Compositional Reasoning

Sérgio Campos, Edmund Clarke

Introduction and Motivation

Symbolic model checking has been very successful in verifying industrial circuits.

However, large complex systems sometimes cannot be verified because of the state explosion problem.

State explosion is most frequently caused by the parallel composition of processes in the system.

Efficient methods for compositional verification can extend the applicability of formal verification methods to even larger systems.

Introduction and Motivation

- Synchronous X Asynchronous composition
  - Partitioned transition relations
- Cone of influence reduction
- Interface processes
- Assume guarantee

The Model

Variables in the model are $\mathcal{VAR} = \{v_0, v_1, \ldots, v_n\}$.

A finite state-transition graph models the system:

- A state V is defined by an assignment of values to the variables in $\mathcal{VAR}$.
- The transition relation is described in terms of two sets of variables:
  - Unprimed for the current state.
  - Primed for the next state.

$$N(V, V')$$

Composition of Processes

Frequently the system is described by a set of processes $P = \{P_0, P_1, ..., P_{n-1}\}$ that execute concurrently.

The transition relation N is constructed from the transition relation of each process N_i:

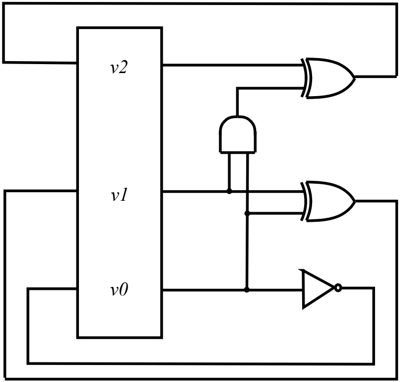
- Each process defines the value of certain variables in the next state as a function of values in the current state:

$$v_i' = f_i(V).$$

- These equations are used to define the relations:

$$N_i(V, V') = (v_i' \Leftrightarrow f_i(V)).$$

Example: Mod8counter



- $N_0 = (v_0' \Leftrightarrow \neg v_0)$
- $N_1 = (v_1' \Leftrightarrow v_0 \oplus v_1)$
- $N_2 = (v_2' \Leftrightarrow (v_0 \wedge v_1) \oplus v_2)$

Synchronous Composition

In the synchronous model all processes $P_0 \ldots P_{n-1}$ execute at each step.

The conjunction of all N_is forms the transition relation:

$$N(V, V') = N_0(V, V') \wedge \cdots \wedge N_{n-1}(V, V').$$

Asynchronous Composition

In the asynchronous model, only one process executes at a time, and all others maintain the values of their variables.

$$N_i(V, V') = \left(v_i' \Leftrightarrow f_i(V)\right) \wedge \bigwedge_{j \neq i} (v_j' \Leftrightarrow v_j).$$

Consequently, the disjunction of all N_is forms the transition relation:

$$N(V, V') = N_0(V, V') \vee \cdots \vee N_{n-1}(V, V'),$$

Pre-Image Computation

One of the most expensive operations in model checking is computing the set of predecessors of a set of states S.

It is computed by the relational product:

$$\exists V' \left[S(V') \wedge N(V, V') \right].$$

where $\exists V'$ is the existential quantification of all variables in V'.

Partitioned Transition Relations

However, the size of N can be significantly larger than the sum of the sizes of all N_is.

The goal is to implicitly conjunct (or disjunct) the N_is for image computation without constructing N.

Disjunctive Partitioning

For a disjunctive partitioned transition relation, the relational product computed is of the form

$$\exists V' \left[S(V') \wedge (N_0(V, V') \vee \cdots \vee N_{n-1}(V, V')) \right].$$

It can be computed by distributing the existential quantification:

$$\begin{array}{l} \exists V' \left[S(V') \wedge N_0(V, V') \right] \vee \cdots \vee \\ \exists V' \left[S(V') \wedge N_{n-1}(V, V') \right] \end{array}$$

Much larger circuits can be verified using this representation than with traditional methods.

Conjunctive Partitioning

The relational product computed has the form

$$\exists V' \left[S(V') \wedge (N_0(V, V') \wedge \cdots \wedge N_{n-1}(V, V')) \right].$$

However, existential quantification does not distribute over conjunction!

$$\exists a[(a \vee b) \wedge (\neg a \vee c)] \not\equiv \exists a[(a \vee b)] \wedge \exists a[(\neg a \vee c)]$$

It reduces to:

$$[b \vee c] \not\equiv true$$

Conjunctive Partitioning (cont.)

We can still apply partitioning because:

- Circuits exhibit locality: most N_is depend on only a small number of variables in V and V'.
- Subformulas can be moved out of the scope of existential quantification if they do not depend on any of the variables being quantified:

$$\exists a[(a \vee b) \wedge (b \vee c)] \equiv \exists a[(a \vee b)] \wedge (b \vee c)$$

Conjunctive Partitioning (cont.)

We can compute the relational product using early quantification for variables in each N_i:

- Choose an order in which to consider partitions for early quantification ρ.
- D_i is the set of variables process P_i depends on.
- E_i is the set of variables that process P_i depends on that processes considered later in the ordering do *not* depend on, i.e.,

$$E_i = D_{\rho(i)} - \bigcup_{k=i+1}^{n-1} D_{\rho(k)}.$$

Example:

ρ :	P_0	P_1	P_2
Depends on	$\{a, b, c, d\}$	$\{b, c\}$	$\{c, d\}$
E_i :	$\{a\}$	$\{b\}$	$\{c, d\}$

Computing the Relational Product

We now can compute the relational product by:

$$
\begin{aligned}
S_1(V, V') &= \exists_{v \in E_0} \left[S(V)' \wedge N_{\rho(0)}(V, V') \right] \\
S_2(V, V') &= \exists_{v \in E_1} \left[S_1(V, V') \wedge N_{\rho(1)}(V, V') \right] \\
&\vdots \\
S_n(V') &= \exists_{v \in E_{n-1}} \left[S_{n-1}(V, V') \wedge N_{\rho(n-1)}(V, V') \right].
\end{aligned}
$$

Intuitively

$$
\underbrace{\underbrace{\exists V' \underbrace{\left[S(V') \wedge (N_0(V, V')}_{S_1} \wedge N_1(V, V')) \wedge \cdots \right]}_{S_2}}_{S_n}
$$

Conjunctive Partitioning (cont.)

Problem with partitioned transition relations:

- Extremely sensitive to the order in which partitions are considered.
- However, there are heuristics to assist in determining a good order.

Lazy Parallel Composition

During pre-image computation, usually only a small subset of transitions is considered.

We can use this observation to simplify each N_i *before* computing the relational product.

Composing the simplified N_is can generate significantly smaller transition relations and speed up verification.

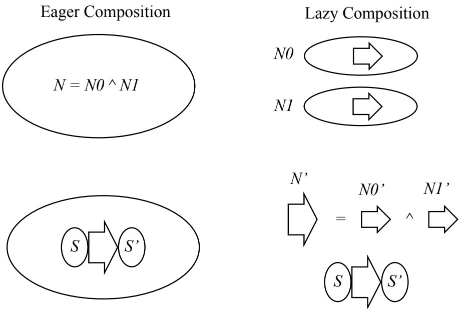
The Lazy Pre-Image

- Simplify each N_i: Determine N'_i agreeing with N_i on transitions satisfying S:

$$N'_i(V, V') = N_i(V, V') \mid_S$$

- Compose all N'_is into a simplified N':

$$N' = N'_0(V, V') \wedge N'_1(V, V') \wedge \cdots \wedge N'_{n-1}(V, V')$$



The Constrain Operator

$constrain(f, g)$ is a BDD that:

- Agrees with f for valuations that satisfy g.
- Has an undetermined value for valuations that do not satisfy g.
- Is (hopefully) smaller than f.

Consequently, the restricted transition relation N' is a transition relation that:

- Preserves transitions that start in S.
- Does not necessarily preserve other transitions.
- Is smaller than N.

[Coudert,Berthet,Madre 89]

Partitioned vs. Lazy Composition

Lazy parallel composition is less sensitive to partition ordering:

- Partitioned transition relations: step i depends on step $i-1$

$$\underbrace{\exists v_0 \left[\underbrace{\exists v_1 \left[S(V') \wedge N_0(V, V') \right]}_{\text{step1}} \wedge N_1(V, V') \right]}_{\text{step2}}$$

- Lazy parallel composition: independent steps.

$$\exists V' \left[S(V') \wedge (\underbrace{N_1(V, V') \mid_S}_{\text{step1}} \wedge \underbrace{N_2(V, V') \mid_S}_{\text{step2}}) \right].$$

Cone of Influence Reduction

We can compute $P|_\sigma$ using the *cone of influence*:

- Assume the system is specified by a set of equations:

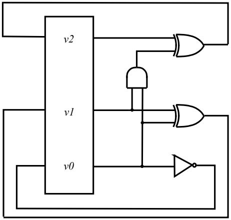
$$v_i' = f_i(V).$$

- Variables in the cone of influence C_i of $v_i \in \sigma$:
  - v_i,
  - v_j if $\exists v_l \in C_i$ such that f_j depends on v_l.
- Construct a new model P':
  - Variables in P' are the variables in all C_i.
  - The transition relation is constructed by removing equations for variables not in any C_i.

Show that $P \models \varphi$ iff $P' \models \varphi$.

Cone of Influence Example

Given the modulo 8 counter:



- $v_0' = \neg v_0$
- $v_1' = v_0 \oplus v_1$
- $v_2' = (v_0 \wedge v_1) \oplus v_2$

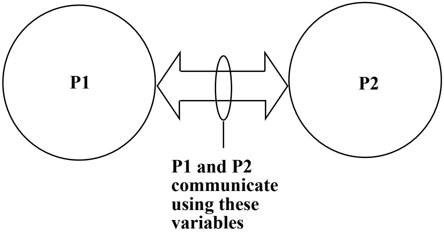
We have $C_1 = \{v_0, v_1\}$ because:

- $v_0 \in C_1$ because f_1 depends on v_0,
- $v_1 \in C_1$ because f_1 depends on v_1,
- $v_2 \notin C_1$ because no variable in C_1 depends on v_2.

Interface Processes

An important observation leads to another approach to compositional verification:

- The communication between processes is well defined and usually involves a small number of variables.



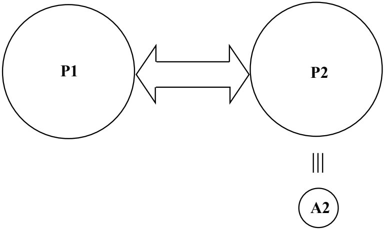
Interface Processes (cont.)

Assume two processes P_1 and P_2 communicate using a set of variables σ.

P_1 can only observe the behavior of P_2 through σ.

We can replace P_2 by an equivalent process A_2 that is indistinguishable from P_2 with respect to σ.

- A_2 is usually simpler than P_2 because it hides all events that do not relate to σ.



Interface Processes (cont.)

The *interface rule* guarantees the correctness of A_2:

($P|_\sigma$ is the restriction of P to the variables in σ)

If the following conditions are satisfied,

- $P_2|_\sigma \equiv A_2$,
- $P_1||A_2 \models \varphi$,
- φ is a CTL formula such that $\varphi \in \mathcal{L}(\sigma)$,

Then φ is also true in $P_1||P_2$.

Soundness Conditions

The soundness of the interface rule depends on:

- Suppose $\Sigma_{P_1} = \Sigma_{P_2}$, then $P_1 \equiv P_2$ implies $\forall \varphi \in \mathcal{L}(\Sigma_{P_1})[P_1 \models \varphi \leftrightarrow P_2 \models \varphi]$
- If $P_1 \equiv P_2$ then $P_1||Q \equiv P_2||Q$ and $Q||P1 \equiv Q||P_2$
- $(P_1||P_2)|_\sigma \equiv P_1||(P_2|_\sigma) and (P_1||P_2)|_\sigma \equiv (P_1|_\sigma)||P_2$
- If $\varphi \in \mathcal{L}(\Sigma)$ and $\Sigma \subseteq \Sigma_P$, then $P \models \varphi$ iff $P|_{\Sigma_\varphi} \models \varphi$

where

- Σ_P is the set of atomic propositions in P,
- $\mathcal{L}(\Sigma)$ is the language of temporal formulas over alphabet Σ.

Proof of Soundness

1. $P_2|_\sigma \equiv A_2$, so $P_1||A_2 \equiv P_1||(P_2|_\sigma)$.
2. $P_1||(P_2|_\sigma) \equiv (P_1||P_2)|_\sigma$.
3. $P_1||A_2 \equiv (P_1||P_2)|_\sigma$.
4. $P_1||A_2 \models \varphi$, so $(P_1||P_2)|_\sigma \models \varphi$.
5. Since $\varphi \in \mathcal{L}(\sigma)$, we conclude $P_1||P_2 \models \varphi$ as required.

Definition of Equivalence

The interface processes methods relies on the *equivalence* relation.

For the logic CTL we define equivalence using:

- *Bisimulation equivalence*
  - Synchronous systems
  - Equivalence with respect to time
- *Stuttering equivalence*
  - Asynchronous systems
  - Allows different number of steps in each system

There are "efficient" polynomial algorithms to determine equivalence between processes in both cases.

Bisimulation Equivalence

Given a model with a set of states 2^{Σ} and transition relation N, two states s and t are equivalent iff

- $\forall s'[N(s,s') \text{ implies } \exists s''[N(t,s'') \wedge (s' \equiv s'')]]$
- $\forall s''[N(t,s'') \text{ implies } \exists s'[N(s,s') \wedge (s' \equiv s'')]]$

where $s' \in 2^{\Sigma}, s'' \in 2^{\Sigma}$.

Stuttering Equivalence

We define:

- $\tau_\sigma(s, t)$ iff s and t agree on the value of the all variables in σ.
- $N_S(s, t)$ iff $\exists \pi = s_0, s_1, \ldots, s_n$ such that $s_0 = s$, $s_n = t$ and $\forall 0 < i < n[\tau_\sigma(s_{i-1}, s_i)]$.

We now use the same definition as bisimulation equivalence, but using N_S instead of N:

Given a model with a set of states 2^Σ, a transition relation N and a "stuttering" transition relation N_S, two states s and t are equivalent iff

- $\forall s'[N_S(s, s')$ implies $\exists s''[N_S(t, s'') \wedge (s' \equiv s'')]]$
- $\forall s''[N_S(t, s'')$ implies $\exists s'[N_S(s, s') \wedge (s' \equiv s'')]]$

where $s' \in 2^\Sigma, s'' \in 2^\Sigma$.

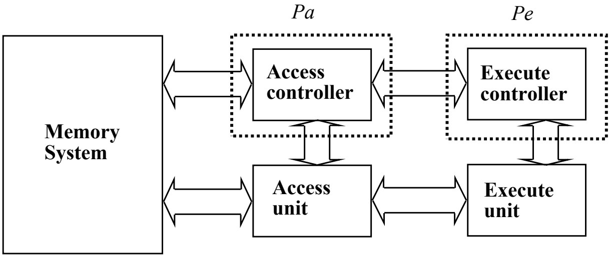
Interface Processes Example

A CPU controller with two units:

- The access unit P_a: Fetches instructions and stores them in an instruction queue.
- The execution unit P_e: Interprets machine code.

Using the interface process A_{P_e} we have been able to verify $P_a || A_{P_e}$:

- The number of states in $P_a || A_{P_e}$ is ten times smaller than $P_a || P_e$.



Assume Guarantee Reasoning

- Generalizes interface processes because it allows the definition of interfaces using:
  - Automata,
  - Temporal logic formulas.
- The goal is to use knowledge about the environments of the individual processes to reason compositionally about the concurrent system.

Assume Guarantee Reasoning

- Works with triples $\langle\varphi\rangle M\langle\psi\rangle$

  "If the system satisfies φ and contains M, then it also satisfies ψ."

- Typical example of assume-guarantee reasoning:

$$\frac{\langle true\rangle M\langle\varphi\rangle \quad \langle\varphi\rangle M'\langle\psi\rangle}{\langle true\rangle M \mid M'\langle\psi\rangle}$$

Implementing Assume-Guarantee

- Consider the assume-guarantee proof

$$\frac{\langle true \rangle M \langle \varphi \rangle \qquad \langle \varphi \rangle M' \langle \psi \rangle}{\langle true \rangle M \mid M' \langle \psi \rangle}$$

- In our framework, this corresponds to

$$\frac{M \models \varphi \qquad T_\varphi \mid M' \models \psi}{M \mid M' \models \psi}$$

Introduction and Motivation

- Synchronous X Asynchronous composition
  - Partitioned transition relations
- Cone of influence reduction
- Interface processes
- Assume guarantee