivalence checker.

The paper is organized as follows. Section 2 introduces the FSMD model and the simulation/bisimulation relations for FSMD models. Section 3 describes the method of deriving (bi)simulation relation between two FSMDs from the outputs of a path extension based equivalence checker. The paper is concluded in section 4.

2 The FSMD model and simulation/bisimulation relation between FSMD models

The FSMD model for equivalence checking is described in brief; a detailed description can be found in [11]. The FSMD model [9] is formally defined as an ordered tuple $\langle Q, q_0, I, V, O, \tau : Q \times 2^{\mathcal{S}} \rightarrow Q, h : Q \times 2^{\mathcal{S}} \rightarrow U \rangle$, where Q is the finite set of control states, q_0 is the reset state, I is the set of input variables, V is the set of storage variables, O is the set of output variables, τ is the state transition function, $\mathcal{S}$ is a set of binary predicates (relations between arithmetic expressions on variables in I and V), U represents a set of assignments of expressions over inputs and storage variables to some storage or output variables and h is the update function capturing the conditional updates of the output and storage variables taking place in the transitions through the members of U. It may be noted that $\mathcal{S}$ contains the conditional guards on the state transitions, which are of the form $e_1 \ \rho \ e_2$, where ρ represents a relational predicate (e.g., $=, \neq, \geq$) and e_1 and e_2 are two arithmetic expressions.

Let $\overline{v}$ denote a vector (an ordered tuple) of variables of I and V assuming values from the domain $D_{\overline{v}} = \Pi_{v \in I \cup V} D_v$, where D_v is the domain of the variable v and Π represents the Cartesian product of sets. Let $q(\overline{v}), q \in Q$, denote the set of values assumed by the variables in the state q. Each element of $q(\overline{v})$ is called a data state at q and is denoted as σ_q. Let $\Sigma = \bigcup_{q \in Q} \{\sigma_q\}$ be the set of all the data states.

An FSMD can be represented in a natural way as a digraph with the control states as the vertices and the labeled edges capturing the transition function τ and the update function h. Specifically, if there are two control states q_i and q_j such that $\tau(q_i, c(\overline{v})) = q_j$ and $h(q_i, c(\overline{v}))$ is represented as $\overline{v} \Leftarrow f(\overline{v})$, then there is an edge from q_i to q_j with label $c(\overline{v})/\overline{v} \Leftarrow f(\overline{v})$ in the digraph representation of the FSMD. A transition may, therefore, be represented as a 4-tuple $\langle q_i, c_{i,k}, h_{i,k}, q_k \rangle$, where q_i is the initial control state of the transition, q_k is its final control state, $c_{i,k}$ is the condition of execution of the transition and $h_{i,k}$ represents the update of the storage variables and output variables that results from execution of the transition. Often we abbreviate such a transition as $\langle q_i, q_k \rangle$ when other components are obvious or of no concern in the context. A transition which updates a storage variable with an input is called an *input transition*. A transition which updates an output variable is called an *output transition*. A control state along with a data state together constitute a *configuration* as defined below.

Definition 2.1 (Configuration) *Given an FSMD $M = \langle Q, q_0, I, V, O, \tau, h \rangle$, we define a configuration to be a pair $\langle q, \sigma \rangle$, where $q \in Q$ and $\sigma \in q(\overline{v})$.*

Definition 2.2 (Execution Sequence) *For a given FSMD $M = \langle Q, q_0, I, V, O, \tau, h \rangle$, an execution sequence η is a sequence of configurations of the form $\langle \langle q_{i_0}, \sigma_{i_0} \rangle, \langle q_{i_1}, \sigma_{i_1} \rangle, \ldots, \langle q_{i_n}, \sigma_{i_n} \rangle \rangle$, such that $\forall j, 0 \leq j \leq n$, there is a transition $\langle q_{i_j}, c_{i_j,i_{j+1}}, h_{i_j,i_{j+1}}, q_{i_{j+1}} \rangle$ and $\sigma_{i_j} \in q_{i_j}(\overline{v})$, where $c_{i_j,i_{j+1}}(\sigma_{i,j})$ is true, $h_{i_j,i_{j+1}}(q_{i_j}, c_{i_j,i_{j+1}}(\overline{v})) \in U$ and $h_{i_j,i_{j+1}}(q_{i_j}, c_{i_j,i_{j+1}}(\sigma_{i_j}))$ defines the update operation by which the data state $\sigma_{i_{j+1}}$ is obtained from σ_{i_j}.*

Existence of such an execution sequence η is denoted as $\langle q_{i_0}, \sigma_{i_0} \rangle \rightsquigarrow^{\eta} \langle q_{i_n}, \sigma_{i_n} \rangle$. We denote by $\mathcal{N}_i$ the set of all execution sequences in FSMD M_i. From now onward we speak of two FSMDs $M_1 = \langle Q_1, q_{1,0}, I, V_1, O, \tau_1, h_1 \rangle$ and $M_2 = \langle Q_2, q_{2,0}, I, V_2, O, \tau_2, h_2 \rangle$ whose behaviours we seek to examine for equivalence; $V_1 \cap V_2 \neq \Phi$. Two output transitions $\langle q_{1,i}, q_{1,k} \rangle$ of M_1 and $\langle q_{2,j}, q_{2,l} \rangle$ of M_2 are said to be equivalent if they update the same output variable(s) with the same values.

Definition 2.3 (Equivalence of Execution Sequences) *For any execution sequence η_1 (η_2) of M_1 (M_2), let the sequence of output transitions be denoted as $\eta_1|_O$ ($\eta_2|_O$). For an output variable $v \in O$, let $\eta_1|_{O,v}$ ($\eta_2|_{O,v}$) denote the sequence of values output (or synonymously, the output list) on the variable v over the execution sequence η_1 (η_2). Two execution sequences η_1 of M_1 and η_2 of M_2 are said to be equivalent if for any variable $v \in O$, $\eta_1|_{O,v} = \eta_2|_{O,v}$.*

For the subsequent discussion, we suffix the (set of) data states in some control state of the machine $M_i, i \in \{1, 2\}$, in a natural way. Similarly, we rename the variables occurring in FSMD M_i by adding the suffix i. Thus, for machine $M_i, i \in \{1, 2\}$, $\sigma_i \in \Sigma_i$ represents a data state of M_i and $\sigma_{i,j} \in q_{i,j}(\overline{v_i})$ represents a data state at the control state $q_{i,j}$. A verification relation between two FSMDs M_1 and M_2 is a set of triples $\langle q_{1,i}, q_{2,j}, \phi_{i,j} \rangle$, where $q_{1,i} \in Q_1$, $q_{2,j} \in Q_2$, $\phi_{i,j} \subseteq q_{1,i}(\overline{v_1}) \times q_{2,j}(\overline{v_2})$. The notation $\phi_{i,j}(\sigma_{1,i}, \sigma_{2,j}) = true$ indicates that $\phi_{i,j}$ is satisfied by the data states $\sigma_{1,i}$ of M_1 and $\sigma_{2,j}$ of M_2. The predicate $\phi_{i,j}$ is a formula over variables appearing in FSMD M_1 and FSMD M_2 at the states $q_{1,i} \in Q_1$ and $q_{2,j} \in Q_2$ where the only atomic predicates allowed are equalities between their variables (and not their values). It is important to note that $\phi_{0,0}$ is identically $true$ because a computation is assumed to define always storage variables (which are later used) through some input transitions. Based on the above discussions, a simulation relation can be defined as follows (in accordance with the definition of simulation relation given in [5]).

Definition 2.4 (Simulation Relation) *A simulation relation S for two FSMDs $M_1 = \langle Q_1, q_{1,0}, I, V_1, O, \tau_1, h_1 \rangle$ and $M_2 = \langle Q_2, q_{2,0}, I, V_2, O, \tau_2, h_2 \rangle$ is a verification relation which*

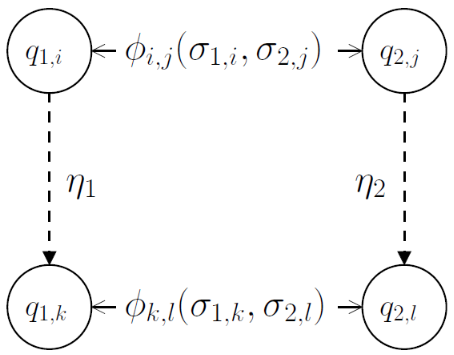
$$\begin{tikzcd}
q_{1,i} \arrow[d, dashed, "\eta_1"] & \phi_{i,j}(\sigma_{1,i},\sigma_{2,j}) \arrow[l] \arrow[r] & q_{2,j} \arrow[d, dashed, "\eta_2"'] \\
q_{1,k} & \phi_{k,l}(\sigma_{1,k},\sigma_{2,l}) \arrow[l] \arrow[r] & q_{2,l}
\end{tikzcd}$$

Fig. 1: An example showing two execution sequences from two FSMDs.

satisfies the following two clauses:
1. $S(q_{1,0}, q_{2,0},\ true)$, *and*
2. $\forall q_{1,i}, q_{1,k} \in Q_1, \sigma_{1,i} \in q_{1,i}(\overline{v}), \sigma_{1,k} \in q_{1,k}(\overline{v}),\ \eta_1 \in \mathcal{N}_1, q_{2,j} \in Q_2, \sigma_{2,j} \in q_{2,j}(\overline{v})$

$$\begin{aligned}&[\ \langle q_{1,i}, \sigma_{1,i}\rangle \rightsquigarrow^{\eta_1} \langle q_{1,k}, \sigma_{1,k}\rangle \wedge \phi_{i,j}(\sigma_{1,i}, \sigma_{2,j}) \ \wedge \\ &\quad S(q_{1,i}, q_{2,j}, \phi_{i,j}) \Rightarrow \\ &\qquad \exists q_{2,l} \in Q_2, \sigma_{2,l} \in q_{2,l}(\overline{v}), \eta_2 \in \mathcal{N}_2 \\ &\qquad\quad \{\ \langle q_{2,j}, \sigma_{2,j}\rangle \rightsquigarrow^{\eta_2} \langle q_{2,l}, \sigma_{2,l}\rangle \wedge \eta_1 \equiv \eta_2\ \wedge \\ &\qquad\qquad \phi_{k,l}(\sigma_{1,k}, \sigma_{2,l})\ \wedge\ S(q_{1,k}, q_{2,l}, \phi_{k,l})\ \}\ ].\end{aligned}$$

Intuitively, the two clauses mentioned above state that: (i) the reset state of FSMD M_1 must be related to the reset state of FSMD M_2 for all data states, and (ii) if two states of M_1 and M_2 with their respective data states are related and M_1 can proceed along an execution sequence η_1, then M_2 must also be able to proceed along an execution sequence η_2 such that $\eta_1 \equiv \eta_2$ and the end states of the two execution sequences must also be related. Fig. 1 depicts the second clause of the definition diagrammatically. We now define a *bisimulation relation* using the definition of simulation relation.

Definition 2.5 (Bisimulation Relation) *A verification relation B for two FSMDs M_1 and M_2 is a bisimulation relation between them iff B is a simulation relation for M_1, M_2 and $B^{-1} = \{(q_{2,j}, q_{1,i}, \phi) \mid B(q_{1,i}, q_{2,j}, \phi)\}$ is a simulation relation for M_2, M_1.*

3 Deriving simulation relation from the output of a path extension based equivalence checker

A *computation* of an FSMD is a finite walk from the reset state back to itself without having any intermediary occurrence of the reset state. The condition of execution R_μ of a computation μ is a symbolic logical expression over the variables in I such that R_μ is satisfied by the initial data state iff the computation μ is executed. The data transformation r_μ of the computation μ is the tuple $\langle \mathsf{s}_\mu, \theta_\mu\rangle$; the first member s_μ represents the symbolic expression values of the program variables in V at the end of the computation μ in terms of the input variables I and the second member θ_μ represents the output list (of sequences of symbolic output expressions over the variables of I – one sequence corresponding to each output variable) of the computation μ. Two computations μ_1 and μ_2 of an FSMD are said to be equivalent, denoted as $\mu_1 \simeq \mu_2$, if $R_{\mu_1} \equiv R_{\mu_2}$ and $r_{\mu_1} = r_{\mu_2}$.

Definition 3.1 (FSMD Containment) *An FSMD M_1 is said to be contained in an FSMD M_2, symbolically $M_1 \sqsubseteq M_2$, if for any computation μ_1 of M_1 on some inputs, there exists a computation μ_2 of M_2 on the same inputs such that $\mu_1 \simeq \mu_2$.*

Definition 3.2 (FSMD Equivalence) *Two FSMDs M_1 and M_2 are said to be computationally equivalent, if $M_1 \sqsubseteq M_2$ and $M_2 \sqsubseteq M_1$.*

However, an FSMD may consist of infinite number of indefinitely long computations because of presence of loops. So, the idea of directly comparing the computations of the FSMDs exhaustively will not work in practice. Therefore, cut-points are introduced such that each loop is cut in at least one cut-point thereby permitting an FSMD to be considered as a combination of paths.

A *path* α in an FSMD model is a finite sequence of states where the first and the last states are cut-points and there are no intermediary cut-points and any two consecutive states in the sequence are in τ. The initial (start) and the final control states of a path α are denoted as α^s and α^f, respectively. A path α is characterized by an ordered pair $\langle R_\alpha, r_\alpha\rangle$, where R_α is the condition of execution of α satisfying the property that $\forall \sigma_{\alpha^s} \in \Sigma$, $R_\alpha(\sigma_{\alpha^s})$ implies that the path α is executed if control reaches the state α^s; the second member $r_\alpha = \langle \mathsf{s}_\alpha, \theta_\alpha\rangle$, where s_α is the functional transformation of the path α, i.e., $\sigma_{\alpha^f} = \mathsf{s}_\alpha(\sigma_{\alpha^s})$ and θ_α represents the output list of data values of some variables produced along the path α.

Definition 3.3 (Path Equivalence) *Two paths α and β are said to be computationally equivalent, denoted as $\alpha \simeq \beta$, iff $R_\alpha \equiv R_\beta$ and $r_\alpha = r_\beta$.*

It is worth noting that if two behaviors are to be computationally equivalent, then their outputs must match. So, when some variable is output, its counterpart in the other FSMD must attain the same value. In other words, equivalence of θ_α hinges upon the equivalence of s_α. Hence, the rest of the paper focuses on computation of s_α; the computation of θ_α has deliberately been omitted for the sake of brevity.

Any computation μ of an FSMD M can be considered as a computation along some concatenated path $[\alpha_1\alpha_2\alpha_3...\alpha_k]$ of M such that, for $1 \le i < k$, α_i terminates in the initial state of the path α_{i+1}, the path α_1 emanates from and the path α_k terminates in the reset state q_0 of M; α_i's may not all be distinct. Hence, we have the following definition.

Definition 3.4 (Path Cover of an FSMD) *A finite set of paths $P = \{\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_k\}$ is said to be a path cover of an FSMD M if any computation μ of M can be expressed as a concatenation of paths from P.*

The set of all paths from a cut-point to another cut-point without having any intermediary cut-point is a path cover of the FSMD [18]. In course of establishing equivalence of paths of a path cover, a path based equivalence checker produces a natural correspondence between the control states from two FSMDs as defined below.

Definition 3.5 (Corresponding States) *Let $M_1 = \langle Q_1, q_{1,0}, I, V_1, O, \tau_1, h_1\rangle$ and $M_2 = \langle Q_2, q_{2,0}, I, V_2, O, \tau_2, h_2\rangle$ be two FSMDs having identical input and output sets, I and O, respectively.*

1) *The respective reset states $q_{1,0}$ and $q_{2,0}$ are corresponding states and a non-reset state does not have correspondence with a reset state.*
2) *If $q_{1,i} \in Q_1$ and $q_{2,j} \in Q_2$ are corresponding states and there exist $q_{1,k} \in Q_1$ and $q_{2,l} \in Q_2$ such that, for some path α from $q_{1,i}$ to $q_{1,k}$ in M_1, there exists a path β from $q_{2,j}$ to $q_{2,l}$ in M_2 such that $\alpha \simeq \beta$, then $q_{1,k}$ and $q_{2,l}$ are corresponding states (note that $q_{1,k}$ and $q_{2,l}$ must both be either reset states or non-reset states).*

Theorem 3.1 *An FSMD M_1 is contained in another FSMD M_2 ($M_1 \sqsubseteq M_2$), if there exists a finite path cover $P_1 = \{\alpha_1, \alpha_2, \ldots, \alpha_{l_1}\}$ of M_1 for which there exists a set $P_2 = \{\beta_1, \beta_2, \ldots, \beta_{l_2}\}$ of paths of M_2 such that for any corresponding state pair $\langle q_{1,i}, q_{2,j}\rangle$, for any path $\alpha_m \in P_1$ emanating from $q_{1,i}$, there exists a path $\beta_n \in P_2$ emanating from $q_{2,j}$ such that $\alpha_m \simeq \beta_n$.*

Proof: We say that $M_1 \sqsubseteq M_2$, if for any computation μ_1 of M_1 on some inputs, there exists a computation μ_2 of M_2 on the same inputs such that $\mu_1 \simeq \mu_2$ (Definition 3.1). Let there exist a finite path cover $P_1 = \{\alpha_1, \alpha_2, \ldots, \alpha_{l_1}\}$ of M_1. Corresponding to P_1, let a set $P_2 = \{\beta_1, \beta_2, \ldots, \beta_{l_2}\}$ of paths of M_2 exist such that for any corresponding state pair $\langle q_{1,i}, q_{2,j}\rangle$, for any path $\alpha_m \in P_1$ emanating from $q_{1,i}$, there exists a path $\beta_n \in P_2$ emanating from $q_{2,j}$ such that $\alpha_m \simeq \beta_n$.

Since P_1 is a path cover of M_1, any computation μ_1 of M_1 can be looked upon as a concatenated path $[\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_t}]$ from P_1 starting from the reset state $q_{1,0}$ and ending again at this reset state of M_1. Now, we have to show that a computation μ_2 exists in M_2 such that $\mu_1 \simeq \mu_2$.

The reset states $q_{1,0}$ of M_1 and $q_{2,0}$ of M_2 must be corresponding states by clause 1 of Definition 3.5. Therefore, it follows from the hypothesis that a path β_{j_1} exists in P_2 such that $\alpha_{i_1} \simeq \beta_{j_1}$; thus, the states $\alpha^f_{i_1}$ and $\beta^f_{j_1}$ must again be corresponding states by clause 2 in Definition 3.5. By repetitive applications of the above argument, it follows that that there exists a concatenated sequence of paths $\beta_{j_1}\ldots\beta_{j_t}$ such that $\alpha_{i_k} \simeq \beta_{j_k}, 1 \leq k \leq t$. What remains to be proved for $[\beta_{j_1}\beta_{j_2}\ldots\beta_{j_t}]$ to be a computation of M_2 is that $\beta^f_{j_t} = q_{2,0}$. Let $\beta^f_{j_t} \neq q_{2,0}$; now $\langle\alpha^f_{i_t}, \beta^f_{j_t}\rangle$, i.e., $\langle q_{1,0}, \beta^f_{j_t}\rangle$ must be a corresponding state pair. However, by Definition 3.5, a non-reset state cannot have correspondence with a reset state. Consequently, $\beta^f_{j_t}$ must be $q_{2,0}$ and thus $[\beta_{j_1}\beta_{j_2}\ldots\beta_{j_t}]$ is a computation, μ_2 say, and $\mu_1 \simeq \mu_2$. ■

Example 3.1 Fig. 2 gives an example where Theorem 3.1 is used to establish equivalence between two FSMDs. Fig. 2(a) shows the source FSMD and Fig. 2(b) shows the transformed FSMD. Both of these FSMDs compute the greatest common divisor of two numbers. We use the notation $q_i \twoheadrightarrow q_j$ to represent a path from the state q_i to the state q_j. To distinguish between paths originating from the same state due to conditional branches, we write $q_i \xrightarrow{c} q_j$ to denote the path from q_i to q_j which is traversed when the condition c is satisfied.

Considering the reset state and all other states with more than one outgoing edges as the cut-points, the initial set of cut-points in M_1 is $\{q_{1,0}, q_{1,1}, q_{1,2}, q_{1,3}, q_{1,4}, q_{1,5}\}$ and the initial path cover $P'_1 = \alpha_1 : q_{1,0} \twoheadrightarrow q_{1,1}$, $\alpha_2 : q_{1,1} \xrightarrow{y1=y2} q_{1,0}$, $\alpha_3 : q_{1,1} \xrightarrow{!(y1=y2)} q_{1,2}$, $\alpha_4 : q_{1,2} \xrightarrow{even(y1)} q_{1,3}$, $\alpha_5 : q_{1,2} \xrightarrow{!even(y1)} q_{1,4}$, $\alpha_6 : q_{1,3} \xrightarrow{even(y2)} q_{1,1}$, $\alpha_7 : q_{1,3} \xrightarrow{!even(y2)} q_{1,1}$, $\alpha_8 : q_{1,4} \xrightarrow{even(y2)} q_{1,1}$, $\alpha_9 : q_{1,4} \xrightarrow{!even(y2)} q_{1,5}$, $\alpha_{10} : q_{1,5} \xrightarrow{y1>y2} q_{1,1}$, $\alpha_{11} : q_{1,5} \xrightarrow{!(y1>y2)} q_{1,1}$.
Following a similar choice for cut-points in M_2, we get the initial set of cut-points as $\{q_{2,0}, q_{2,1}\}$ and the initial path cover $P'_2 = \beta_1 : q_{2,0} \twoheadrightarrow q_{2,1}$, $\beta_2 : q_{2,1} \xrightarrow{y1=y2} q_{2,0}$, $\beta_3 : q_{2,1} \xrightarrow{!(y1=y2)\&even(y1)\&even(y2)} q_{2,1}$, $\beta_4 : q_{2,1} \xrightarrow{!(y1=y2)\&even(y1)\&!even(y2)} q_{2,1}$, $\beta_5 : q_{2,1} \xrightarrow{!(y1=y2)\&!even(y1)\&even(y2)} q_{2,1}$, $\beta_6 : q_{2,1} \xrightarrow{!(y1=y2)\&!even(y1)\&!even(y2)\&y1>y2} q_{2,1}$, $\beta_7 : q_{2,1} \xrightarrow{!(y1=y2)\&!even(y1)\&!even(y2)\&!(y1>y2)} q_{2,1}$.

Note that upon finding a mismatch between a pair of paths, the one with the weaker condition of execution is extended. The algorithm reported in [11] proceeds in the following sequence:
1) finds β_1 as the equivalent path of α_1;
2) finds β_2 as the equivalent path of α_2;
3) fails to find equivalent path of α_3; also, the paths β_3, β_4, β_5, β_6 and β_7 of M_2 emanating from $q_{2,1}$, the corresponding state of $q_{\alpha^s_3}$, are such that $R_{\beta_i} \implies R_{\alpha_3}$, $3 \leq i \leq 7$; for the remaining paths of M_2 emanating from $q_{2,1}$, the conditions of execution cannot hold when R_{α_3} holds. Hence the path α_3 having a weaker condition of execution is extended in all possible ways with the hope that the extended paths will have stronger conditions of execution and their equivalent paths may be found from among the paths $\beta_3, \ldots, \beta_7$. The extended paths are obtained as $\alpha_3\alpha_4$ and $\alpha_3\alpha_5$;
4) fails to find equivalent path of $\alpha_3\alpha_4$; hence extends it along similar lines; the extended paths are $\alpha_3\alpha_4\alpha_6$ and $\alpha_3\alpha_4\alpha_7$;
5) and 6) finds β_4 and β_3 as the respective equivalent paths of $\alpha_3\alpha_4\alpha_6$ and $\alpha_3\alpha_4\alpha_7$;
7) fails to find equivalent path of $\alpha_3\alpha_5$, hence, extends it; the extended paths are $\alpha_3\alpha_5\alpha_8$ and $\alpha_3\alpha_5\alpha_9$;
8) finds β_6 as the equivalent path of $\alpha_3\alpha_5\alpha_8$;
9) fails to find equivalent path of $\alpha_3\alpha_5\alpha_9$, hence, extends it; the extended paths are $\alpha_3\alpha_5\alpha_9\alpha_{10}$ and $\alpha_3\alpha_5\alpha_9\alpha_{11}$;
10) and 11) find β_5 and β_7 as the respective equivalent paths of $\alpha_3\alpha_5\alpha_9\alpha_{10}$ and $\alpha_3\alpha_5\alpha_9$ α_{11}.

Thus, we obtain the respective path covers of the source and the transformed FSMDs such that there is a one-to-one correspondence between their members in terms of path equivalence as given below:
$P_1 = \{$ $q_{1,0} \twoheadrightarrow q_{1,1}$,
$q_{1,1} \xrightarrow{y1=y2} q_{1,0}$,
$q_{1,1} \xrightarrow{!(y1=y2)} q_{1,2} \xrightarrow{even(y1)} q_{1,3} \xrightarrow{even(y2)} q_{1,1}$,
$q_{1,1} \xrightarrow{!(y1=y2)} q_{1,2} \xrightarrow{even(y1)} q_{1,3} \xrightarrow{!even(y2)} q_{1,1}$,
$q_{1,1} \xrightarrow{!(y1=y2)} q_{1,2} \xrightarrow{!even(y1)} q_{1,4} \xrightarrow{even(y2)} q_{1,1}$,
$q_{1,1} \xrightarrow{!(y1=y2)} q_{1,2} \xrightarrow{!even(y1)} q_{1,4} \xrightarrow{!even(y2)}$

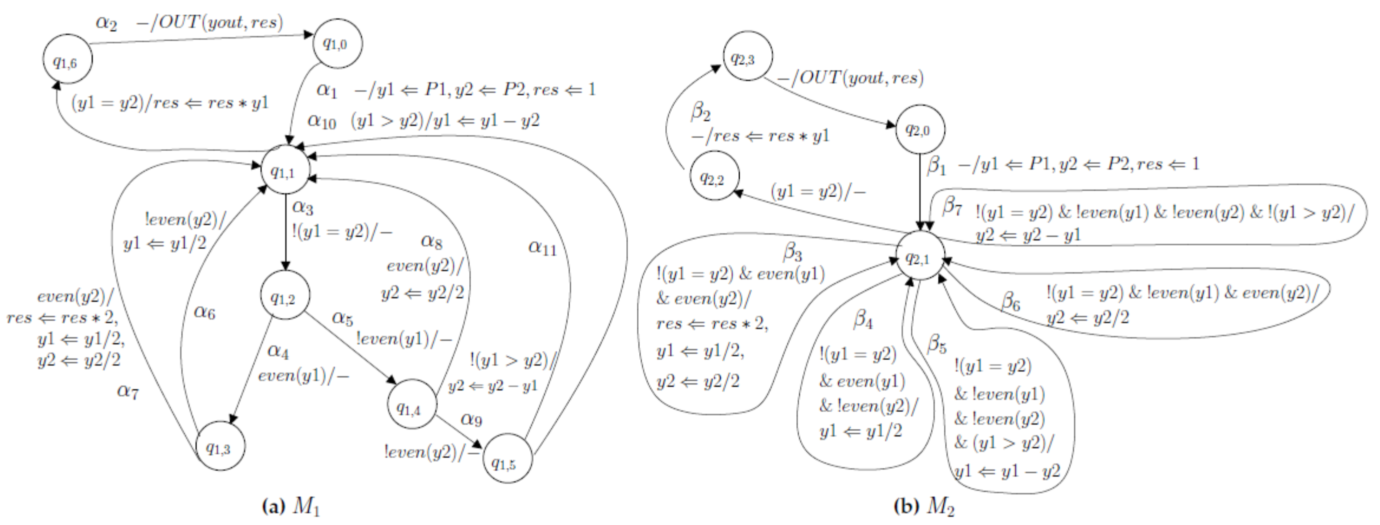
α_2 $-/OUT(yout, res)$
$(y1 = y2)/res \Leftarrow res * y1$
α_1 $-/y1 \Leftarrow P1, y2 \Leftarrow P2, res \Leftarrow 1$
α_{10} $(y1 > y2)/y1 \Leftarrow y1 - y2$
α_3 $!(y1 = y2)/-$
$!even(y2)/$ $y1 \Leftarrow y1/2$
α_8 $even(y2)/$ $y2 \Leftarrow y2/2$
α_{11}
$even(y2)/$ $res \Leftarrow res * 2,$ $y1 \Leftarrow y1/2,$ $y2 \Leftarrow y2/2$
α_6
α_5 $!even(y1)/-$
α_4 $even(y1)/-$
$!(y1 > y2)/$ $y2 \Leftarrow y2 - y1$
α_7
α_9 $!even(y2)/-$
(a) M_1
$-/OUT(yout, res)$
β_2 $-/res \Leftarrow res * y1$
β_1 $-/y1 \Leftarrow P1, y2 \Leftarrow P2, res \Leftarrow 1$
$(y1 = y2)/-$
β_7 $!(y1 = y2)$ & $!even(y1)$ & $!even(y2)$ & $!(y1 > y2)/$ $y2 \Leftarrow y2 - y1$
β_3 $!(y1 = y2)$ & $even(y1)$ & $even(y2)/$ $res \Leftarrow res * 2,$ $y1 \Leftarrow y1/2,$ $y2 \Leftarrow y2/2$
β_6 $!(y1 = y2)$ & $!even(y1)$ & $even(y2)/$ $y2 \Leftarrow y2/2$
β_4 $!(y1 = y2)$ & $even(y1)$ & $!even(y2)/$ $y1 \Leftarrow y1/2$
β_5 $!(y1 = y2)$ & $!even(y1)$ & $!even(y2)$ & $(y1 > y2)/$ $y1 \Leftarrow y1 - y2$
(b) M_2

Fig. 2: (a) M_1: Source FSMD. (b) M_2: Transformed FSMD.

$q_{1,5} \xrightarrow{y1>y2} q_{1,1}$,
$q_{1,1} \xrightarrow{!(y1=y2)} q_{1,2} \xrightarrow{!even(y1)} q_{1,4} \xrightarrow{!even(y2)}$
$q_{1,5} \xrightarrow{!(y1>y2)} q_{1,1}$ },

$P_2 = \{\ q_{2,0} \twoheadrightarrow q_{2,1}$,
$q_{2,1} \xrightarrow{y1=y2} q_{2,0}$,
$q_{2,1} \xrightarrow{!(y1=y2)\&even(y1)\&even(y2)} q_{2,1}$,
$q_{2,1} \xrightarrow{!(y1=y2)\&even(y1)\&!even(y2)} q_{2,1}$,
$q_{2,1} \xrightarrow{!(y1=y2)\&!even(y1)\&even(y2)} q_{2,1}$,
$q_{2,1} \xrightarrow{!(y1=y2)\&!even(y1)\&!even(y2)\&y1>y2} q_{2,1}$,
$q_{2,1} \xrightarrow{!(y1=y2)\&!even(y1)\&!even(y2)\&!(y1>y2)} q_{2,1}\ \}$.

The set of corresponding states for this example is $\{\langle q_{1,0}, q_{2,0}\rangle, \langle q_{1,1}, q_{2,1}\rangle\}$. As demonstrated through this example, to get to the path covers and the set of corresponding states for two FSMDs M_1 and M_2, one may need to revise one's initial choice of cut-points by employing path extension and the extendability of a path in an FSMD depends upon whether its condition of execution is stronger/weaker than that of its counterpart in the other FSMD. The details of obtaining the suitable path covers using the path extension method can be found in [11]. ■

Let the relation of corresponding states be denoted as δ. It may be noted that from Definition 3.5 of the state correspondence relation δ, a pair $\langle q_{1,k}, q_{2,l}\rangle$ will be in δ even if only one pair of paths of the form $\langle \alpha : q_{1,i} \twoheadrightarrow q_{1,k}$ in $M_1, \beta : q_{2,j} \twoheadrightarrow q_{2,l}$ in $M_2\rangle$ is found, where $\langle q_{1,i}, q_{2,j}\rangle \in \delta$, such that $\alpha \simeq \beta$. If, however, the path based equivalence checker indeed ascertains that $M_1 \sqsubseteq M_2$, then there surely exists some subset of δ such that all pairs of paths (possibly through extension) from one such pair to another in the subset comprise equivalent paths. The subset is formally defined as follows.

Definition 3.6 (Perfectly Corresponding States) *Let the path based equivalence checker ascertain that $M_1 \sqsubseteq M_2$ and produce path covers P_1 of M_1 and P_2 of M_2 where for each path $\alpha \in P_1$, there exists a path $\beta \in P_2$ such that $\alpha \simeq \beta$. Then two states $q_{1,k}$ of M_1 and $q_{2,l}$ of M_2 are said to be perfectly corresponding states if $\langle q_{1,k}, q_{2,l}\rangle \in \delta$ and $\forall \alpha \in P_1[\alpha^f = q_{1,k} \Rightarrow \exists \beta \in P_2(\beta^f = q_{2,l} \wedge \alpha \simeq \beta)]$.*

The subset of *perfectly corresponding states* is denoted as δ'. If the path based equivalence checker ascertains that $M_1 \sqsubseteq M_2$, then δ' is non-empty because it obviously contains the pair $\langle q_{1,0}, q_{2,0}\rangle$; in such a case, it may also contain other pairs as given by the following lemma.

Lemma 3.1 *If the path based equivalence checker ascertains that $M_1 \sqsubseteq M_2$, then for any loop in M_1, the relation δ' surely contains at least one state pair $\langle q, q'\rangle$, where q lies within the loop.*

Proof (By Contradiction): Let the initial sets of paths obtained from the cut-points in M_1 and M_2 be P_1^i and P_2^i, respectively. Let there be a loop in M_1 containing cut-points $q_1, q_2, \ldots, q_n$ such that $\forall i, 1 \le i \le n, q_i \notin dom(\delta')$, where $dom(\delta')$ is the domain of the relation δ'. So there exist paths of P_1^i namely, $\alpha_1 : q \twoheadrightarrow q_1, \alpha_i : q_i \twoheadrightarrow q_{i+1}, 1 \le i \le n-1$, and $\alpha_{n+1} : q_n \twoheadrightarrow q_1$ in M_1, for which there are no equivalent paths in P_2^i; also, $\forall i, 2 \le i \le n+1$, none of the extended paths $\alpha_1\alpha_2 \cdots \alpha_i$ from q has any equivalent path in M_2. Since path extension does not cross loop boundaries, the equivalence checker cannot ascertain that $M_1 \sqsubseteq M_2$ in such a case. (Contradiction) ■

We next define the set of paths whose end points are in perfect correspondence with some states in the other FSMD. Recall that $P_1 = \{\alpha_1, \alpha_2, \ldots, \alpha_p\}$ is the set of paths of the FSMD M_1 which span from a cut-point to another cut-point without having any intermediate cut-point.

Definition 3.7 (Perfect path set) *Let $P_e = \{\alpha_{e_1}, \alpha_{e_2}, \ldots, \alpha_{e_m}\}$ be a set of either extended or concatenated paths of P_1 such that for all $i, 1 \le i \le m, \alpha_{e_i}^s, \alpha_{e_i}^f \in \delta'$ and α_{e_i} does not contain any intermediary cut-points (states) which are in δ'.*

Inclusion of the extended paths in P_e is obvious; their constituent paths do not have any equivalent paths in M_2. The set P_e may also contain paths whose constituents do have equivalent paths in M_2; the constituents are still concatenated together before inclusion in P_e; specifically, let $\alpha_{e_i} = \alpha_{i,1}.\alpha_{i,2}.\cdots.\alpha_{i,m_i}$, where, for all $j, 1 \leq j \leq m_i, \alpha_{i,j}$ has some equivalent paths in M_2; in such a case, the concatenation yielding α_{e_i} is needed to satisfy the requirement that $\alpha^s_{e_i}, \alpha^f_{e_i} \in \delta'$; in other words, $\alpha^f_{i,j} \notin \delta', 1 \leq j < m_i$, because there exist some other paths leading to $\alpha^f_{i,j}$ which have no equivalent paths in M_2.

Note that the path extension based equivalence checker yields a path cover which comprises some paths which are concatenation of some paths of P_1 obtained through path extension in addition to some of the original paths of P_1. From Lemma 3.1, every loop contains at least one state in $dom(\delta')$. Hence the set P_e also provides a path cover of M_1. We refer to P_e as a *perfect path cover* of M_1.

A member of the form $\langle q_{1,i}, q_{2,j}, \phi_{i,j}\rangle$ in a simulation relation indicates that the data states at $q_{1,i}$ and $q_{2,j}$ satisfy the predicate $\phi_{i,j}$. The predicate $\phi_{i,j}$ involves the variables appearing in the two FSMDs as free variables. For path based equivalence checkers, this formula is identically $true$ for the pair of reset states; for any other pair of corresponding states $\langle q_{1,i}, q_{2,j}\rangle$, it is of the form $v_{1,k_1} = v_{2,k_1} \wedge \ldots \wedge v_{1,k_n} = v_{2,k_n}$, where $v_{k_1}, \ldots, v_{k_n}$ are the common variables live at $q_{1,i}$ and $q_{2,j}$, which is precisely the criterion that must be satisfied for the two paths to be declared equivalent by a path based equivalence checker. Note that the variables that appear in both the FSMDs M_1 and M_2 are termed as *common variables*; other variables (that appear in either of the FSMDs but not both) are termed as *uncommon variables*; a variable v is said to be live at a state q if there is an execution sequence starting at q along which its value is used before v is redefined [19]. It is also to be noted that presence of *live* uncommon variables always leads to path extension; hence, at the corresponding state pairs, there is no live uncommon variable; consequently, the uncommon variables do not appear in the $\phi_{i,j}$'s.

For establishing the fact that the path extension based equivalence checking method leads to a simulation relation, we use the following notations. Let the symbol $\overline{v_c}$ represent the name vector comprising the common variables in $V_1 \cap V_2$; the vector $\overline{v_c}$ assumes values from the domain $(\Sigma_1 \cup \Sigma_2)|_{\overline{v_c}}$ (read as "the restriction of $\Sigma_1 \cup \Sigma_2$ for the vector $\overline{v_c}$"). Let $\overline{v_{1c}}$ and $\overline{v_{2c}}$ represent the name vectors over the common variables after renaming the components of $\overline{v_c}$ by respectively adding suffix 1 for FSMD M_1 and suffix 2 for FSMD M_2. Thus, $\overline{v_{1c}}$ ($\overline{v_{2c}}$) assumes values from the domain $\Sigma_1|_{\overline{v_c}}$ ($\Sigma_2|_{\overline{v_c}}$). We additionally use the symbol $L_{i,j}(\overline{v_i}), i \in \{1, 2\}$, to denote a vector containing only those variables from $\overline{v_i}$ that are live at state $q_{i,j}$; the operation $\{\overline{e}/\overline{v}\}$ is called a substitution; the expression $\mathcal{T}\{\overline{e}/\overline{v}\}$ represents that all the occurrences of each variable $v_j \in \overline{v}$ in $\mathcal{T}$ is replaced by the corresponding expression $e_j \in \overline{e}$ simultaneously with other variables. Let $\sigma_{1,i} \in q_{1,i}(\overline{v_1})$ and $\sigma_{2,j} \in q_{2,j}(\overline{v_2})$; then $L_{1,i}(\sigma_{1,i})$ represents the values assumed by the live variables corresponding to the data state $\sigma_{1,i}$ at the control state $q_{1,i}$ in M_1; $L_{2,j}(\sigma_{2,j})$ is defined similarly.

For instance, for the Example 3.1, there is only one perfectly corresponding state pair $\langle q_{1,1}, q_{2,1}\rangle$ over and above the reset state pair $\langle q_{1,0}, q_{2,0}\rangle$; in step 2 of Algorithm 1, all

Algorithm 1 deriveSimRel (FSMD M_1, FSMD M_2, Set δ')

```
Inputs: Two FSMDs $M_1$ and $M_2$, and the set $\delta'$ of perfectly corresponding state pairs obtained from $M_1$ and $M_2$ by the path extension based equivalence checker.
Outputs: A verification relation $\mathcal{V} = \{\langle q_{1,k}, q_{2,l}, \phi_{k,l}\rangle | \langle q_{1,k}, q_{2,l}\rangle \in \delta'\}$.
1: Let the relation $\mathcal{V}$ be empty.
2: Perform live variable analyses on $M_1$ and $M_2$ and compute the set of live variables for each of the states that appears as a member of the state pairs in $\delta'$.
3: Rename each $v_j \in V_i$ as $v_{i,j}, i \in \{1,2\}$.
4: For the pair $\langle q_{1,0}, q_{2,0}\rangle$ in $\delta'$, let $\phi_{0,0}$ be $true$ and $\mathcal{V} \leftarrow \mathcal{V} \cup \{\langle q_{1,0}, q_{2,0}, \phi_{0,0}\rangle\}$.
5: For each of the other state pairs $\langle q_{1,i}, q_{2,j}\rangle$ in $\delta'$, let $\phi_{i,j}$ be $v_{1,k_1} = v_{2,k_1} \wedge \ldots \wedge v_{1,k_n} = v_{2,k_n}$, where $v_{k_1}, \ldots, v_{k_n}$ are the common variables live at $q_{1,i}$ and $q_{2,j}$; $\mathcal{V} \leftarrow \mathcal{V} \cup \{\langle q_{1,i}, q_{2,j}, \phi_{i,j}\rangle\}$.
6: return $\mathcal{V}$.
```

the program variables $y1, y2$ and res are identified to be live at these states; after renaming in step 3, they become $y1_1$, $y2_1$ and res_1 in M_1 and $y1_2$, $y2_2$ and res_2 in M_2. In step 4, $\phi_{0,0}$ is ascertained as true; in step 5, $\phi_{1,1}$ for the other perfectly corresponding state pair $\langle q_{1,1}, q_{2,1}\rangle$ is ascertained as $y1_1 = y1_2 \wedge y2_1 = y2_2 \wedge res_1 = res_2$.

Theorem 3.2 *If a path based equivalence checker declares an FSMD M_1 to be contained in another FSMD M_2, then there is a simulation relation between M_1 and M_2 corresponding to the perfect state correspondence relation δ' (Definition 3.6) produced by the equivalence checker. Symbolically,*
$\forall\langle q_{1,i}, q_{2,j}\rangle \in Q_1 \times Q_2, \ \delta'(q_{1,i}, q_{2,j}) \Rightarrow \exists\phi_{i,j}\ S(q_{1,i}, q_{2,j}, \phi_{i,j})$.

Proof: Algorithm 1 shows the steps to obtain a verification relation $\mathcal{V}$ from the output of a path extension based equivalence checker. We prove that the verification relation $\mathcal{V}$ constructed in Algorithm 1 is indeed a simulation relation, i.e., the relation $\mathcal{V}$ conforms with Definition 2.4.

Let $\mathcal{V}(q_{1,m}, q_{2,n}, \phi_{m,n})$ hold for some $q_{1,m} \in Q_1, q_{2,n} \in Q_2$. Proving that $S(q_{1,m}, q_{2,n}, \phi_{m,n})$ holds consists in substituting in Definition 2.4 of simulation relation, the relational symbol $\mathcal{V}$ for S, instantiating some of the quantified entities in this definition suitably and then showing that the clauses of the definition hold. Accordingly, by instantiating k by m and l by n we set the proof goal as

1. $\mathcal{V}(q_{1,0}, q_{2,0},\ true) \ldots$ (g.1a)
2. $\forall q_{1,i} \in Q_1, \sigma_{1,i} \in q_{1,i}(\overline{v}),\ \sigma_{1,m} \in q_{1,m}(\overline{v}),\ \eta_1 \in \mathcal{N}_1, q_{2,j} \in Q_2, \sigma_{2,j} \in q_{2,j}(\overline{v})$
$[\ \langle q_{1,i}, \sigma_{1,i}\rangle \rightsquigarrow^{\eta_1} \langle q_{1,m}, \sigma_{1,m}\rangle \wedge \phi_{i,j}(\sigma_{1,i}, \sigma_{2,j}) \wedge \mathcal{V}(q_{1,i}, q_{2,j}, \phi_{i,j}) \Rightarrow$
$\exists\sigma_{2,n} \in q_{2,n}(\overline{v}), \eta_2 \in \mathcal{N}_2$
$\{\ \langle q_{2,j}, \sigma_{2,j}\rangle \rightsquigarrow^{\eta_2} \langle q_{2,n}, \sigma_{2,n}\rangle \wedge \eta_1 \equiv \eta_2 \wedge \phi_{m,n}(\sigma_{1,m}, \sigma_{2,n}) \wedge \mathcal{V}(q_{1,m}, q_{2,n}, \phi_{m,n})\ \}\ ] \ldots$ (g.1b).

Now, $\mathcal{V}(q_{1,m}, q_{2,n}, \phi_{m,n}) \Rightarrow \delta'(q_{1,m}, q_{2,n})$ from steps 4 and 5 of Algorithm 1. From Definition 3.5 of δ and Definition 3.6 of δ', either $q_{1,m} = q_{1,0}$ (i.e., $m = 0$) and $q_{2,n} = q_{2,0}$ (i.e., $n = 0$) or both $m, n \neq 0$. If $m = 0$, then n must be 0 (since the respective reset states $q_{1,0}, q_{2,0}$ form a corresponding state pair in δ and also a perfectly corresponding state pair in δ' only between themselves as per Definition 3.5 and Definition 3.6; accordingly, Algorithm'1 puts only the $\mathcal{V}(q_{1,0}, q_{2,0}, \phi_{0,0})$ in step 4 involving these states. So the goal (g.1a) is proved.

Now, let $m, n \neq 0$. In this case, goal (g.1b) is to be proved. By Definition 3.6, for any path α with $\alpha^f = q_{1,m}$, there exists a path β with $\beta^f = q_{2,n}$ such that $\alpha \simeq \beta$.

Now let us consider any choice of the universally quantified entities in the goal (g.1b) such that the antecedent of (g.1b) holds. In such a case, we show that the consequent of (g.1b) holds by induction on the length $|\eta_1|$ of the chosen execution sequence η_1 (in terms of the number of paths of the perfect path cover P_e of M_1 that occur in η_1).

Basis case ($|\eta_1| = 1$): Let $\eta_1 = \alpha_1$. We rewrite the goal (g.1b) as
$\langle q_{1,i}, \sigma_{1,i}\rangle \twoheadrightarrow^{\alpha_1} \langle q_{1,m}, \sigma_{1,m}\rangle \wedge \phi_{i,j}(\sigma_{1,i}, \sigma_{2,j}) \wedge \mathcal{V}(q_{1,i}, q_{2,j}, \phi_{i,j}) \Rightarrow$
$\exists \sigma_{2,n} \in q_{2,n}(\overline{v}), \eta_2 \in \mathcal{N}_2$
$\{ \langle q_{2,j}, \sigma_{2,j}\rangle \rightsquigarrow^{\eta_2} \langle q_{2,n}, \sigma_{2,n}\rangle \wedge \alpha_1 \equiv \eta_2 \wedge \phi_{m,n}(\sigma_{1,m}, \sigma_{2,n}) \wedge \mathcal{V}(q_{1,m}, q_{2,n}, \phi_{m,n}) \} \ldots$ (g.2).

Since the antecedents are true, the second conjunct $\phi_{i,j}(\sigma_{1,i}, \sigma_{2,j}) : \sigma_{1,i}|_{\overline{v_{1c}}} = \sigma_{2,j}|_{\overline{v_{2c}}}$ holds; from the third conjunct $\mathcal{V}(q_{1,i}, q_{2,j}, \phi_{i,j})$, $\delta'(q_{1,i}, q_{2,j})$ holds by step 5 of Algorithm 1.
In the consequent, the fourth conjunct $\mathcal{V}(q_{1,m}, q_{2,n}, \phi_{m,n})$ holds by the starting premise which, in turn, implies $\delta'(q_{1,m}, q_{2,n})$ by step 5 of Algorithm 1; by Definition 3.6, $\exists \beta_1 : q_{2,j} \twoheadrightarrow q_{2,n}$ s.t. $\alpha_1 \simeq \beta_1$. So we choose the existentially quantified entities $\eta_2 = \beta_1$ and $\sigma_{2,n} = s_{\beta_1}(\sigma_{2,j})$. The first and the second conjuncts in the consequent obviously hold by this choice; note that $\alpha_1 \simeq \beta_1 \Rightarrow \alpha_1 \equiv \beta_1$ by soundness of the equivalence checker. From the second conjunct $\phi_{i,j}(\sigma_{1,i}, \sigma_{2,j})$ of the antecedent, we have: $\sigma_{1,i}|_{\overline{v_{1c}}} = \sigma_{2,j}|_{\overline{v_{2c}}}$; since $\alpha_1 \simeq \beta_1$, $s_{\alpha_1}(\sigma_{1,i}|_{\overline{v_{1c}}}) = s_{\beta_1}(\sigma_{2,j}|_{\overline{v_{2c}}})$ which implies $\sigma_{1,m}|_{\overline{v_{1c}}} = \sigma_{2,n}|_{\overline{v_{2c}}} \Rightarrow \phi_{m,n}(\sigma_{1,m}, \sigma_{2,n})$; so the third conjunct holds. Thus, all the conjuncts of the consequents hold. This accomplishes the proof of the basis case.

Induction Hypothesis ($|\eta_1| = l-1$): Let the goal (g.1b) hold whenever $|\eta_1| = l-1$.

Induction step: Let $\eta_1 = \eta_1'\alpha$, where $|\eta_1| = l$; so the goal becomes
$\langle q_{1,i}, \sigma_{1,i}\rangle \rightsquigarrow^{\eta_1'.\alpha} \langle q_{1,m}, \sigma_{1,m}\rangle \wedge \phi_{i,j}(\sigma_{1,i}, \sigma_{2,j}) \wedge \mathcal{V}(q_{1,i}, q_{2,j}, \phi_{i,j}) \Rightarrow$
$\exists \sigma_{2,n} \in q_{2,n}(\overline{v}), \eta_2 \in \mathcal{N}_2$
$\{ \langle q_{2,j}, \sigma_{2,j}\rangle \rightsquigarrow^{\eta_2} \langle q_{2,n}, \sigma_{2,n}\rangle \wedge \eta_1'\alpha \equiv \eta_2 \wedge \phi_{m,n}(\sigma_{1,m}, \sigma_{2,n}) \wedge \mathcal{V}(q_{1,m}, q_{2,n}, \phi_{m,n}) \} \ldots$ (g.3).

By induction hypothesis applied non-vacuously on $\eta_1' : \langle q_{1,i}, \sigma_{1,i}\rangle \rightsquigarrow \langle q_{1,i'}, \sigma_{1,i'}\rangle$ with its equivalent sequence $\eta_2' : \langle q_{2,j}, \sigma_{2,j}\rangle \rightsquigarrow \langle q_{2,i'}, \sigma_{1,i'}\rangle$, we have the following premise (that holds non-vacuously)
$\langle q_{1,i}, \sigma_{1,i}\rangle \rightsquigarrow^{\eta_1'} \langle q_{1,i'}, \sigma_{1,i'}\rangle \wedge \phi_{i,j}(\sigma_{1,i}, \sigma_{2,j}) \wedge \mathcal{V}(q_{1,i}, q_{2,j}, \phi_{i,j}) \Rightarrow$
$\{ \langle q_{2,j}, \sigma_{2,j}\rangle \rightsquigarrow^{\eta_2'} \langle q_{2,j'}, \sigma_{2,j'}\rangle \wedge \eta_1' \equiv \eta_2' \wedge \phi_{i',j'}(\sigma_{1,i'}, \sigma_{2,j'}) \wedge \mathcal{V}(q_{1,i'}, q_{2,j'}, \phi_{i',j'}) \} \ldots$ (i)
and the following goal results from (g.3) over the path α:
$\langle q_{1,i'}, \sigma_{1,i'}\rangle \twoheadrightarrow^{\alpha} \langle q_{1,m}, \sigma_{1,m}\rangle \wedge \phi_{i,j}(\sigma_{1,i'}, \sigma_{2,j'}) \wedge \mathcal{V}(q_{1,i'}, q_{2,j'}, \phi_{i',j'}) \Rightarrow$
$\exists \sigma_{2,n} \in q_{2,n}(\overline{v}), \eta_2 \in \mathcal{N}_2$
$\{ \langle q_{2,j'}, \sigma_{2,j'}\rangle \rightsquigarrow^{\eta_2} \langle q_{2,n}, \sigma_{2,n}\rangle \wedge \alpha \equiv \eta_2 \wedge \phi_{m,n}(\sigma_{1,m}, \sigma_{2,n}) \wedge \mathcal{V}(q_{1,m}, q_{2,n}, \phi_{m,n}) \} \ldots$ (g.4).
The proof of (g.4) is exactly identical to that of the basis case.

Since the premise (i) holds non-vacuously, the fourth conjunct $\mathcal{V}(q_{1,i'}, q_{2,j'}, \phi_{i',j'})$ of the consequent (which is also the third conjunct in the antecedent of the goal (g.4)) holds implying that $\delta'(q_{1,i'}, q_{2,j'})$ holds by step 5 of Algorithm 1. Again, in the consequent of the goal (g.4), the fourth conjunct $\mathcal{V}(q_{1,m}, q_{2,n}, \phi_{m,n})$ holds by the premise which, in turn, implies that $\delta'(q_{1,m}, q_{2,n})$ holds by step 5 of Algorithm 1. By Definition 3.6, $\exists \beta : \langle q_{2,j'} \twoheadrightarrow q_{2.n}\rangle$ s.t. $\alpha \simeq \beta$. So in the consequent of (g.4), choosing the existentially quantified η_2 as β and $\sigma_{2,n} = s_\beta(\sigma_{2,j'})$, the first and the second conjunct of (g.4) hold immediately. From the second conjunct $\phi_{i',j'}(\sigma_{1,i'}, \sigma_{2,j'})$ of the antecedent of (g.4), we have $\sigma_{1,i'}|_{\overline{v_{1c}}} = \sigma_{2,j'}|_{\overline{v_{2c}}}$; since $\alpha_1 \simeq \beta_1$, $s_{\alpha_1}(\sigma_{1,i'}|_{\overline{v_{1c}}}) = s_{\beta_1}(\sigma_{2,j'}|_{\overline{v_{2c}}})$, i.e., $\sigma_{1,m}|_{\overline{v_{1c}}} = \sigma_{2,n}|_{\overline{v_{2c}}}$ which is $\phi_{m,n}(\sigma_{1,m}, \sigma_{2,n})$; so the third conjunct holds. Thus, all the four conjuncts of the consequent of (g.4) hold. This accomplishes the proof of the induction step. ■

Note that the path extension based equivalence checker ensures $M_1 \sqsubseteq M_2$ first and then $M_2 \sqsubseteq M_1$. In the second step, it does not have to change the set δ' of perfectly corresponding state pairs constructed during the first step. Hence, the simulation relation $\mathcal{V}$ obtained from Algorithm 1 will be a bisimulation relation too by Definition 2.5.

It is important to note that as per Definition 2.4 of simulation relation, there is no restriction on the end states of execution sequences. Algorithm 1, however, produces a bisimulation relation comprising triples for only the perfectly corresponding states. This does not impair generality because if one is interested in enhancing the bisimulation relation by incorporating triples for some control states other than the perfectly corresponding states, one may easily do so by applying Dijkstra's weakest precondition computation starting from the perfectly corresponding states appearing immediately next to one's states of choice in the control flow graph of FSMD. For instance, for Example 3.1, Algorithm 1 yields the data state predicates $\phi_{0,0}$ and $\phi_{1,1}$ only at the two perfectly corresponding state pairs $\langle q_{1,0}, q_{2,0}\rangle$ and $\langle q_{1,1}, q_{2,1}\rangle$, respectively; now the predicate corresponding to some intermediary state pair, $\langle q_{1,4}, q_{2,1}\rangle$, say, can be computed as $\phi_{4,1} = wp(\phi_{1,1}, q_{1,4} \twoheadrightarrow q_{1,1}) = (y1_1 = y1_2 \wedge (y2_1/2) = y2_2 \wedge res_1 = res_2)$, where wp represents Dijkstra's weakest precondition computation. An analogous situation arises in Kundu et al.'s method described in [4], [5], where the bisimulation relation produced comprises triples only for the *pairs of interest* which are basically pairs of control states, one from the specification and the other from the implementation (similar to our corresponding states).

4 Conclusion

Both bisimulation relation based methods and path extension based equivalence checking approaches are prevalent in the literature on translation validation of programs. The basic methodologies of these two approaches differ; the (conventional) bisimulation relation based approach tries to construct a relation that serves as a witness of the two programs being symbolically executed in an equivalent manner, whereas, the path extension based approach tries to obtain path covers in the two FSMDs such that each path in one is found to be equivalent with a path in the other and vice-versa. Both these methods have their own merits and demerits. The bisimulation relation based approach can be used to validate complex transformations such as loop shifting which are not handled by the present path based approaches. However, all methods that have adopted this approach fail to guarantee termination. These methods are, currently, highly susceptible to modifications of the control structure. On the other hand, the path extension based approach has been shown to be better equipped to handle control structure modifying transformations and this verification scheme is guaranteed to terminate. In this work, we relate these two (apparently different) approaches by explaining how bisimulation relations can be derived from the outputs of path extension based equivalence checkers. Moreover, as a result of the current work, it is now possible to construct bisimulation relations between behaviours which were not possible using earlier techniques. Developing a unified framework that encompasses all the benefits of these two approaches seems to be an interesting future work.

Acknowledgments

The authors would like to thank the anonymous reviewers for their comments and suggestions which helped us improve this article. The work of K. Banerjee was supported by Tata Consultancy Services (TCS) Research Fellowship. This work was also funded by Department of Science and Technology (DST), India, Project No: SB/EMEQ-281/2013.

References

[1] A. Pnueli, M. Siegel, and E. Singerman, "Translation validation," in *TACAS*, 1998, pp. 151–166.
[2] M. Rinard, "Credible compilation," In Proceedings of CC 2001: International Conference on Compiler Construction, Tech. Rep., 1999.
[3] G. C. Necula, "Translation validation for an optimizing compiler," in *PLDI*, 2000, pp. 83–94.
[4] S. Kundu, S. Lerner, and R. Gupta, "Validating high-level synthesis," in *CAV*, 2008, pp. 459–472.
[5] ——, "Translation validation of high-level synthesis," *IEEE Trans on CAD of ICS*, vol. 29, no. 4, pp. 566–579, 2010.
[6] R. Camposano, "Path-based scheduling for synthesis," *IEEE Trans on CAD of ICS*, vol. 10, no. 1, pp. 85–93, Jan. 1991.
[7] M. Rahmouni and A. A. Jerraya, "Formulation and evaluation of scheduling techniques for control flow graphs," in *EURO-DAC*, 1995, pp. 386–391.
[8] T. Li, Y. Guo, W. Liu, and M. Tang, "Translation validation of scheduling in high level synthesis," in *GLSVLSI*, 2013, pp. 101–106.
[9] D. D. Gajski, N. D. Dutt, A. C. Wu, and S. Y. Lin, *High-Level Synthesis: Introduction to Chip and System Design*. Kluwer Academic, 1992.
[10] C. Karfa, C. Mandal, D. Sarkar, S. R. Pentakota, and C. Reade, "A formal verification method of scheduling in high-level synthesis," in *ISQED*, 2006, pp. 71–78.
[11] C. Karfa, D. Sarkar, C. Mandal, and P. Kumar, "An equivalence-checking method for scheduling verification in high-level synthesis," *IEEE Trans on CAD of ICS*, vol. 27, pp. 556–569, 2008.
[12] C.-H. Lee, C.-H. Shih, J.-D. Huang, and J.-Y. Jou, "Equivalence checking of scheduling with speculative code transformations in high-level synthesis," in *ASP-DAC*, 2011, pp. 497–502.
[13] C. Karfa, C. Mandal, and D. Sarkar, "Formal verification of code motion techniques using data-flow-driven equivalence checking," *ACM Trans. Design Autom. Electr. Syst.*, vol. 17, no. 3, p. 30, 2012.
[14] R. Lezuo, "Scalable translation validation," Ph.D. dissertation, Vienna University of Technology, 2014.
[15] R. Lezuo, G. Barany, and A. Krall, "CASM: implementing an abstract state machine based programming language," in *Software Engineering - Workshopband*, 2013, pp. 75–90.
[16] C. Karfa, D. Sarkar, and C. Mandal, "Verification of datapath and controller generation phase in high-level synthesis of digital circuits," *IEEE Trans. on CAD of ICS*, vol. 29, no. 3, pp. 479–492, 2010.
[17] A. Darte and G. Huard, "Loop shifting for loop compaction," *J. Parallel Programming*, vol. 28, no. 5, pp. 499–534, 2000.
[18] R. W. Floyd, "Assigning meaning to programs," in *Proceedings the 19^{th} Symposium on Applied Mathematics*, 1967, pp. 19–32.
[19] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Princiles, Techniques, and Tools*. Pearson Education, 2006.

Kunal Banerjee is currently a Research Scientist at Intel's Parallel Computing Lab in Bangalore, India. His research interests are in the areas of program analysis, formal methods and parallel programming. He received his B Tech degree in Computer Science and Engineering from Heritage Institute of Technology. He received his PhD from the Department of Computer Science and Engineering, IIT Kharagpur. He has been a recipient of Senior Research Fellowship from the Department of Science and Technology, India, and TCS Research Fellowship for supporting his doctoral studies.

Dipankar Sarkar did his B Tech and M Tech in Electronics and Electrical Communication Engg. and PhD in Engineering from IIT Kharagpur. He is a faculty member at IIT Kharagpur for last thirty five years. His area of interest is formal verification of circuits and systems.

Chittaranjan Mandal is a senior member of the IEEE. He received his PhD from IIT Kharagpur in 1997. After serving as a reader at Jadavpur University, Calcutta he joined IIT Kharagpur in 1999 where he is currently a professor in the Department of Computer Science and Engineering. His research interests include formal modeling and verification, high-level system design and also network and web technologies. He has been an Industrial Fellow of Kingston University, UK, since 2000 and was also a recipient of a Royal Society Fellowship for conducting collaborative research in the UK. He has active participation in educational accreditation activities. He has handled sponsored projects from Indian government agencies such as DIT, DST, MHRD, Indian Railways and also from private agencies such as Nokia, Natsem and Intel. He serves as a reviewer for several international journals and conferences. He has supervised several PhD students and has over a hundred and twenty five publications some of which have attracted best paper awards and significant citations.