A Divide And Conquer Strategy For Consistency And Troubleshooting In Specifications Of Large Designs

Nina Saxena
Univ. of Texas, Austin

Jacob A. Abraham
Univ. of Texas, Austin

Avijit Saha
IBM, Austin

1. Introduction

Coding specifications correctly is an essential part of the verification process. Most often specifications of large designs, having been coded by different people using different sources, tend to be inconsistent and prone to errors. We address this problem and present a method for checking pair-wise consistency, using a system of incremental specification. The incremental specification approach involves breaking up specifications into a set of sub-specifications, through case-based refinement, such that the union of the sub-specifications contains the original specification. Such a system of specification would also facilitate compositional verification as in [2, 3]. Pair-wise consistency is then checked using the conjunctive combination of two sub-specifications at a time. We show that pair-wise consistency using this method is sufficient to ensure global consistency in specifications. We also present a method for fast diagnostics and rectification of erroneous specifications.

2 Incremental Specification

Most formal verification tools are built to consider only that part of a design that pertains to the specification against which the design is being verified. The rest of the logic is regarded as superfluous with reference to the specification. If ϕ is a specification and D is a design such that ϕ is involved with only D_ϕ part of the design, then the other part $D_{\overline{\phi}}$ is pruned off from the model to get a reduced model for verification. This pruning allows us to use a divide and conquer approach through specifications. Instead of verifying an entire specification that would normally result in space explosion, we partition the behavior into multiple cases $\phi_1 ... \phi_r$ such that the union of the sub-specifications $\phi_1 \cup ... \cup \phi_r$ contains the original specification ϕ.

3 Consistency Check

Typically, the behavior of a module M is described by a number of formulae $\phi_1 ... \phi_n$ which are formulated using variables $v_1 ... v_m$. For convenience, we shall use $\perp$ to denote failure and $\top$ to denote a successful verification run. The design description or model shall be called ϕ_D. We propose an algorithm Incremental_Check, depicted in Figure 1, that checks pair-wise consistency through incremental specification and verification.

Definition 1: Let specifications ϕ_i and ϕ_j be in agreement with the design, that is, $\phi_i \wedge \phi_D = \top$ and $\phi_j \wedge \phi_D = \top$. Then, ϕ_i and ϕ_j are said to be inconsistent iff for at least one state s and input i, ϕ_i and ϕ_j specify dissimilar transitions T_i and T_j, from s to s_i and s to s_j, respectively.

Lemma 1: If specifications ϕ_i and ϕ_j are inconsistent, then there is at least one common variable v such that both ϕ_i and ϕ_j dictate the behavior of v.

Proof: This follows directly from Definition 1.

Lemma 2: Let $\phi_k = \phi_i \wedge \phi_j$. Then $\phi_k \wedge \phi_D \neq \perp$ implies that $\phi_i \wedge \phi_j \neq \perp$.

Proof: Let specification ϕ_k agree with the design description ϕ_D. Also, let ϕ_i and ϕ_j be inconsistent specifications. Then, by Lemma 1, there is at least one variable v that is assigned dissimilar values by ϕ_i and ϕ_j at the same instance. However this would cause the specification ϕ_k to be violated, as v cannot have 2 dissimilar assignments at the same time, violating the assumption $\phi_k \wedge \phi_D = \top$. Thus, by contradiction, it is proved that
$\phi_k \wedge \phi_D \neq \perp => \phi_i \wedge \phi_j \neq \perp$.

Definition 2: Specifications ϕ_i and ϕ_j are α-*consistent* if ϕ_i passes on description ϕ_D, ϕ_j passes on description ϕ_D and $\phi_i \wedge \phi_j \neq \perp$.

Definition 3: A set, $\phi = \{\phi_1, \phi_2, ..., \phi_n\}$, of specifications is said to be globally α-consistent if $\phi_1 \wedge \phi_2 \wedge ... \wedge \phi_n \wedge \phi_D \neq \perp$.

Theorem 1: Algorithm Incremental_Check ensures global α-consistency.

Proof: We first consider the base case. If $\phi_1 \wedge \phi_D \neq \perp$ then ϕ_1 is α-consistent by definition. If $\phi_2 \wedge \phi_D \neq \perp$

```
Begin
    ϕ = {}
    for each i do
    begin
        flag = true;
        compute ϕi ∧ ϕD;        /* verifying ϕi */
        if ϕi ∧ ϕD ≠ ⊥ do       /* ϕi passes */
        begin
            for j = 1 to i-1 do
            begin
                ϕk = ϕi ∧ ϕj
                compute ϕk ∧ ϕD;        /* verifying ϕk */
                if ϕk ∧ ϕD = ⊥ do       /* ϕk fails */
                flag = false;
            end
            if flag = true do ϕ = ϕ ∪ ϕi
            else Rectify;
        end
        else Rectify;
    end
End
```

Figure 1: Algorithm: Incremental_Check

and $\phi_2 \wedge \phi_1 \neq \perp$ then ϕ_2 is α-consistent. Now if set $\phi = \{\phi_1, ..., \phi_n\}$ is α-consistent, then by Lemma 2, $\phi_1 \wedge \phi_2 \wedge ... \wedge \phi_n \wedge \phi_D \neq \perp$. Then, let ϕ_{n+1} be a new formula such that $\phi_{n+1} \wedge \phi_D \neq \perp$ and $\phi_{n+1} \wedge \phi_j \neq \perp$ where j = 1 to n. Again, by Lemma 2, $\phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4 \wedge ... \wedge \phi_n \wedge \phi_{n+1} \wedge \phi_D \neq \perp$. Hence, by induction algorithm Incremental_Check ensures global α-consistency.

4 Causality Based Troubleshooting

Troubleshooting within inconsistent specifications is as important as checking consistency. Usually, whenever changes are made in the design or the specifications themselves are rectified, some other specifications are violated due to these changes. Often the number of failed specifications makes it difficult to find the exact cause of failure.

```
Begin
    S = set of failed specifications;
    S_causal = {};
    for each specification ϕm in S do
    begin
        if ( ϕm depends on ϕn ) do S_causal = S_causal ∪ ϕn ;
    end
    S = S_causal;
    S_causal = {};
End     /* Resultant S is the set of specs. to be edited */
```

Figure 2: Algorithm: Rectify

By considering causality we can make diagnostics easier. We first consider the transitive closure of all inconsistent specifications. Within that transitive closure S, we now pick only those specifications on which other specifications in the set S are causally dependent. Let us call this set S_{causal}. This now becomes our new set S and we look for causal links within this reduced set. We continue this process until no more causal specifications are found within the set S. Specifications in this resultant S then become our primary candidates for editing and rectification. Figure 2 gives the algorithm Rectify for causality based troubleshooting. Figure 3 pictorially depicts the diagnostic process.

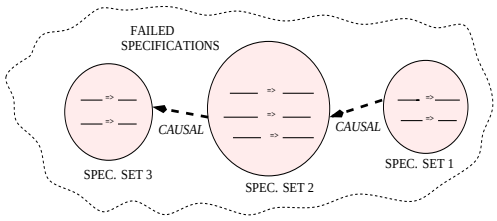

Figure 3: Causality based diagnostics

5 Conclusion

A method for incremental specification and consistency checking was presented. It was also shown that by exploiting causalities in specifications, we can quickly rectify the faulting specifications. This method is especially useful for verifying changes while the design is still in the development phase. In that case, each changed specification is treated as a new specification and the incremental verification algorithm is then used to ascertain its consistency. The presented methods were tested using a hybrid system that uses common specifications for model checking and simulation [4, 1].

Acknowledgement

This work was supported in part by IBM under Project No. 08503.

References

[1] W. Canfield, E. A. Emerson, and A. Saha. Checking formal specifications under simulation. *International Conference on Computer Design*, October 1997.

[2] T. A. Henzinger. *The Temporal Specification and Verification of Real-Time Systems*. Ph.D. Thesis, Dept. of Computer Science, Stanford Univ., 1991.

[3] D. Long. *Model Checking, Abstraction, and Compositional Verification.* Ph.D. Thesis, CMU, July 1993.

[4] N. Saxena, J. Baumgartner, A. Saha, and J. A. Abraham. To model check or not to model check. *International Conference on Computer Design*, 1998.