Translation Validation of Scheduled Conditional Behavior using Petri Net based Program Models

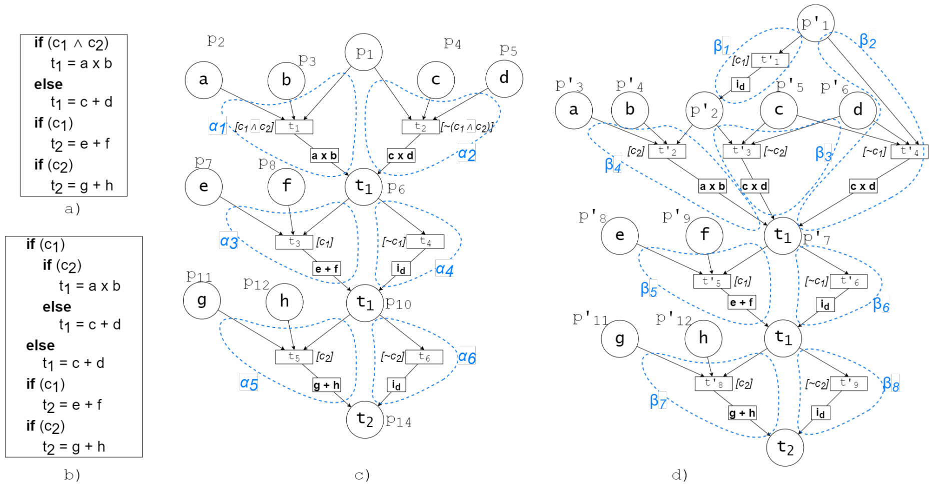
Rakshit Mittal[1,2] and Soumyadip Bandyopadhyay[1,3]

[1] BITS Pilani, KK Birla Goa Campus, Goa, 403726 India
[2] Telecom Paris, Institut Polytechnique de Paris, 91120 France
[3] Hasso-Platner Institute, Potsdam, 14482 Germany

High-Level Synthesis (HLS) tools and schedulers are large and complex systems. More often than not, they are developed without formal proofs. In spite of rigorous testing, bugs do occur in these tools. These bugs may result in generation a target code that is not semantically equivalent to the original code. Ensuring the correctness of the translations performed by these HLS tools like compilers and schedulers is necessary for their reliability. Translation validation is the process of verifying that the target code generated by these translations of the HLS tools is semantically equivalent to the original code. Our paper targets the translation validation of the scheduling phase of these HLS tools.

Path-based equivalence checking (PBEC) approaches have made significant progress in translation validation. Such approaches have been developed for various modelling paradigms like Control and Data Flow Graphs (CDFG), Petri net, etc. The CDFG based approach reported in [1] can handle the transformation using costly value propagation based method. Being value based and with the inherent ability of emulating instruction level parallelism, we enhance the validation method using Petri net-based model[2] such that costly value propagation is not required for equivalence checking. Our approach aims to establish equivalence between the two scheduled behaviors by proving the equivalence between the paths present in the Petri nets of these two behaviors.

Motivating Example: Through a motivating example we describe the equivalence checking between two codes depicted in Figs. 1 (a) and (b). The corresponding models are depicted in Figs. 1 (c) and (d). In a general program with loop/s, we do not know how many times the loop/s will be executed. To analyze translations involving loops, it is necessary to express the CPN model's computations (with a possibly infinite number of loop traversals) into a finite number of paths. This is done by identifying cut-points. The in-ports (place with no pre-transition), the out-ports (place with no post-transition), the places with back edge and, the places (bifurcation points) with more than one post-places are all cut-points. A path-constructor module identifies these cut-points and gives us the set of paths. In Figure 1 (c), the set of cut-points is $\{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_{10}, p_{11}, p_{12}, p_{14}\}$. The set of paths, $\Pi_0 = \{\alpha_1 = \langle\{t_1\}\rangle$, $\alpha_2 = \langle\{t_2\}\rangle$, $\alpha_3 = \langle\{t_3\}\rangle$, $\alpha_4 = \langle\{t_4\}\rangle$, $\alpha_5 = \langle\{t_5\}\rangle$, $\alpha_6 = \langle\{t_6\}\rangle$. Note that a path is a sequence of maximally parallelizable transitions from a set of



```
if (c1 ∧ c2)
    t1 = a x b
else
    t1 = c + d
if (c1)
    t2 = e + f
if (c2)
    t2 = g + h
```

a)

```
if (c1)
    if (c2)
        t1 = a x b
    else
        t1 = c + d
else
    t1 = c + d
if (c1)
    t2 = e + f
if (c2)
    t2 = g + h
```

b)

Fig. 1. Illustrative example

cut-points to a cut-point without intermediary cut-points. Similarly in Fig. 1(d), cut-points are $\{p'_1, p'_2, p'_3, p'_4, p'_5, p'_6, p'_7, p'_8, p'_9, p'_{11}, p'_{12}, p'_{13}, p'_{15}\}$; the path cover, $\Pi_1 = \{\beta_1 = \langle\{t'_1\}\rangle,\ \beta_2 = \langle\{t'_4\}\rangle,\ \beta_3 = \langle\{t'_3\}\rangle,\ \beta_4 = \langle\{t'_2\}\rangle,\ \beta_5 = \langle\{t'_5\}, \{t'_7\}\rangle,$ $\beta_6 = \langle\{t'_6\}\rangle,\ \beta_7 = \langle\{t'_7\}\rangle,\ \beta_8 = \langle\{t'_8\}\rangle\}$.

The notion of equivalence checking is as follows: "$\forall$ paths $\in \Pi_0$ $\exists$ an equivalent path in Π_1 and vice-versa". A path-based equivalence checking module takes these sets of paths for both the Petri net models, and using the concept of path merging as well as extension of paths that we have developed, checks for equivalence between the paths and returns a Yes/No answer for the semantic equivalence between source and translated programs. The module does not give a false positive result, but since the approach is not complete, a 'No' answer is interpreted as 'Can't Say'. Equivalence between two expressions is checked using the Z3 SMT solver [3]. Using this approach the set of equivalent pairs of paths is $\{\langle\alpha_1, (\beta_1 \cdot \beta_4)\rangle, \langle\alpha_2, \beta_2||\beta_3\rangle, \langle\alpha_3, \beta_5\rangle, \langle\alpha_4, \beta_6\rangle, \langle\alpha_5, \beta_7\rangle, \langle\alpha_6, \beta_8\rangle\}$

Capabilities and limitations: Our approach can handle several uniform and non-uniform code optimizing and parallelizing transformations for scalar programs. The major limitation of our method is that it cannot validate array handling programs. The method also incapable to validate loop shifting class of transformations.

References

1. Chouksey, R., Karfa, C.: Verification of scheduling of conditional behaviors in high-level synthesis. IEEE Trans. on Very Large Scale Integration (VLSI) Systems (2020)
2. Mittal, R., Banerjee, R., Sarkar, S., Bandyopadhyay, S.: Translation validation of loop involving code optimizing transformations using petri net based models of programs. In: PNSE 2020. vol. 2651, pp. 138–146. CEUR-WS.org (2020)
3. Moura, L.D., Bjørner, N.: Z3: an efficient smt solver. In: TACAS'08/ETAPS'08 Proceedings of the Theory and Practice of Software. pp. 337–340 (2008)