Circular Compositional Reasoning about Liveness

K. L. McMillan

Cadence Berkeley Labs

Abstract. Compositional proofs about systems of many components often involve apparently circular arguments. That is, correctness of component A must be assumed when verifying component B, and *vice versa*. The apparent circularity of such arguments can be resolved by induction over time. However, previous methods for such circular compositional proofs apply only to safety properties. This paper presents a method of circular compositional reasoning that applies to liveness properties as well. It is based on a new circular compositional rule implemented in the SMV proof assistant. The method is illustrated using Tomasulo's algorithm for out-of-order instruction execution. An implementation is proved live for arbitrary resources using compositional model checking.

1 Introduction

Compositional model checking methods [7,4] reduce the verification of large systems to smaller, localized verification problems. This is necessary because model checking is limited by the "state explosion problem". When reasoning compositionally about two processes A and B, it is often necessary to assume correctness of A to verify B and *vice versa*. The apparent circularity of such arguments can be resolved by induction over time. However, existing methods for such circular compositional proofs [2,3,7] apply only to safety properties. Nonetheless, mutual dependence of liveness properties does occur in real systems. Consider, for example, the problem of multiple execution units in an instruction set processor. At some times, the instruction in unit A may depend on the result of the instruction in unit B, and at other times the inverse relation may hold. Thus, in order to prove that unit A is live (always eventually produces a result), we must assume the B is live, and *vice versa*. Here, we introduce a compositional technique that allows this kind of circular compositional reasoning. In essence, it makes explicit the induction over time implied in the above approach, by assuming property P *only* up to time $t-1$ when proving Q at time t, and *vice versa*. This condition (Q up to $t-1$ implies P up to t) is expressible in temporal logic. Thus, the proof obligations incurred using this method can be discharged by model checking. This proof method has been integrated with others in a proof assistant based on the SMV model checking system. The integration of the circular proof rule with various reduction techniques, including *symmetry reduction*, *temporal case splitting* and *data type reduction* makes it possible to verify liveness of systems with unbounded arrays, as we illustrate, using Tomasulo's algorithm as an example. The approach is only sketched here – a more extensive treatment can be found in [1].

L. Pierre and T. Kropf (Eds.): CHARME'99, LNCS 1703, pp. 342–346, 1999.

2 Circular Compositional Proofs

Using the standard linear temporal logic (LTL) as our framework, we will formalize an inference rule for circular compositional reasoning. Note that there is no notion of process or process composition in this system. As in, for example, TLA [5], a process is simply viewed as a temporal proposition.

Now, suppose that we have a collection of formulas P, and we would like to prove Gp for all $p \in P$. We first fix a well founded order $\prec$ on the formulas in P. Intuitively, if $p \prec p'$ then we may assume p up to time t when proving p' at time t, otherwise we may assume p only up to time $t-1$. For any proposition $p \in P$, we denote by Δ_p the set of propositions assumed up to time t when proving p, and by Θ_p the set assumed at time t. Every element of Θ_p must be less than p, according to $\prec$. However, any $p' \in P$ (including p itself) may be an element of Δ_p. This is what allows us to construct (apparently) circular arguments. The notion that "p' up to time $t-1$ implies p at time t" can be expressed in as a formula in LTL, as in the following theorem:

Theorem 1. *Given sets Γ, P of formulas, a well founded order $\prec$ on P, and, for all $p \in P$, sets $\Delta_p \subseteq \Theta_p \subseteq P$, such that $q \in \Delta_p$ implies $q \prec p$, if*

$$\models \Gamma \Rightarrow \neg(\Delta_p \; U \; (\Theta_p \wedge \neg p))$$

for all $p \in P$, then $\models Gp$ for all $p \in P$.

That is, to prove Gp for all p, it suffices to prove $\Gamma \Rightarrow \neg(\Delta_p \; U \; (\Theta_p \wedge \neg p))$ for all p. Note that the latter are linear temporal formulas whose validity we can verify by model checking methods.

Proof Graphs The SMV system applies the above theorem in the context of a *proof graph*. This supports both assumption/guarantee style reasoning, where we assume A at all time to prove B at all time, and circular compositional reasoning, where we assume A only up to time t or $t-1$. A proof graph is a directed graph (V, E), where the vertices V are the propositions to be proved, and an edge $(p, p') \in E$ indicates that proposition p is to be assumed when proving p'. A subset $E^+ \subseteq E$ of the edges are identified as *unit delay edges*. If $(p, p') \in E^+$, then p is assumed only up to time $t-1$ when proving p' at time t. Although the proof graph may be cyclic, the graph $(V, E \backslash E^+)$ must have no infinite backward paths. If V is finite, this means that every cycle must contain at least one unit delay edge. Each strongly connected component of (V, E) corresponds to an application of theorem 1. Thus, every formula on a cycle must be of the form Gp, so that we may apply the theorem. Formulas not on a cycle may be of any form, however.

Proof obligations are constructed as follows. Suppose that some proposition $Gp \in V$ is on a cycle. Let C be the strongly connected component containing Gp. Let Δ_p be the set of all propositions $Gp' \in C$ such that $(Gp', Gp) \in E$. Let Θ_p be the subset of these such that $(Gp', Gp) \notin E^+$. Let Γ_p be the set of propositions q, such that $(q, Gp) \in E$ and $q \notin C$. On the other hand, if a proposition q is not on a cycle, then let Γ_q be the set of q' such that $(q', q) \in E$.

Theorem 2. *Let $\mathcal{C}$ be the union of the strongly connected components of (V, E). If*

- *for all $Gp \in \mathcal{C}$: $\sigma \models \Gamma_p \Rightarrow \neg(\Delta_p \; U \; (\Theta_p \wedge \neg p))$*
- *for all $q \notin \mathcal{C}$: $\sigma \models \Gamma_q \Rightarrow q$*

then for all $p \in V$: $\sigma \models p$.

In order to apply circular compositional reasoning in SMV, we have only to supply the set of properties to be proved and the proof graph. From these, the above theorem can be used to construct a sufficient set of proof obligations in the form of LTL formulas.

3 Verifying a Version of Tomasulo's Algorithm

As an example, we now consider how the circular compositional approach can be used to prove liveness of an implementation of Tomasulo's algorithm. This design, and its functional verification are described elsewhere in this volume [6]. Here, we assume familiarity with that material, and consider only the verification of liveness. In fact, the liveness proof follows the structure of the functional proof almost exactly.

We prove that an instruction in any reservation station eventually terminates. As in the functional proof, the first step is to break the problem into two lemmas. The first lemma states that the operands required by an instruction in a reservation station eventually arrive. The second states that the result of an instruction in a reservation station eventually returns on the result bus (that is, the reservation station is eventually cleared). We use operand liveness to prove result liveness and *vice versa*. Here is the SMV specification of the operand liveness lemma (for the `opra` operand):

```
forall (i in TAG)
  live1a[i] : assert G (st[i].valid -> F st[i].opra.valid);
```

That is, for all reservation stations i, if station i is `valid` (contains an instruction) then its `opra` operand will eventually be valid (hold a value and not a tag). A similar lemma is stated for the `oprb` operand. The result liveness lemma is just as simply stated:

```
forall (i in TAG)
  live2[i] : assert G (st[i].valid -> F~st[i].valid);
```

That is, for all reservation stations i, if station i is `valid` it is eventually cleared. Note that the reservation station is cleared when its result returns on the result bus.

Operand Liveness We use the same case splits for the liveness proof as for the functional proof. That is, to prove liveness of operands arriving at consumer reservation station k, we consider a particular producer reservation station i and a particular intermediate register j. To prove a given case, we need to use only reservation stations i and k, and register j. This case split is specified in SMV as follows (for the `opra` operand):

```
forall(i,k in TAG; j in REG)
  subcase live1a[k][i][j] of live1a[k]
    for st[k].opra.tag = i & aux[i].srca = j;
```

To prove that operands of consumer reservation station k eventually arrive, we have to assume that the producer reservation station i eventually produces a result. On the other hand, we also have to assume that the operands of an instruction eventually arrive in order to prove that it eventually produces a result. This is where the circular compositional rule comes into play. Note that the producer instruction always enters the machine at least one time unit before the consumer instruction. Thus, to prove that the consumer operand eventually arrives for instructions arriving at time t, it is

sufficient to assume that results eventually arrive for producer instructions arriving up to time $t-1$. Thus, we add a unit arc to the proof graph, as follows (for the `opra` operand):

```
forall (i,k in TAG; j in REG) using (live2[i]) prove live1a[k][i][j];
```

That is, when proving `live1a[k][i][j]` at time t, we assume `live2[i]` up to time $t-1$ (parentheses indicate the unit delay). As in the functional verification, the default data type reductions automatically eliminate all but reservation stations i, k and register j, and also reduce the types of register indices and tags to two and three values respectively. Symmetry automatically reduces the n^3 cases we need to verify to just 2 representative cases (for $i = k$ and $i \neq k$).

Result Liveness For the result liveness lemma, we again consider the possible paths of a data item from producer to consumer. In this case, operands are sent from reservation station i to some execution unit j. The result then returns on the result bus tagged for reservation station i, which in turn clears the reservation station. We would therefore like to split into cases based on the execution unit. This presents a problem, since at the time the instruction enters the reservation station, the execution unit is not yet determined. Nonetheless, it is possible to split cases on a future value of variable, using the following declaration:

```
forall(i in TAG; j in EU)
  subcase live2[i][j] of live2[i]
    for (aux[i].eu = j) when st[i].issued;
```

That is, for each execution unit j, a result must eventually arrive if the instruction *will* be in execution unit j at the next time the reservation station is found to be in the `issued` state. Note that `when` simply is a derived temporal operator. SMV recognizes that at any time `v = i when q` must be true for at least one value of i, and thus that the given set of cases is complete.

To prove that the instruction in a given execution unit eventually terminates, we must assume that its operands eventually arrive. Thus, we add the following arcs to the proof graph:

```
forall (i in TAG; j in EU) using live1a[i],live1b[i] prove live2[i][j];
```

Note that here, we do not use a unit delay arc. This is allowable, since all the cycles in the proof graph are broken by a unit delay arc. Note, we must also apply appropriate fairness constraints to the arbiters in the design to prove liveness. This is discussed in detail in [1].

Verification The result of applying the above described proof decomposition is a set of proof subgoals that can be solved by model checking. Our implementation is shown to satisfy the given lemmas for an arbitrary number of registers, reservation stations, and execution units. All told, there are 5 model checking subgoals, with a maximum of 20 state variables. The overall verification time (including the generation of proof goals and model checking) is 3.2 CPU seconds (on 233MHz Pentium II processor).

4 Conclusions

Mutual dependence of liveness properties does occur in practice, for example in processors with multiple execution units. The standard assumption/guarantee approach to compositional verification cannot be followed in such a situation. However, we can use an appropriate circular compositional rule, combined with model checking, to prove such mutually dependent liveness properties by induction over time. Such a rule was obtained by extending the technique for safety verification in [7].

References

1. http://www-cad.eecs.berkeley.edu/~kenmcmil/papers/1999-02.ps.gz, Feb. 1999. 342, 345
2. M. Abadi and L. Lamport. Composing specifications. *ACM Trans. on Prog. Lang. and Syst.*, 15(1):73–132, Jan. 1993. 342
3. R. Alur and T. A. Henzinger. Reactive modules. In *11th annual IEEE symp. Logic in Computer Science (LICS '96)*, 1996. 342
4. R. Alur, T. A. Henzinger, F. Mang, S. Qadeer, S. K. Rajamani, and S. Tasiran. Mocha: Modularity in model checking. In *CAV '98*, number 1427 in LNCS, pages 521–25. Springer-Verlag. 342
5. L. Lamport. The temporal logic of actions. Research report 79, Digital Equipment Corporation, Systems Research Center, Dec. 1991. 343
6. K. L. McMillan. Verification of infinite state systems by compositional model checking. this volume. 344
7. K. L. McMillan. Verification of an implementation of Tomasulo's algorithm by compositional model checking. In *CAV '98*, number 1427 in LNCS, pages 100–21. Springer-Verlag, 1998. 342, 346