Property-Specific Witness Graph Generation for Guided Simulation

A. Casavant, A. Gupta, S. Liu, A. Mukaiyama, K. Wakabayashi, and P. Ashar
NEC Corp. {ashar@ccrl.nj.nec.com}

A practical solution to the complexity of design validation is semi-formal verification, where the specification of correctness criteria is done formally, as in model checking, but checking is done using simulation, which is *guided* by directed vector sequences derived from knowledge of the design and/or the property being checked. Simulation vectors must be effective in targeting the types of bugs designers expect to find rather than some generic coverage metrics. The focus of our work is to generate property-specific testbenches for guided simulation, that are targeted either at proving the correctness of a *full* CTL property or at finding a bug. This is facilitated by generation of a property-specific model, called a *"Witness Graph"*, which captures interesting paths in the design. Starting from an initial abstract model of the design, symbolic model checking, pruning, and refinement steps are applied in an iterative manner, until either a conclusive result is obtained or computing resources are exhausted. The witness graph is annotated with, e.g., state or transition priorities before testbench generation. The overall testbench generation flow, and the iterative flow for witness graph generation are shown in Figures 1 and 2.

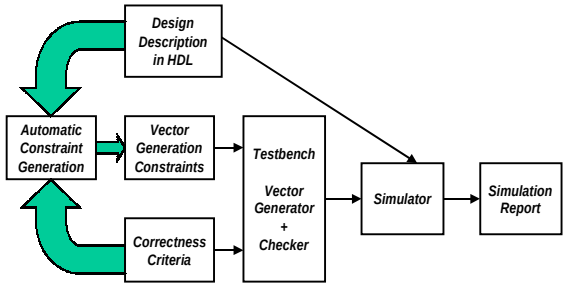

Figure 1: Smart Testbench Generation

To generate the abstract model, *m*, we first use cone-of-influence abstraction, whereby any part of the design, *d*, that does not affect the property is removed. Since the number of control states is small, explicit traversal is used to identify irrelevant datapath operations. Next, we identify datapath variables that do not directly appear as atomic propositions in the CTL property, and are therefore suitable for abstraction as pseudo-primary inputs. Again, we use explicit traversal over the control states to identify datapath dependencies for ranking these candidates and abstracting them. The resulting model constitutes an upper bound approximation. The next step is to perform deterministic analysis using model checking on the abstract model in order to identify states that contribute to any witness or counter-example (*CE*) for the property of interest. The input CTL formula *f* is in negation normal form, i.e. where all negations appear only at the atomic level. For E-type sub-formulas, we look for all witnesses; while for A-type sub-formulas, we look for all *CEs*. This state set is used for guidance during simulation over *d*, in order to demonstrate a concrete witness/*CE*. In particular, we target *over-approximate sets* (*upper*) of satisfying states during model checking, so that we can search through an over-approximate set of witnesses/*CEs* during simulation. For sub-formulas with an E- operator (EX, EF, EU, EG), standard model checking over *m* ensures an over-approximation over *d*. For sub-formulas with an A- operator (AX, AF, AU, AG), we compute *upper* by considering the corresponding E- (e.g. AG becomes EG). Since the over-approximation for the A- operators is coarse, we also compute a set of abstract states called *negative* corresponding to the intersection of set *upper* with a set which is recursively computed for the *negation* of the A- sub-formula. We show that the set computation has the same complexity as standard symbolic model checking. In some cases, a conclusive proof may be obtained during this *upper/negative* identification.

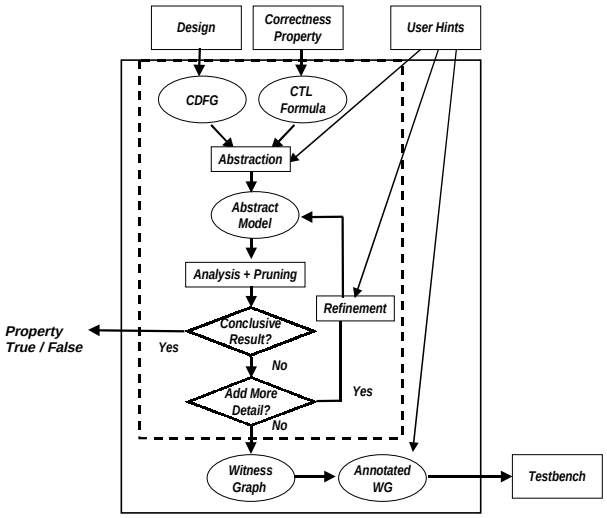

Figure 2: Flow for Witness Graph Generation

When the result is inconclusive, we fall back upon simulation. An abstract state *s* which belongs to *upper*, but not to *negative*, is a very desirable state to target as a witness for the A- sub-formula because the proof of the A- sub-formula is complete for state *s* due to model checking itself, i.e., as soon as state *s* is reached during simulation, there is no further proof obligation. If a state *t* belongs to *negative* also, our task during simulation is to check whether there is a concrete path starting from *t* that shows the *CE* for the A- sub-formula. If such a *CE* is found, state *t* is not a true witness state, and can be eliminated from further consideration. The contribution of our work is the choice of over approximation and how it is used to guide the simulation. Using the *upper/negative* sets we prune the abstract model by removing states that do not start a witness/*CE* and are not needed for demonstrating it fully. Once pruning is done, it may be possible to refine the model by bringing back some of the datapath variables abstracted out earlier and performing the analysis again. Sets derived from *upper/negative* during pruning/marking are used in the backtrack-search procedure of the testbench during simulation for proving the property or its negation. Apart from using a Witness Graph for generating a testbench, it can also be used as a coverage metric for evaluating the effectiveness of a given set of simulation vectors.