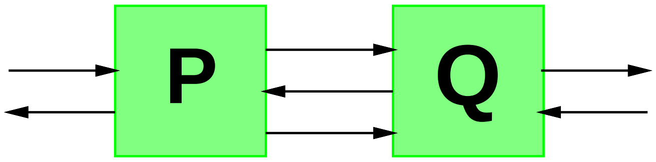
A Compositional Rule for Hardware Design Refinement

K. L. McMillan
Cadence Berkeley Labs
mcmillan@cadence.com

September 24, 1997

Compositional verification

- Problem: automated methods don't scale
- Solution: compositional approach
  - Divide-and-conquer strategy
  - Environment assumptions (assume/guarantee)



  - Circular inference rule (AL95):

$$\frac{\begin{array}{c} P, \phi \models \psi \\ Q, \psi \models \phi \end{array}}{P \parallel Q \models \psi \wedge \phi}$$

Outline

- A trivial compositional system
  - Allows "circular" proofs.
  - Lemmas discharged by model checking
  - Built in to SMV system
- Proof of implementation of Tomasulo's algorithm
  - History variables and refinement maps
  - Exploitation of symmetry

A trivial compositional system

- Let $\mathcal{S}$ be a finite set of *signals*
- Let $\mathcal{V}$ be a finite set of *values*
- Let a *model* be any function $\pi : \mathcal{S} \rightarrow \mathbb{N} \rightarrow \mathcal{V}$
- A *process* p is a predicate of the form:

$$\sigma_p(t) \in f(\gamma_1(t) \ldots \gamma_k(t))$$

  or

$$\begin{aligned} \sigma_p(0) &\in I \\ \sigma_p(t+1) &\in f(\gamma_1(t) \ldots \gamma_k(t)) \end{aligned}$$

- Composition: $p \parallel q \quad \equiv \quad p \wedge q$.

In SMV. . .

A conjunction of two processes in SMV:

```
init(count) := 0;
next(count) := ovfl ? 0 : (count + 1);

ovfl := (count = 3);
```

This conjunction of processes has exactly one model:

```
count = 0,  1,  2,  3,  0,  1, ...
ovfl  = 0,  0,  0,  1,  0,  0, ...
```

A nondeterministic model:

```
init(count) := 0;
next(count) := count + {0,1} mod 4;
```

Example model:

```
count = 0,  0,  1,  2,  3,  3,  0,...
```

Compositional proofs

Suppose we have two sets of processes, P and Q and want to prove:

$$(\bigwedge Q) \Rightarrow (\bigwedge P)$$

Would like to use some components of P to prove others, *e.g.*

$$\frac{\begin{array}{c} q_1 \wedge p_2 \Rightarrow p_1 \\ p_1 \wedge q_2 \Rightarrow p_2 \end{array}}{(q_1 \wedge q_2) \Rightarrow (p_1 \wedge p_2)}$$

But are all such proofs valid?

A valid example

Let $P = \{p_1, p_2\}$ be

p_1:: `x := 1;`
p_2:: `y := 1;`

and let $Q = \{q_1, q_2\}$ be

q_1:: `init(x) := 1; next(x) := y;`
q_2:: `y := x;`

Then we have the following:

$$\frac{q_1 \wedge p_2 \Rightarrow p_1 \ , \quad p_1 \wedge q_2 \Rightarrow p_2}{Q \Rightarrow P}$$

An invalid example

Let $P = \{p_1, p_2\}$ be

$$\begin{array}{ll} p_1:: & \texttt{x := 1;} \\ p_2:: & \texttt{y := 1;} \end{array}$$

and let $Q = \{q_1, q_2\}$ be

$$\begin{array}{ll} q_1:: & \texttt{x := y;} \\ q_2:: & \texttt{y := x;} \end{array}$$

The following is not valid:

$$\frac{q_1 \wedge p_2 \Rightarrow p_1 \ , \quad p_1 \wedge q_2 \Rightarrow p_2}{Q \Rightarrow P}$$

What went wrong?

Let $p\uparrow^{\tau}$ stand for $\forall t \leq \tau : \; p$.

The following is circular:

$$\frac{\begin{array}{ccc} p_1 \uparrow^{\tau} & \Rightarrow & p_2 \uparrow^{\tau} \\ p_2 \uparrow^{\tau} & \Rightarrow & p_1 \uparrow^{\tau} \end{array}}{p_2 \uparrow^{\tau} \quad \wedge \quad p_1 \uparrow^{\tau}}$$

However, this is sound by induction on τ:

$$\frac{\begin{array}{ccc} p_1 \uparrow^{\tau-1} & \Rightarrow & p_2 \uparrow^{\tau} \\ p_2 \uparrow^{\tau} & \Rightarrow & p_1 \uparrow^{\tau} \end{array}}{p_2 \uparrow^{\tau} \quad \wedge \quad p_1 \uparrow^{\tau}}$$

Note, this is our first example, since $x(t)$ didn't depend on $y(t)$.

So can we just rule out combinational cycles?

Another invalid example

Let $P = \{p_1, p_2\}$ be

$$\begin{array}{ll} p_1 :: & \texttt{x := 1;} \\ p_2 :: & \texttt{y := x;} \end{array}$$

and let $Q = \{q_1, q_2\}$ be

$$\begin{array}{ll} q_1 :: & \texttt{x := 0;} \\ q_2 :: & \texttt{y := 1;} \end{array}$$

The following is not valid:

$$\frac{q_2 \wedge p_2 \Rightarrow p_1 \;\; , \quad p_1 \wedge q_2 \Rightarrow p_2}{Q \Rightarrow P}$$

Could disallow this case [AH96][McM97] but it's useful. . .

Alternative solution

Let $\rightarrow$ be the "0-delay dependency graph" of $P \cup Q$.

By convention,

- if
$$\sigma_p \rightarrow \sigma_q \text{ and } \sigma_q \not\rightarrow \sigma_p$$
we use $p \uparrow^\tau$ to prove $q \uparrow^\tau$,

- else we use $p \uparrow^{\tau-1}$ to prove $q \uparrow^\tau$.

This well-founded order on processes guarantees a sound proof by induction over τ.

Note how previous counterexample is ruled out.

The formalities. . .

Definition 1. Zero-delay dependency. *Let P be a set of processes.*

- *Define $<_P \subseteq \mathcal{S} \times \mathcal{S}$ such that*

$$\gamma <_P \sigma$$

  exactly when there exists $p \in P$ of the form

$$\sigma(t) \in f(\ldots \gamma(t) \ldots)$$

- *Define $\prec_P \subseteq \mathcal{S} \times \mathcal{S}$ such that*

$$\gamma \prec_P \sigma \quad \Longleftrightarrow \quad \gamma <_P^* \sigma \ \wedge \ \neg(\sigma <_P^* \gamma)$$

More formalities. . .

Definition 2. Zero-delay antecedents. *Let* P *be a set of processes. For any* $p \in P$, *let* Z_P^p *be the set of* $p' \in P$ *s.t.*

- p' *is a gate, and*

- $p' \prec_P p$

A compositional rule

Theorem 1. *Given:*

1. *Two process sets P and Q, s.t.*

2. *Any choice of:*

   - *A "refinement" preorder $\hookrightarrow$ on P.*
   - *For each $p \in P$, sets*

$$\mathcal{E}_p^0 \subset Z_{P \cup Q}^p$$

$$\mathcal{E}_p^1 \subset (Z_{P \cup Q}^p)^c$$

   such that for all $q \in (\mathcal{E}_p^0 \cup \mathcal{E}_p^1)$:

   $\sigma_q = \sigma_p$ *implies* $q \hookrightarrow p$

The following rule is sound:

$$\frac{\text{for all } p \in P\text{: } \mathcal{E}_p^0 \uparrow^{\tau} \wedge\, \mathcal{E}_p^1 \uparrow^{\tau - 1} \quad \Rightarrow \quad p}{(\forall t:\ Q) \quad \Rightarrow \quad (\forall t:\ P)}$$

Why the refinement preorder?

Let $P = \{p_1, p_2\}$ be

$$\begin{array}{ll} p_1:: & \texttt{x := 1;} \\ p_2:: & \texttt{x := 1;} \end{array}$$

and let $Q = \{q_1\}$ be

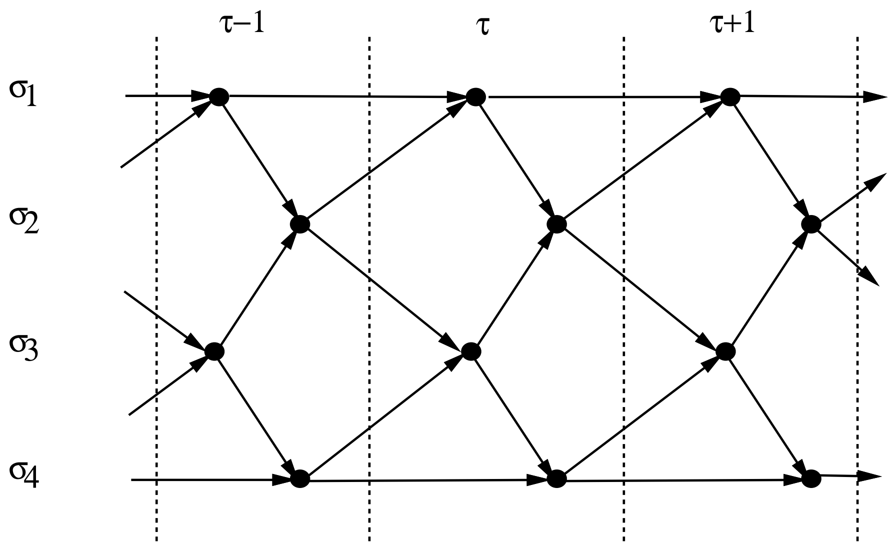
$$\begin{array}{ll} q_1:: & \texttt{x := 0;} \end{array}$$

Then w/o preorder requirement, we have the following:

$$\frac{q_1 \wedge p_2 \Rightarrow p_1 \ , \quad q_1 \wedge p_1 \Rightarrow p_2}{Q \Rightarrow P}$$

Proof of theorem

- Define a lexical order $(<, \prec)$ over $\mathbb{N} \times \mathcal{S}$:



- Assume $\pi \models Q$, $\pi \not\models P$ and choose τ, σ, p s.t.
  - $\pi \not\models p|_{t=\tau}$
  - (τ, σ) minimal w.r.t. $(<, \prec)$
  - p minimal w.r.t. $\hookrightarrow$
- Then $\pi \models \mathcal{E}_p^0 \uparrow^\tau, \mathcal{E}_p^1 \uparrow^{\tau-1}$
- Hence $\pi \models p \uparrow^\tau \quad \Rightarrow\Leftarrow$.

Implementation in SMV

- Processes (= assignments) grouped into layers.

```
layer spec : {
    x := 1;   y := 1;
}
layer impl : {
    init(x) := 1; next(x) := y;
    y := x;
}
```

- Refinement relation expressed on layers

```
impl refines spec;
```

- "Environments" may be specified

```
using   x//spec   prove   y//spec;
```

These declarations may be circular. SMV decides what is in zero-delay or unit-delay environment.

Verification by SMC

- Current/next state variables
  - $\sigma(t) \quad \rightarrow \quad \sigma$
  - $\sigma(t+1) \quad \rightarrow \quad \sigma'$

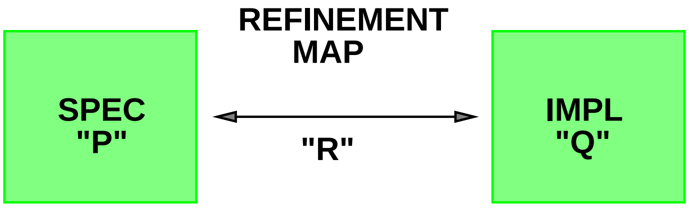
- Transition relation and state invariant
  - $\mathcal{E}_p^0 \quad \rightarrow \quad S$
  - $\mathcal{E}_p^1 \quad \rightarrow \quad T, I$

- Reachability analysis

$$
\begin{aligned}
R_0 &= I \wedge S \\
R_{i+1} &= R_i \vee \mathit{Img}(T, R_i) \wedge S
\end{aligned}
$$

- Property checking: $R_\omega \Rightarrow p$

Refinement maps



- Want to prove:

$$Q \Rightarrow \exists \sigma_P : \ (P \wedge R)$$

- Proof should be compositional
  - R should be a composition of “processes”
  - Use some components to prove others
  - Allow circular proofs

Refinement map approach

Theorem 2. *Given process sets P, Q, W such that*

- σ_W *and* σ_Q *disjoint*
- W *is "acyclic" and "single assignment"*
- Q *does not reference* σ_W

Then the following inference rule is sound:

$$\frac{Q \wedge W \Rightarrow P}{Q \Rightarrow \exists \sigma_W : \; P}$$

The process set W takes the role of a "witness function" for the existential quantifier.

SMV implementation

- Example: Specification:

```
abstract toggle : boolean;
state : boolean;

toggle := {0,1};
init(state) := 0;
if(toggle)next(state) := ~state;
```

- Example: Implementation:

```
init(count) := 0;
next(count) := count + 1 mod 16;
state := (count < 8);
```

- Example: Witness function:

```
toggle := (count = 7 | count = 15);
```

Example – Tomasulo's algorithm

Several points:

- Refinement maps occur naturally.
- Can reduce proof compositionally to a few easy model checking lemmas.
- Need to exploit symmetries.

Basic approach (Abadi and Lamport):

- History variables ("abstract vars")
- Refinement maps

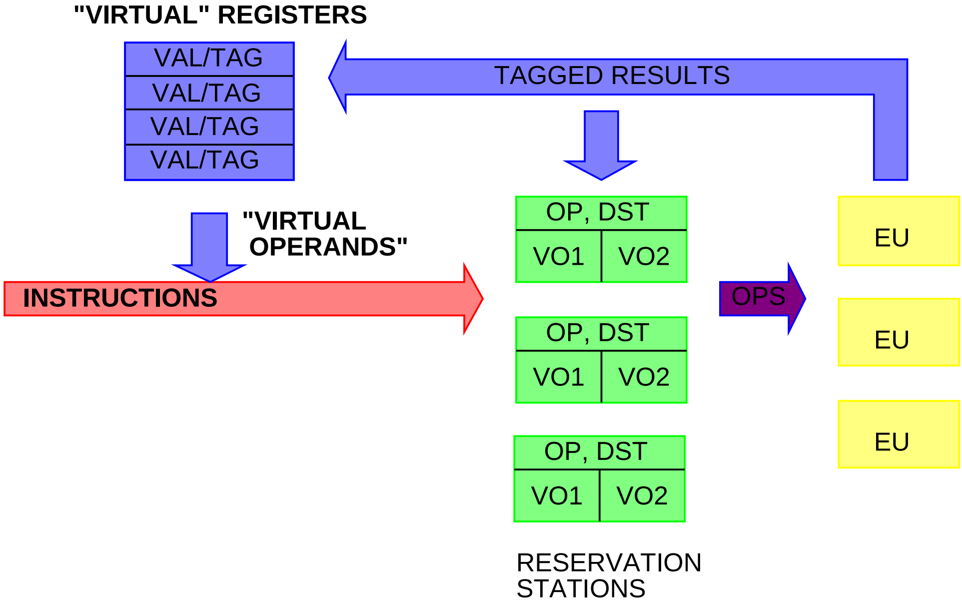
    Relate spec state, history, impl state.

- Circular proof.

The algorithm

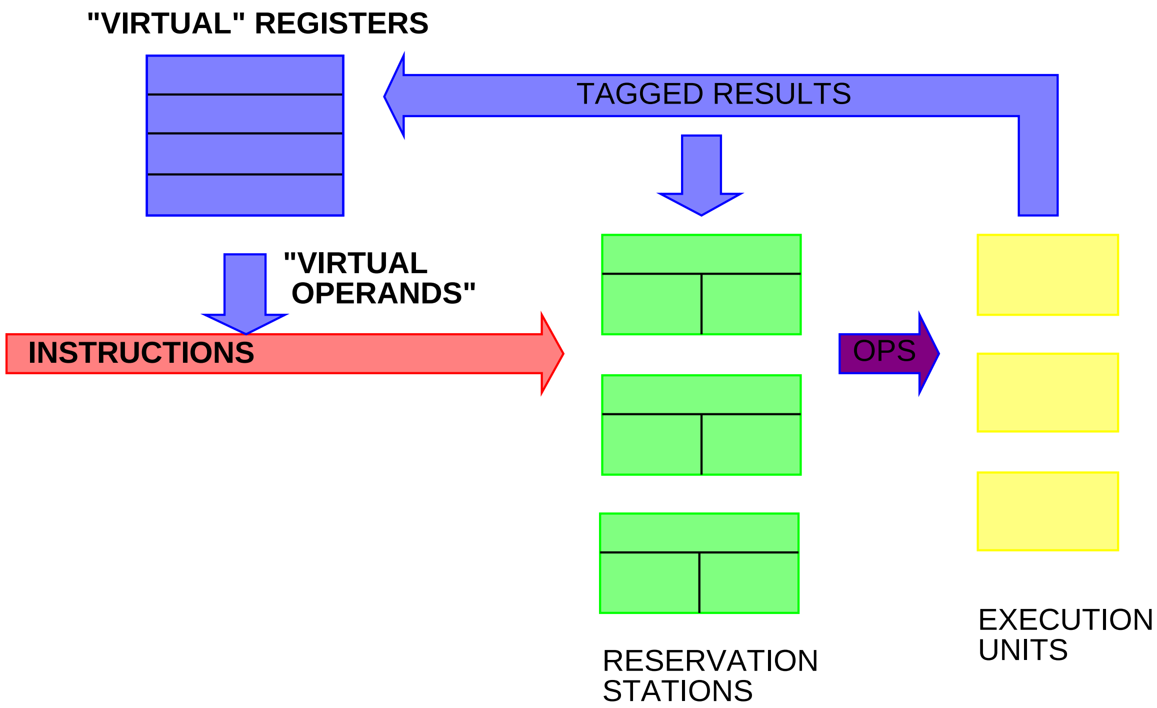
- Execute instructions in data flow order

- Overall architecture



Example

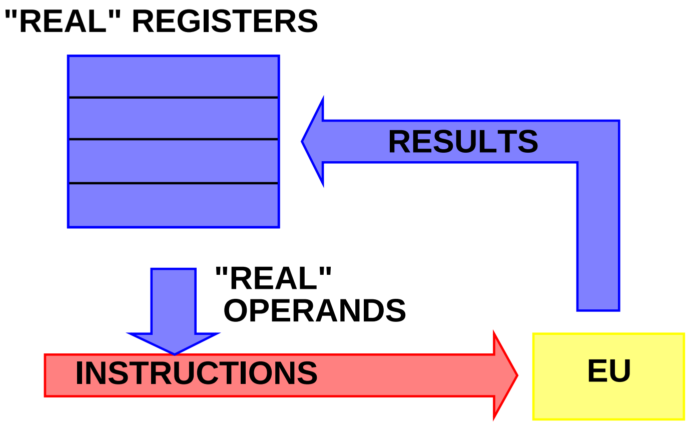
```
 0 - R0   -->  R0
R0 + R1   -->  R2
 0 - R1   -->  R3
```



Verification approach

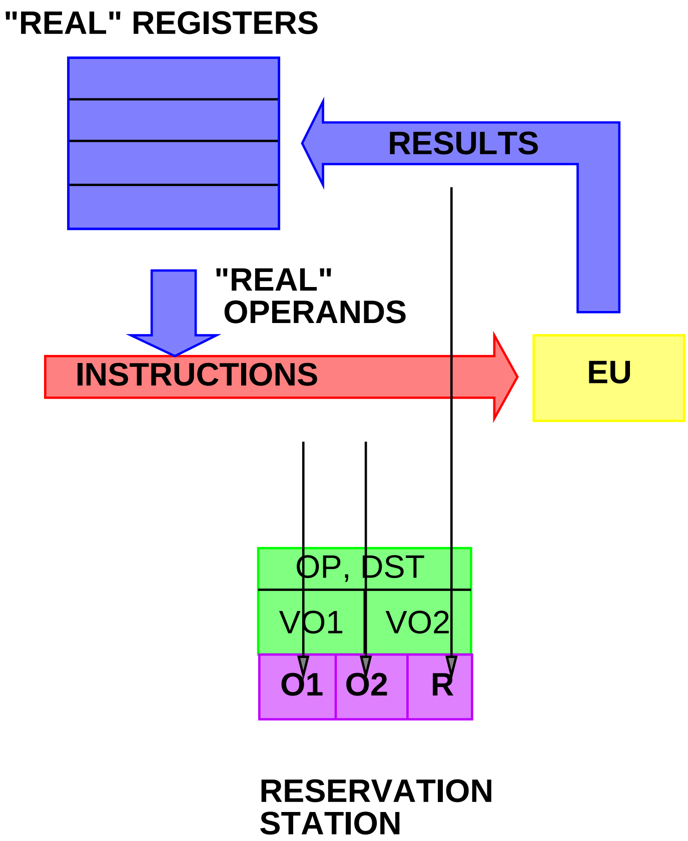
- Specification P
  - Sequential execution
  - Stalls nondeterministic
- History variables for each station
- Refinement maps relate
  - Specification state
  - History variables
  - Implementation state
- Compositional proof

Specification



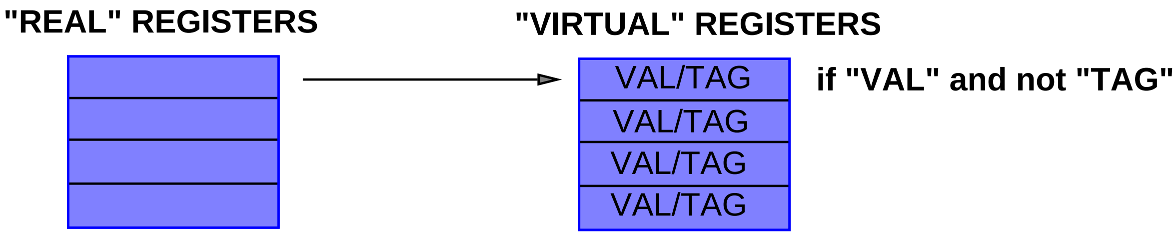

```
layer arch: {
    stallout := {0,1};
    if(~stallout)
      switch(opin){
        ADD_OP : {
          opra := r[srca];
          oprb := r[srcb];
          res := opra + oprb;
          next(r[dst]) := res;
        }
        RD_OP : {
          dout := r[srca];
        }
        ...
```

History variables



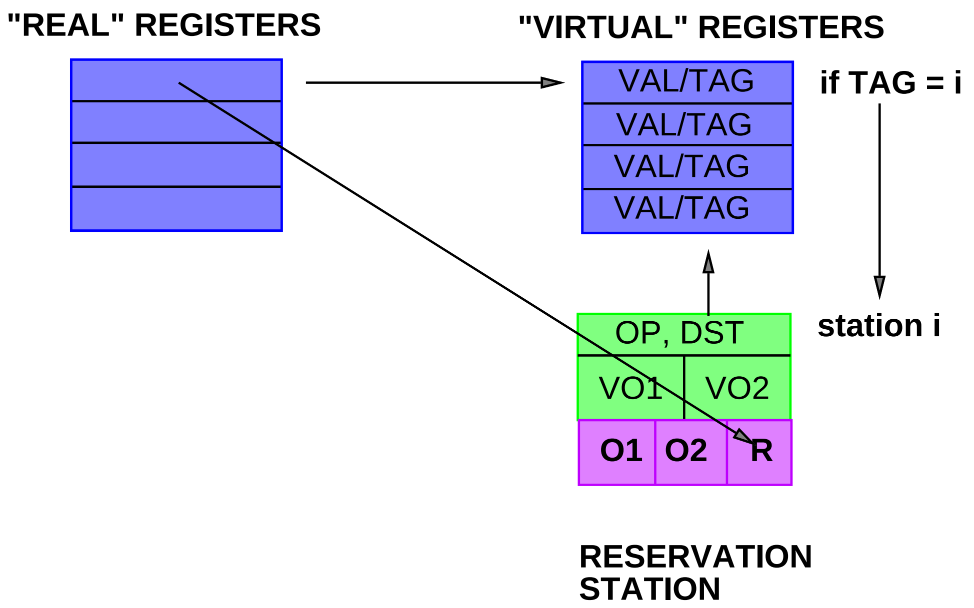

```
if(~stallout & IS_PIPE_OP){
  next(hist[st_choice].opra) := opra;
  next(hist[st_choice].oprb) := oprb;
  next(hist[st_choice].res) := res;
}
```

Refinement maps (1)



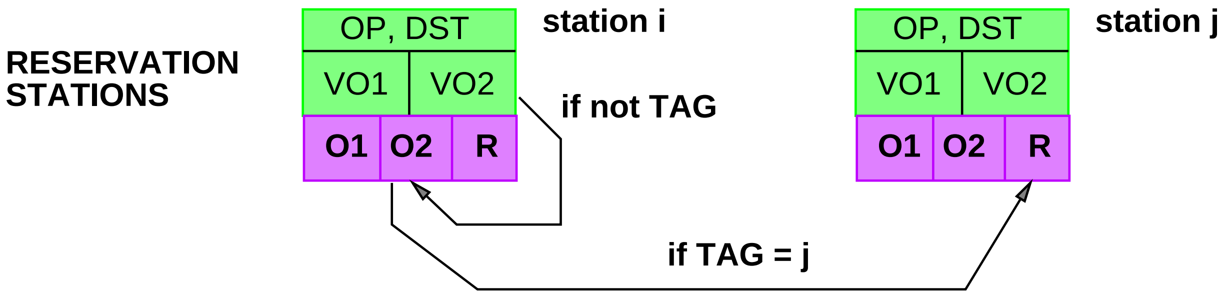

```
layer map1 : {
  forall(i in REG)
    if(~ir[i].resvd)ir[i].val := r[i];
}
```

Refinement maps (2)



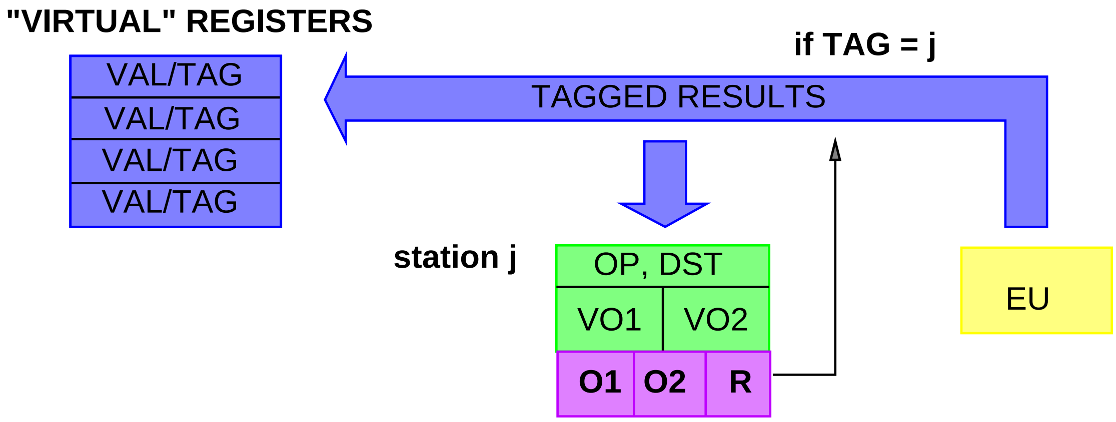

```
forall(j in REG){
  layer  map2[j] :
    forall(i in TAG){
      if(ir[j].resvd & ir[j].tag = i){
        st[i].valid := 1; st[i].dst := j;
        hist[i].res := r[j];
      } else {
        st[i].valid := {0,1};
        if(st[i].valid)st[i].dst :=
                    {k : k in REG, k ~= j};
      }}}
```

Refinement maps (3)



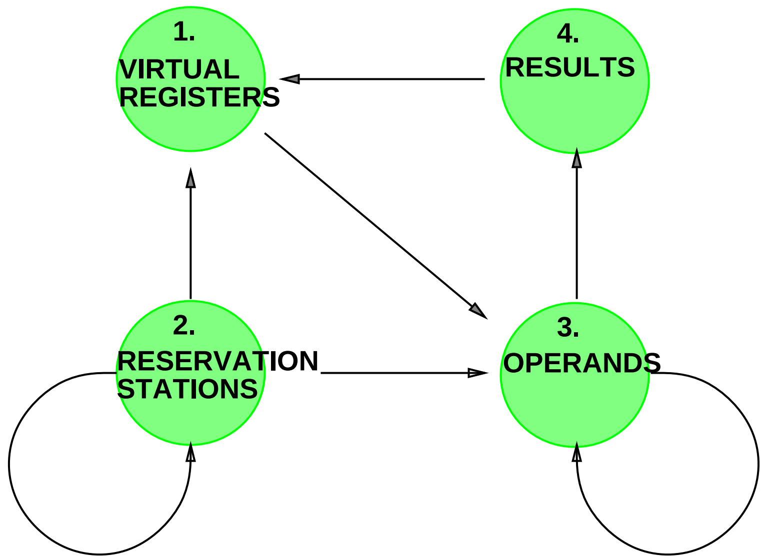

```
forall(i in TAG)
  layer map3a[i] : {
    if(st[i].valid & ~st[i].opra.valid)
       hist[st[i].opra.tag].res :=
                                    hist[i].opra;
    if(st[i].valid & st[i].opra.valid)
      st[i].opra.val := hist[i].opra;
  }
```

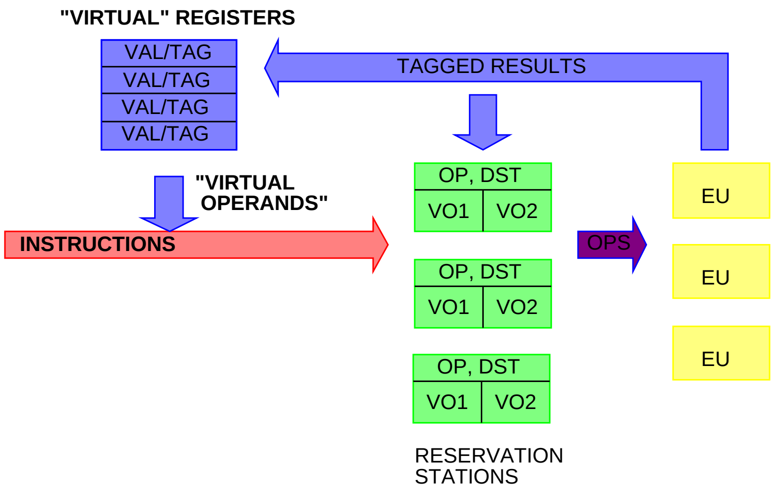
Refinement maps (4)



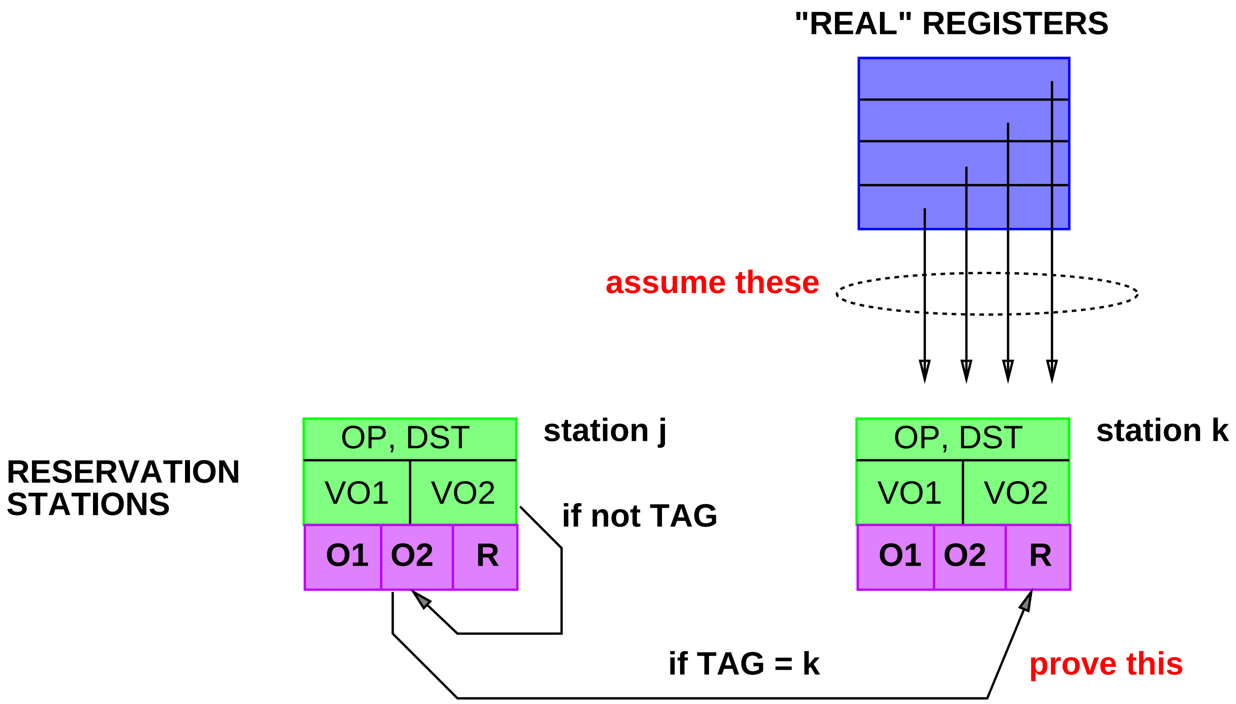

```
layer inv4 : {
  pout.valid :=
         st[pout.tag].valid ? {0,1} : 0;
  if(pout.valid)
    pout.val := hist[pout.tag].res;
}
```

Compositional proof





Example environment



Note: simultaneously use many assignments to the same signal.

Verification results

- 11 "lemmas" verified in 376 secs by SMV
  - 8 registers
  - 8 reservation stations
  - 32 bit data paths
  - Actual arithmetic not verified
- Bypass bug found, verifying map3

Why is it tractable?

- State variable reduction

  At most 1 reservation station and 1 register is "state" in any verification.

- Bit slicing
- Symmetry reduction

Symmetry reduction

- Problem: $O(n^3)$ properties (tags$\times$tags$\times$bits).

```
forall(i in TAG)
  layer map3a[i] :
    if(st[i].valid & ~st[i].opra.valid)
       hist[st[i].opra.tag].res
                          := hist[i].opra;
```

- Solution: symmetry reduction
  - Check only one property per equiv class
  - Not state space reduction

Scalar sets (Murphi)

- Scalar set:
  - Finite type
  - Semantics invariant under permutations

- Declaration

```
scalarset TAG 0..31;

x, y : TAG;
```

Scalarset usage rules

- Equality comparison

```
if(x = y) ...
```

- Array indices

```
z : array TAG of boolean;

next(z[x]) := ~z[x];
```

- Forall construct:

```
forall(i in TAG)
  z[i] := ...
```

- Reduction / comprehension expressions

```
&[ z[i] : i in TAG]

{ i : i in TAG, z[i]}
```

Symmetry reduction

- Two parameterized properties are equivalent when one is obtained from the other by permuting scalarset types.

- Example: arbiter

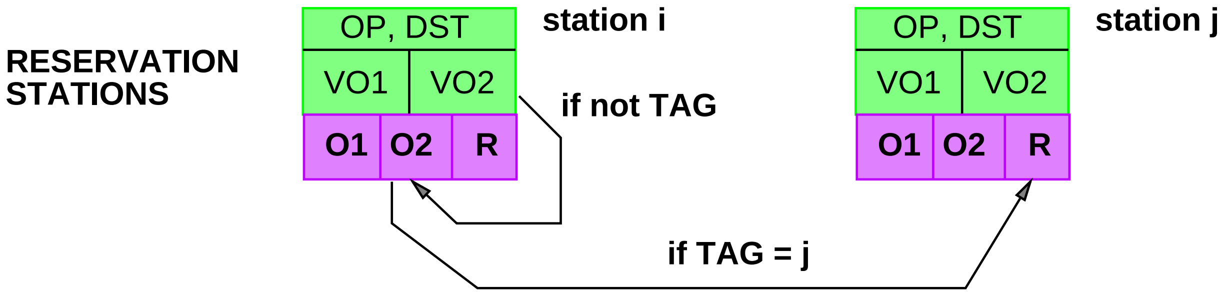
```
scalarset USER 0..31;

forall(i in USER)
  forall(j in USER)
    mutex[i][j] : assert
      G (i ~= j) -> ~(ack[i] & ack[j]);
```

  Here, two representatives are sufficient:

```
    mutex[0][0]
    mutex[0][1]
```

  covering the two cases $i = j$ and $i \neq j$.

Tomasulo example




```
forall(i in TAG)
  layer map3a[i] :
    if(st[i].valid & ~st[i].opra.valid)
      hist[st[i].opra.tag].res := ...
```

- We have 2K proof obligations of form:

```
hist[k].res[i]//map3a[j]
```

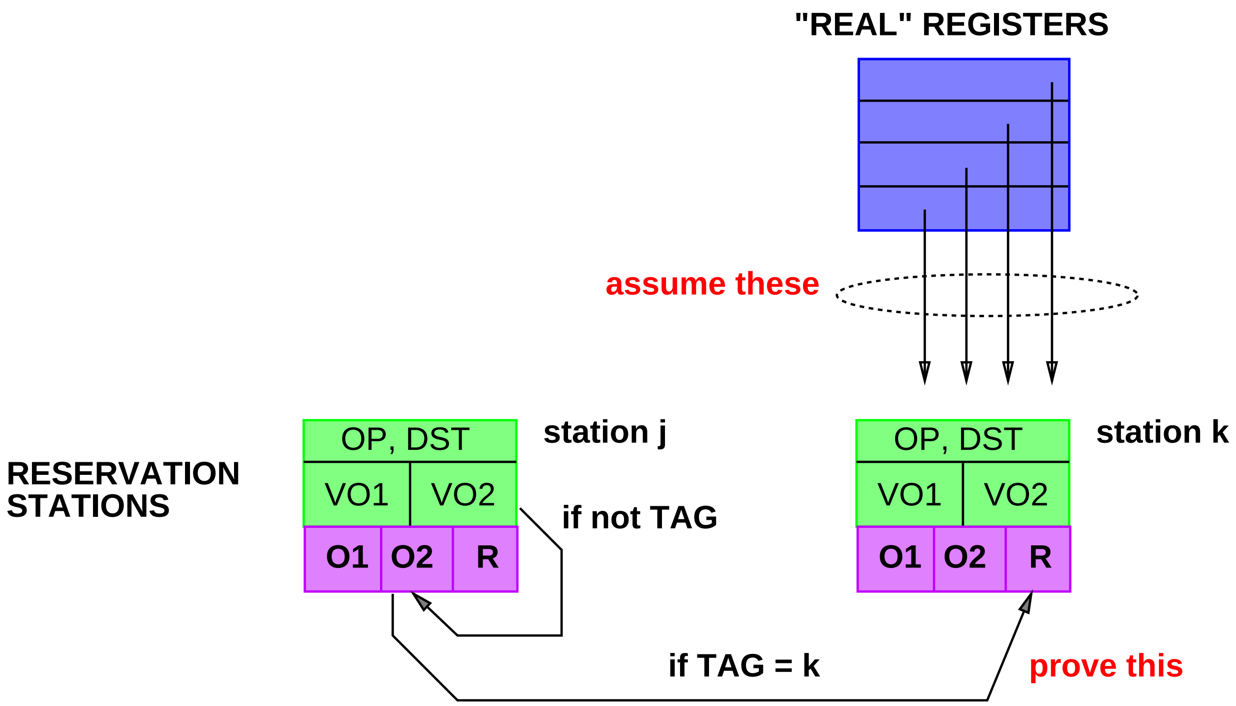
where `i` is BIT, `j`, `k` are TAG's.

- SMV chooses two representatives:

```
hist[0].res[0]//map3a[0]
hist[1].res[0]//map3a[0]
```

Verified for all sizes?

No. We used finite expansion:



Two possibilites:

- Inductive argument
- Just reverify if size changes

Remaining issues

- Symmetry breaking
  - Arithmetic units break bit symmetry
  - Can't use when using symmetry reductions
  - In general, low level logic breaks abstract symmetry
- Exceptions/snapshotting
- Floating point arithmetic

Conclusions

- Refinement maps occur naturally in the state encodings of designs.
- Can use these to break big proof into small lemmas for model checking
- Very simple compositional system
  - Require circular arguments, not just hierarchies!
  - Exploit symmetry to avoid “property explosion”.