Lazy Compositional Verification[a]

N. Shankar
shankar@csl.sri.com
URL: http://www.csl.sri.com/~shankar/

Computer Science Laboratory
SRI International
Menlo Park, CA

[a]Sponsored by the ARPA Contract No. A721, AFOSR under grant number F49620-95-C0044. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of ARPA, AFOSR, or the U.S. Government. Comments of John Rushby, Shaz Qadeer, Pierre Collette, Sam Owre, and Harald Ruess have been helpful.

Overview

Logics for the verification/refinement of reactive systems are typically *non-compositional*.

System properties cannot be established from component properties.

Compositionality is a key enabling technique along with abstraction, invariant generation, and model checking.

Existing compositional methods are too baroque.

Simple compositional principles are needed for effective mechanization.

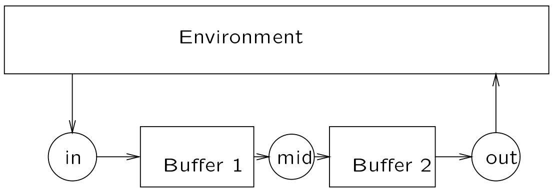
Outline

- Motivating problems
- Existing compositional methods
- Lazy composition
- Examples
- Conclusions
- **Note:** Only dealing with *specifications* not *programs*.

Motivating Example: Buffers

Show the main invariant of an unbounded buffer

is satisfied by the composition of two smaller buffers [Collette93]:



Lazily Composed Buffers

- Initially: b, in, out are empty, as are the input and output histories inh and $outh$.
- Each buffer transition must leave inh unchanged and can either:
  - read the input value from in, add it to b, and set in to $empty$ leaving out, $outh$ unchanged, or
  - write/push the first value in b to $out/outh$ provided out is $empty$.
- An environment transition must leave b and $outh$ unchanged, and either
  - write/push a value into in/inh when in is $empty$ leaving out unchanged, or
  - read out when nonempty, set it to $empty$ leaving in, inh unchanged.

Specification yields invariant

$$inh = in \bullet b \bullet outh$$

Composed Buffers

Given two buffers:

- Buffer 1 with in, inh, b_1, mid, and $midh$, and
- Buffer 2 with mid, $midh$, b_2, out, and $outh$.

Buffer 1 has no constraint regarding out, $outh$ — cannot discharge proof obligations.

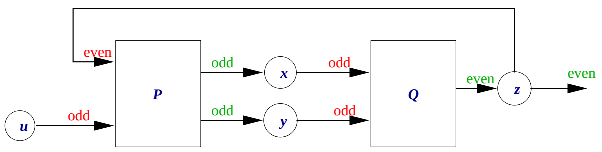
Similarly for Buffer 2, in, and inh.

Each buffer satisfies the buffer invariant.

By composition, the composed buffer also satisfies each buffer invariant, and the conjunction of the invariants yields the composite buffer invariant:

$$inh = in \bullet b_1 \bullet mid \bullet b_2 \bullet outh.$$

Another Example: Odd–Even



Need to show that if the environment delivers odd inputs u, the system $P \| Q$ delivers even outputs z.

Example can also be modeled as a Moore or Mealy machine.

Also show that P can implemented as $x' = u + z$ and $y' = u - z$ with u, z unchanged, and Q as $z' = x + y$ with x, y, u unchanged.

Glossary

System: Generates computations

Computations: Infinite sequence of observable global states.

Component: Subset of constraints on system computations.

Property: Set of global states (branching time) or set of computations (linear time).

Open system: Interacts with environment through shared variables, signals, or messages.

Closed system: No external *interaction*.

Safety property: Fails on a computation if it fails on all suffixes of some finite prefix.

Glossary (contd.)

Liveness property: Holds of some suffix of any finite computation.

Fairness constraint: Primitive progress assertions.

Machine closure: Fairness constraints only disqualify infinite behaviors.

- Every finite computation can be extended to an infinite fair computation.

Receptiveness: Fairness constraints can be met without cooperation from the environment.

Models of Computation/Composition

Shared variable programs: $P \| Q$ consists of the fair interleaving of P and Q actions.

Synchronous message-passing: fair interleaving of P and Q actions with synchronization on matching communication steps in P and Q.

Sequential composition: $P \| Q$ is $P; Q$.

Moore machines: Module state synchronously updated using inputs. Output is a function of local state. $P \| Q$ wires inputs/outputs.

Mealy machines: State at time $t+1$ is determined by inputs at time t and $t+1$ and state at time t. Combinational loops disallowed.

The Owicki-Gries Method

Non-compositional method for verifying shared variable programs.

Prove $P \| Q \vdash A$ by showing

- $P \vdash A$ and
- $Q \vdash A$

where A is an invariant assertion.
$P \vdash A$ means A is inductive for P.

Buffer example: environment and other buffer must not interfere with individual buffer invariant.

Extended to message-passing by Levin–Gries and Apt–Francez–de Roever.

Made compositional for message-passing model by Zwiers, Hooman.

Abadi–Lamport: Composition

Component P written in TLA as $I.P \wedge \Box[N.P]_{\langle m,x\rangle} \wedge L.P$, where $L.P$ is receptive.

$[N]_{\langle m,x\rangle}$: Each computation step is N or leaves output variables m and internal variables x unchanged.

Programs P, properties C, environment constraints E are all TLA formulas.

Composition $P \| Q$ is conjunction.

$\overline{P}$ is smallest safety property containing P.

$E \overset{+}{\longrightarrow\!\!\triangleright} C$: Holds of σ if $\sigma \in E \Rightarrow C$ and C holds at least one step longer than E.

Composition Theorem

If component P has property $E_P \overset{+}{\longrightarrow\!\!\triangleright} C_P$ and Q has property $E_Q \overset{+}{\longrightarrow\!\!\triangleright} C_Q$.

Want to show that $P \| Q$ has property $E \overset{+}{\longrightarrow\!\!\triangleright} C$.

Need to establish:

1. $\overline{E} \wedge \overline{C_P} \wedge \overline{C_Q} \Rightarrow E_P \wedge E_Q$

2. $E \overset{+}{\longrightarrow\!\!\triangleright} (C_P \wedge C_Q \Rightarrow C)$

Why? If C fails of $P \| Q$, then by condition 2, either

- E fails first

- or C_P (C_Q) fails, then E_P (E_Q) must have failed earlier by $E_P \overset{+}{\longrightarrow\!\!\triangleright} C_P$, violating the first condition.

Abadi–Lamport: Decomposition

The goal is to show that $P_I \| Q_I$ implements $P \| Q$ by separate refinements.

To show $E \overset{+}{\longrightarrow\!\!\triangleright} (P_I \| Q_I \Rightarrow P \| Q)$:

- $\overline{E} \wedge P \wedge Q \Rightarrow E_P \wedge E_Q$

- $E_P \overset{+}{\longrightarrow\!\!\triangleright} (P_I \Rightarrow P)$

- $E_Q \overset{+}{\longrightarrow\!\!\triangleright} (Q_I \Rightarrow Q)$

Abadi–Lamport: Critique

Noninterleaving composition is expensive: composing m actions with n actions yields $m \times n$ actions.

Handling of interleaving with *Disjoint* predicate is awkward.

Environment assumptions are *properties* that can constrain both system and environment.

Guarantees have to be constrained to be falsifiable by at most one of system or environment.

Choice of guarantee properties requires undue prescience for proof obligations to be provable.

Lazy Composition

Lazy composition is applied to shared variable transition systems using a TLA-like framework.

Component P given as $\langle I.P, N.P, F.P \rangle$ with initial state predicate $I.P$, next-state relation $N.P$, and (machine-closed) fairness constraints $F.P$.

Closed systems represented as $[P]$.

Composition $P \| Q$ is interleaving: $\langle I.P \wedge I.Q, \quad N.P \vee N.Q, \quad F.P \wedge F.Q \rangle$.

Properties C of P are proved by closing under an abstract environment E_P: $(P, E_P) \models C$, where $(P, E_P) \equiv [P \| E_P]$.

Need C to be preserved when E_P is replaced by Q.

Conjunction

Refinement $P \leq P'$ preserves the property: if $[P'] \models C$ then $[P] \models C$.

For open systems, $P \leq P'$ if for all E, $[P \| E] \leq [P' \| E]$.

Conjunction is the weakest common refinement: $P \wedge Q \equiv \mathit{weakest}\ L\colon L \leq P \wedge L \leq Q$.

In particular, $P \wedge Q \equiv \langle I.P \wedge I.Q,\quad N.P \wedge N.Q,\quad F.P \wedge F.Q \rangle$.

Lazy Composition

Closed composition $(P, E_P) \otimes (Q, E_Q)$ is $[P \wedge E_Q \| Q \wedge E_P]$.

If $(P, E_P) \models C$, then $(P, E_P) \otimes (Q, E_Q) \models C$.

Open composition $(P, E_P) \times (Q, E_Q)$ is $[(P \wedge E_Q) \| (Q \wedge E_P) \| (E_P \wedge E_Q)]$.

Property preservation: If $(P, E_P) \models C$, then $(P, E_P) \times (Q, E_Q) \models C$.

Note composition is not (necessarily) conjunction.

Lazily Composed Buffers

Buffer 1 consists of the initialization $I.B_1$, and next state relation $N.B_1$.

Environment E_{B_1}: only a next state relation $N.E_{B_1}$ asserting in, inh, mid, and $midh$ are properly updated.

Similarly for B_2.

The individual buffer invariants:

$$(B_1 \| E.B_1) \models \Box\ inh = in \bullet B_1 \bullet midh$$
$$(B_2 \| E.B_2) \models \Box\ midh = mid \bullet B_2 \bullet outh$$

trivially yield invariant

$$\Box\ inh = in \bullet b_1 \bullet mid \bullet b_2 \bullet outh$$

for $(B_1 \| E_{B_1}) \times (B_2 \| E_{B_2})$.

Lazy Composition: Advantages

Environment specified by allowed transitions; not temporal properties.

Environment constraints apply only to environment and not the system.

Composition merely accumulates constraints on each component — no assume-guarantee proof obligations.

No need to anticipate strong guarantee properties.

Accumulated constraints are discharged during refinement.

All component properties are preserved by composition.

Global properties can be proved by examining a subset of components using previously proved global or local properties.

Lazy Composition: Disadvantages

Composition can yield an inconsistent specification.

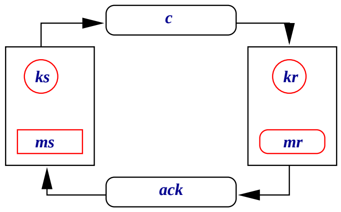
Consistency of the final composite specification has to be established by refining it to a program.

Refinement cannot always be done compositionally.

In practice, however, stronger refinements are possible since any global invariant can be used in justifying a local refinement.

Alternating Bit Protocol

Consists of sender S and receiver R communicating via lossy FIFO channels c and ack.



The Sender

Sender S: Each transition enqueues a message/number pair on channel c or dequeues an ack incrementing counter ks.

$I.S \equiv c = null \wedge ks = 1$

$N.S \equiv$

$$
\begin{array}{lll}
 & (& c' = enq((\#data := ms(ks), num := ks\#), c) \\
 & \wedge & ks' = ks) \\
\vee & (& nonempty?(ack) \\
 & \wedge & ack' = deq(ack) \\
 & \wedge & ks' = (if\ \ ks = fst(ack)\ then\ ks + 1\ else\ ks))
\end{array}
$$

Environment can lose or duplicate messages.

$N.E_S \equiv (ks' = ks \wedge c' \preceq c)$

The Receiver

Receiver can either enqueue an ack or dequeue messages and add to the messages received buffer mr.

$I.R \equiv ack = null \wedge kr = 0$

$N.R \equiv$

$$
\begin{array}{lllll}
 & ack' = enq(kr, ack) & & & \\
\vee & (& nonempty?(c) & & \\
 & \wedge & c' = deq(c) & & \\
 & & (& kr \neq num(fst(c)) & \\
 & \wedge & \supset & (& kr' = kr + 1 \\
 & & & \wedge & mr'(kr+1) = data(fst(c)))) \\
 & \wedge & \multicolumn{3}{l}{kr = num(fst(c)) \supset (kr' = kr \wedge mr' = mr))}
\end{array}
$$

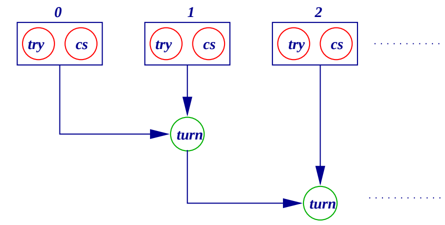
Environment can lose/duplicate ack's.

$$
\begin{array}{lll}
N.E_R \equiv & & kr' = kr \\
 & \wedge & mr' = mr \\
 & \wedge & ack' \preceq ack
\end{array}
$$

Invariants for $(S, E_S) \times (R, E_R)$

1. $(\forall m \in c\colon\ num(m) \leq ks$
   $\wedge\ data(m) = ms(num(m)))$
   $\wedge\ ordered?(c)$
   From sender (S, E_S).
2. $(\forall a \in ack\colon a \leq kr) \wedge ordered?(ack)$
   From receiver (R, E_R).
3. $(c' = enq((\#data{:=}\ ms(ks), num{:=}\ ks\#), c)$
   $\vee\ c' \preceq c)$
   $\wedge\ (ks' = ks \vee ks' = ks + 1)$
   From sender.
4. $(kr' = kr \vee kr' = kr + 1)$
   $\wedge\ \ (ack' \preceq ack \vee ack' = enq(kr, ack))$
   From receiver.
5. $(\forall m \in c\colon kr \leq num(m)) \wedge kr \leq ks$.
   From receiver, 1, and 3.
6. $(\forall a \in ack\colon ks \leq a + 1) \wedge ks \leq kr + 1$.
   From sender, 2, and 4.
7. $(\forall i \leq kr\colon mr(i) \leq ms(i))$.
   From receiver, 5, and 6.

N-Process Mutual Exclusion



In round i, the winner j, $0 \leq j < i$ with $cs(j) = i$ competes with process i using 2-process mutual exclusion with variable $turn(i)$.

Winner increments value of cs.

For N processes $0 \ldots N-1$, the winner is the sole process j with $cs(j) = N$.

Proved by induction, abstraction, and model checking.

Environment constraints needed to ensure that $turn(i)$ value is not changed improperly.

Grumberg–Long: Model Checking $\forall$CTL*

Family of compositionality principles for verifying $\forall$CTL* properties of finite or infinite-state Moore machines.

$$\frac{\begin{array}{c} P \leq P' \\ P' \| Q \models D \\ \mathcal{T}(D) \| P \models C \end{array}}{P \| Q \models C}$$

$\mathcal{T}(D)$ is the tableau for formula D.

Does not handle mutual dependencies.

Similar technique used by Larsen–Liu, Anderson–Stirling–Winskel, to reduce $P \| Q \models A$ to $P \models A/Q$.

Alur–Henzinger: Reactive Modules

$P \| Q$ is language intersection.

$P \leq P'$ is language containment.

Compositionality: If $P \| Q' \leq P'$ and $P' \| Q \leq Q'$, then $P \| Q \leq P' \| Q'$.

$P \leq_F P'$: Fair traces of P contained in P'. Implies $P \leq P'$ by receptiveness

Fair compositionality: If $P \| \overline{Q'} \leq_F P'$ and $P' \| Q \leq_F Q'$ then $P \| Q \leq_F P' \| Q'$.

Since $P \| \overline{Q} \leq P' \| \overline{Q'}$, a fair trace in $P \| Q$ is a trace of $\overline{Q'}$ is a fair trace of P' and hence Q'.

Reactive modules capture synchronous systems (Mealy machines) with finite nondeterminism. (See also McMillan[CAV'97].)

Observations

Lazy composition similar to Grumberg–Long method: properties verified by composing with environment automaton.

Lazy composition can be combined with eager composition: can discharge environment constraints using global invariants.

Lazy composition has been used to verify a real-time railroad crossing controller composed of three subsystems: trains, the controller, and the gate.

The simplicity of lazy composition makes it easy to mechanize and use in conjunction with theorem proving and model checking.

But care is needed in formulating environment constraints that are not too strong.

Conclusion

Existing compositional methods for modular verification are mathematically interesting and deep, but too complex for practical use.

In particular, global prescience is required in formulating guarantees that can discharge constraints imposed by other modules.

Lazy composition makes composition easy at the expense of potential inconsistency.

Lazy composition is easier to use and mechanize, but not easy enough.

Both approaches need to be adapted to specific models of computation.