Verification of Infinite State Systems by Compositional Model Checking

Ken McMillan

Cadence Berkeley Labs

Is hardware really “finite state”?

- Unbounded/infinite structures may be appropriate...
  - For model checking, sometimes $64 = \infty$.

    64 bit words, large address spaces, *etc*.
  - Configurations not known *a priori*
  - Hardware processes (conceptually) infinite sequences

    e.g., a packet router

Thus, for hardware verification, it is useful to be able to treat data structures of unknown or infinite size

Methodology

Reduce unbounded types to fixed finite types by abstraction

- Split property into case based on parameter
  - e.g., memory output correct when addr = 0
- Reduce data types
  - map type "address" to 0 and "not 0"
  - abstract Mem[not 0] to "unknown"
  - problem becomes finite state -- apply model checking
- Apply symmetry
  - case "addr = 0" implies all other cases

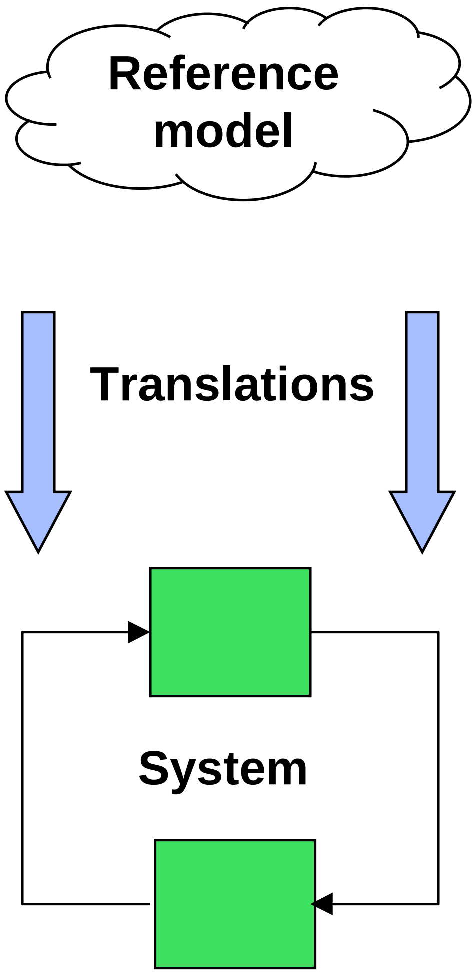
Uninterpreted functions

This methodology also allows us to use uninterpreted functions in model checking.

- Advantages of uninterpreted functions
  - Generic verification
  - Separate “plumbing” from arithmetic
  - Preserve symmetry

Many functions can be abstracted this way, for example, the mapping function of a cache, or the routing table of a packet router.

Refinement framework

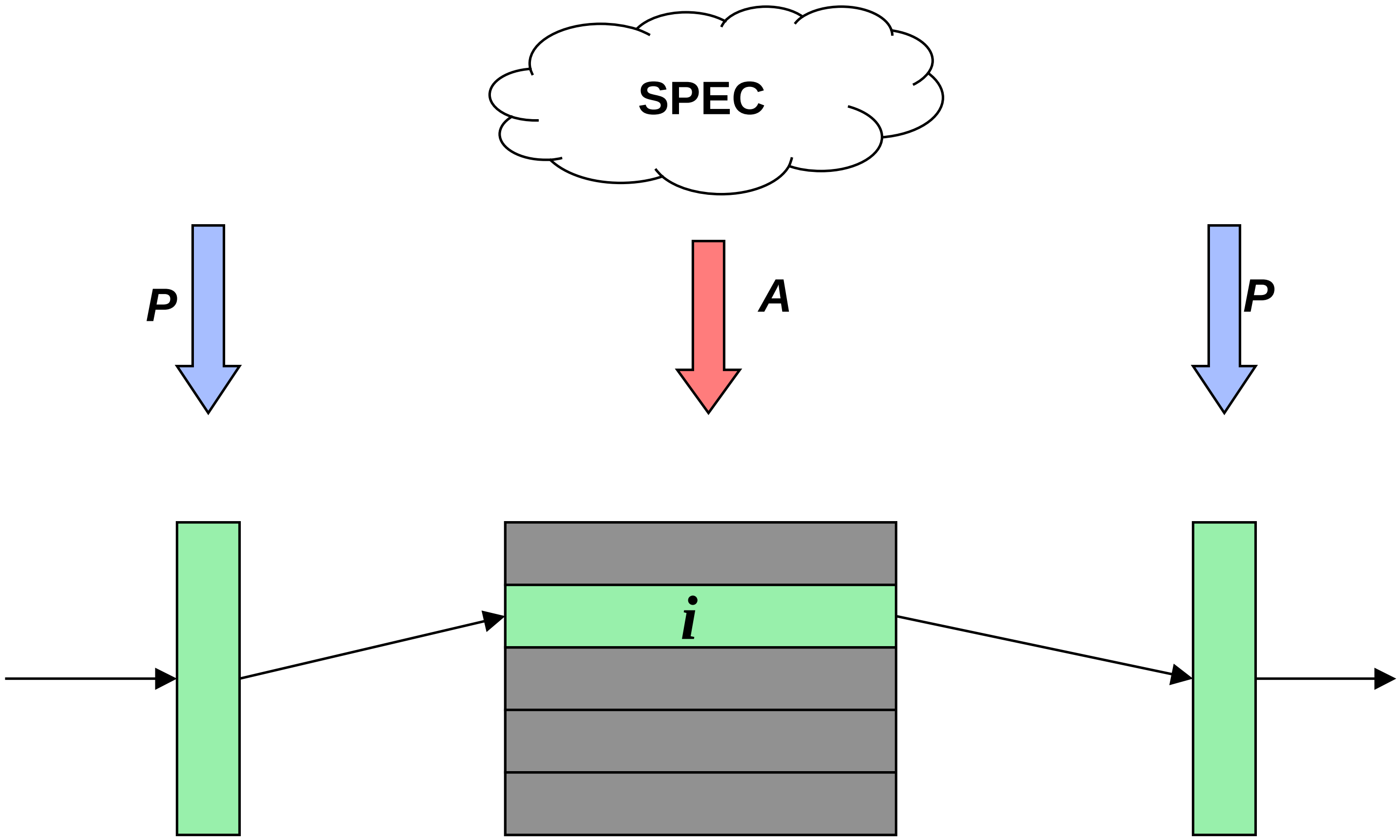


- Refinement relations
  - Specify intermediate results with respect to reference model
  - Each intermediate result uses finite

    operations

    storage locations
  - Thus, can reduce local verification problems to finite state

Decomposing arrays

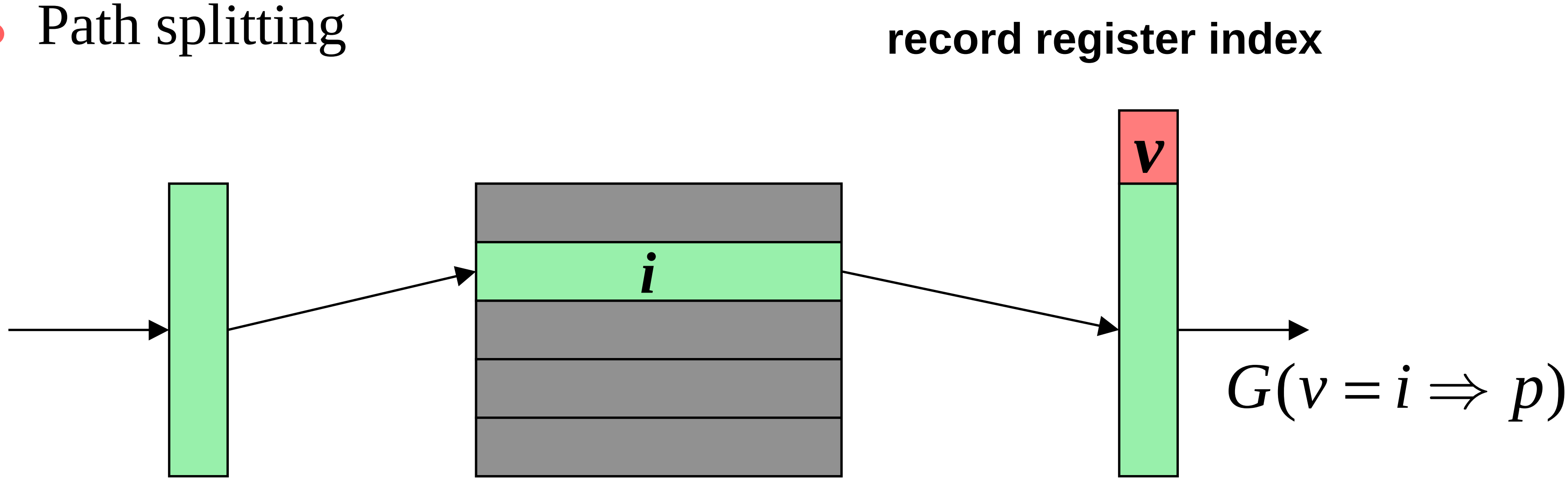


Temporal case splitting

- Prove separately that p holds at all times when $v = i$.

$$\frac{\forall i:\ G(v = i \Rightarrow p)}{G\ p}\,*$$

- Path splitting



Implementation in SMV system

v : *T*;
s : **assert** *G p* **;**

forall (*i* **in** *T***)**
    subcase *c***[***i***] of** *s* **for** *v* **=** *i***;**

*c***[***0***] : assert** *G* **(***v***=***0* $\Rightarrow$ *p***) ;**
*c***[***1***] : assert** *G* **(***v***=***1* $\Rightarrow$ *p***) ;**

...

In general, we can split cases using any set of assertions q_i, where SMV can determine that $\vee_i \, q_i = 1$.

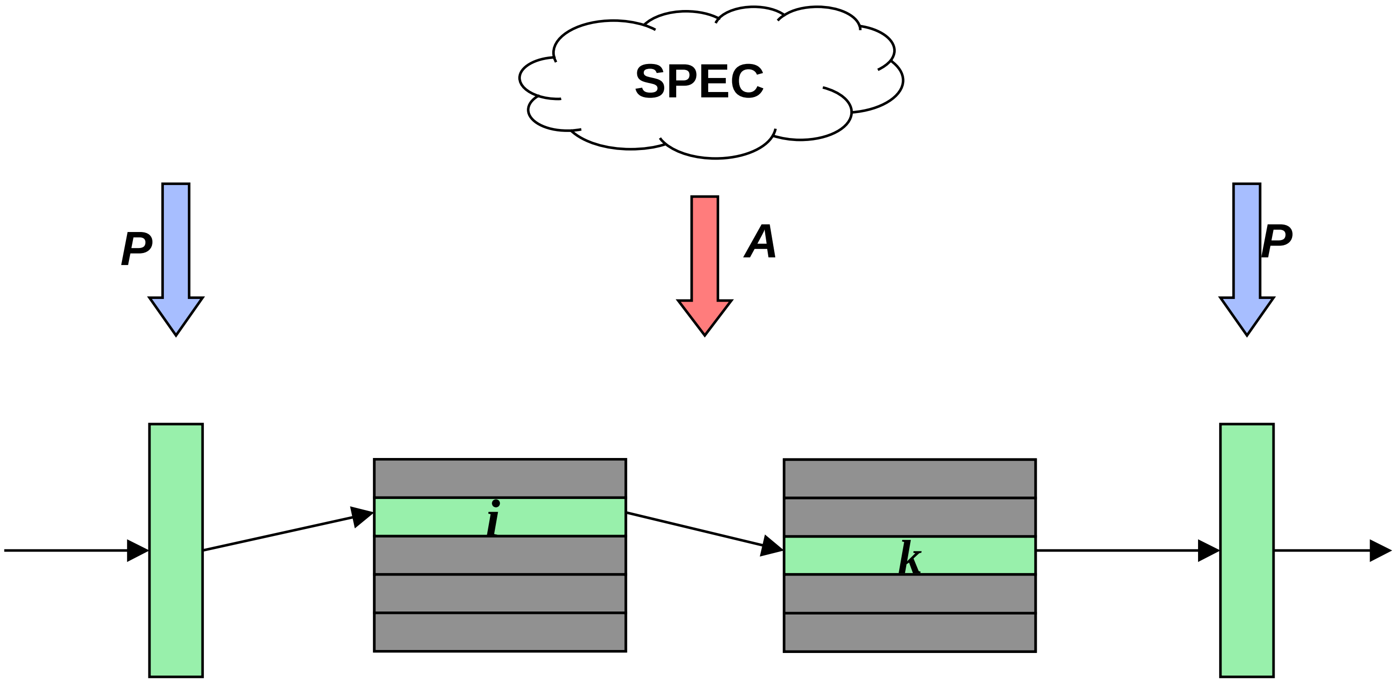
Example:

forall (*i* **in** *T***)**
    subcase *c***[***i***] of** *s* **for** *v* **=** *i* **when** *r***;**

i.e., can split cases on future *value of a variable*

Case explosion

Number of cases may explode exponentially...



... may even be unbounded.

Exploiting symmetry

- Symmetric types **(related: Murphi)**
  - Semantics invariant under permutations of type.
  - Enforced by type checking rules.
- Symmetry reduction rule

$$\frac{\forall i = 0 \ldots k : P}{\forall i : P} *$$

* i is of scalarset type
* *P* references only constants 0..*k*-1 of type

SMV can select a finite set of representative cases under symmetry.

Data type reductions

- Problem: unbounded or infinite data types
- Solution: reduce to finite types by abstraction

$$T \rightarrow \{i, T \setminus i\}$$

where $T \setminus i$ represents all the values in T except i.

- Need a property-preserving abstract interpretation

$$\frac{M_{T \rightarrow \{i, T \setminus i\}} \models P}{M \models P}$$

Abstraction must be both sound and generically applicable.

Data type reduction in SMV

- Values in reduced model *M'* are sets

$$T' = \{\{i\}, T \setminus i\}$$

- For sound abstraction of operator *f*, we need:

$$f'\{\ldots x \ldots\} = \{\ldots f(x) \ldots\}$$

- Examples:
  - Equality

$=$	i	$T \setminus i$
i	1	0
$T \setminus i$	0	{0,1}

Abstraction, continued...

– Array references

a[x] ...becomes... $(x=i)\ ?\ a[i] : \perp$

Unbounded array reduced to one fixed element!

– Other operators ...

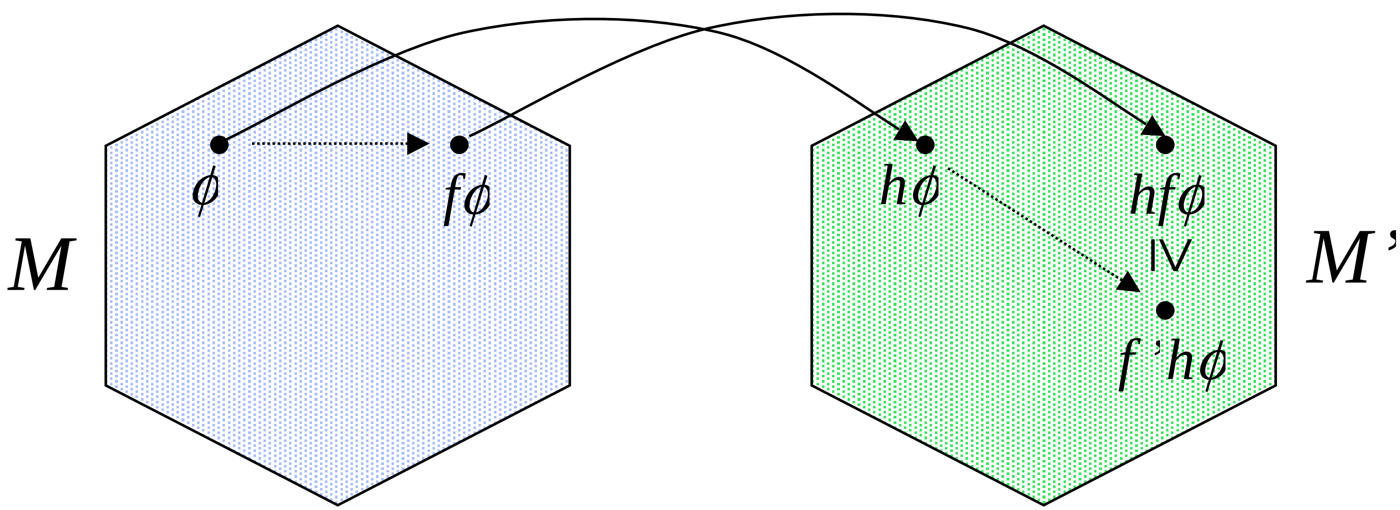
$$\{0\} \wedge' \perp = \{0\} \qquad \{1\} \wedge' \perp = \perp$$

$$\{1\} +' \perp = \perp \qquad \textit{etc...}$$

Must also account for quantifiers, temporal operators, etc...

Property preserving abstractions

- Homomorphism from M to M'

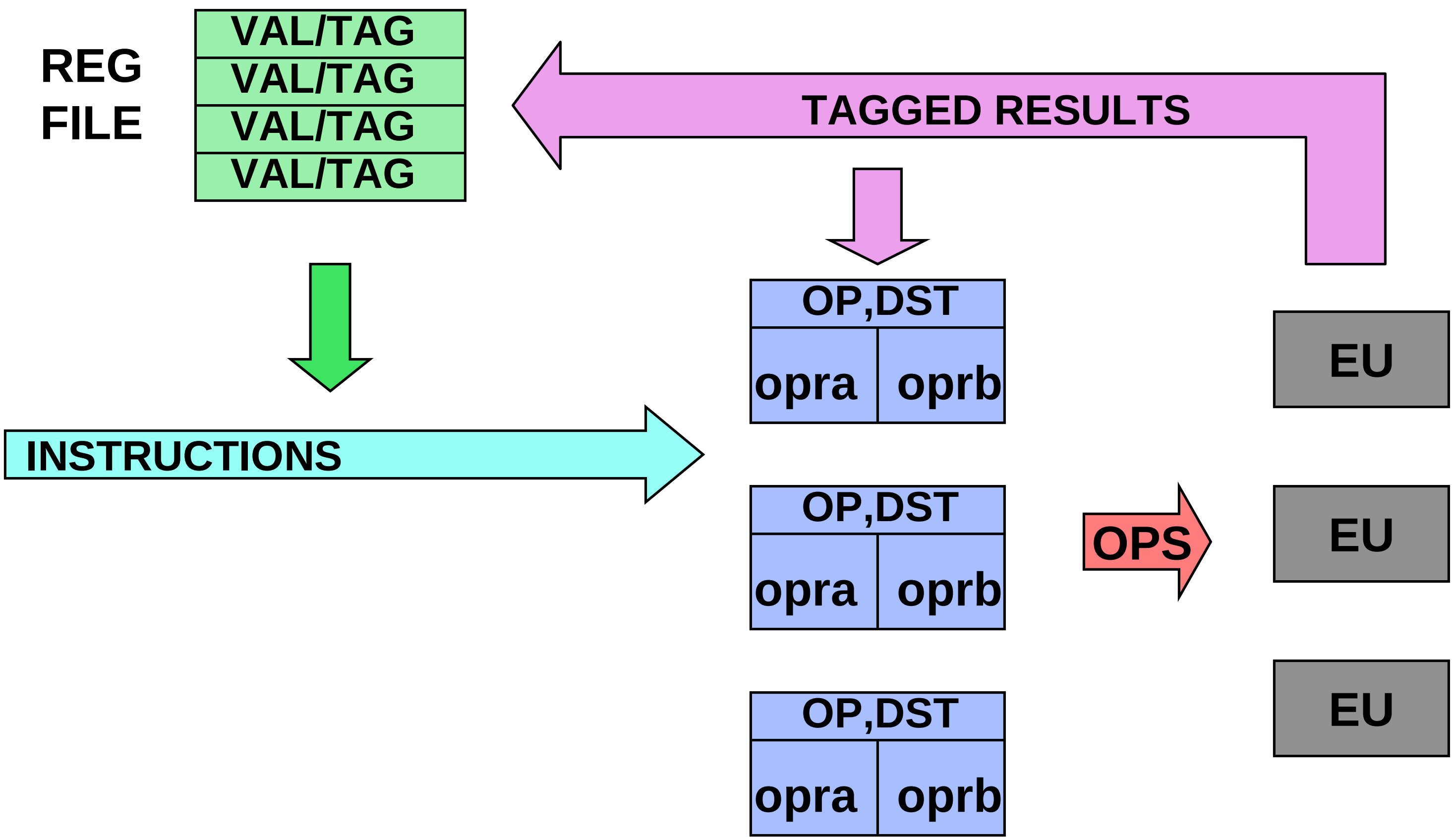


- Thus, if $\phi^{M'} = \{1\}$ then $\phi^{M} = 1$

...but we may get $\phi^{M'} = \perp$

Illustration: Tomasulo's algorithm

- Execute instructions in data flow order



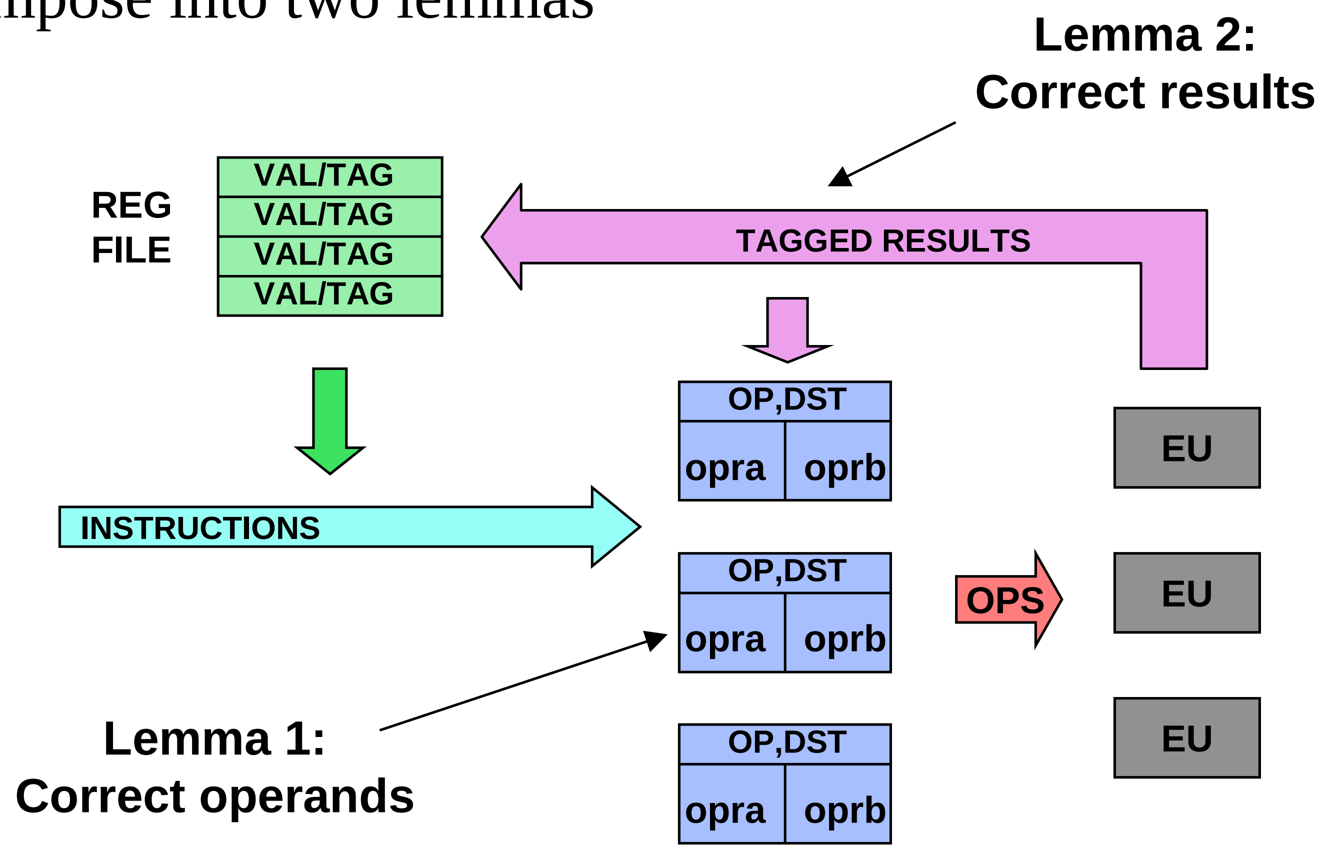
Unbounded resources

- The following types are left undefined:
  - REG (register file indices)
  - TAG (reservation station indices)
  - EU (execution unit indices)
  - WORD (data words)

This allows for an unbounded (or infinite) number of registers, reservation stations, execution units, and bits in the data word.

Compositional rule

- Decompose into two lemmas



Computing each result and operand uses finite resources

Lemmas in SMV

- Operand correctness

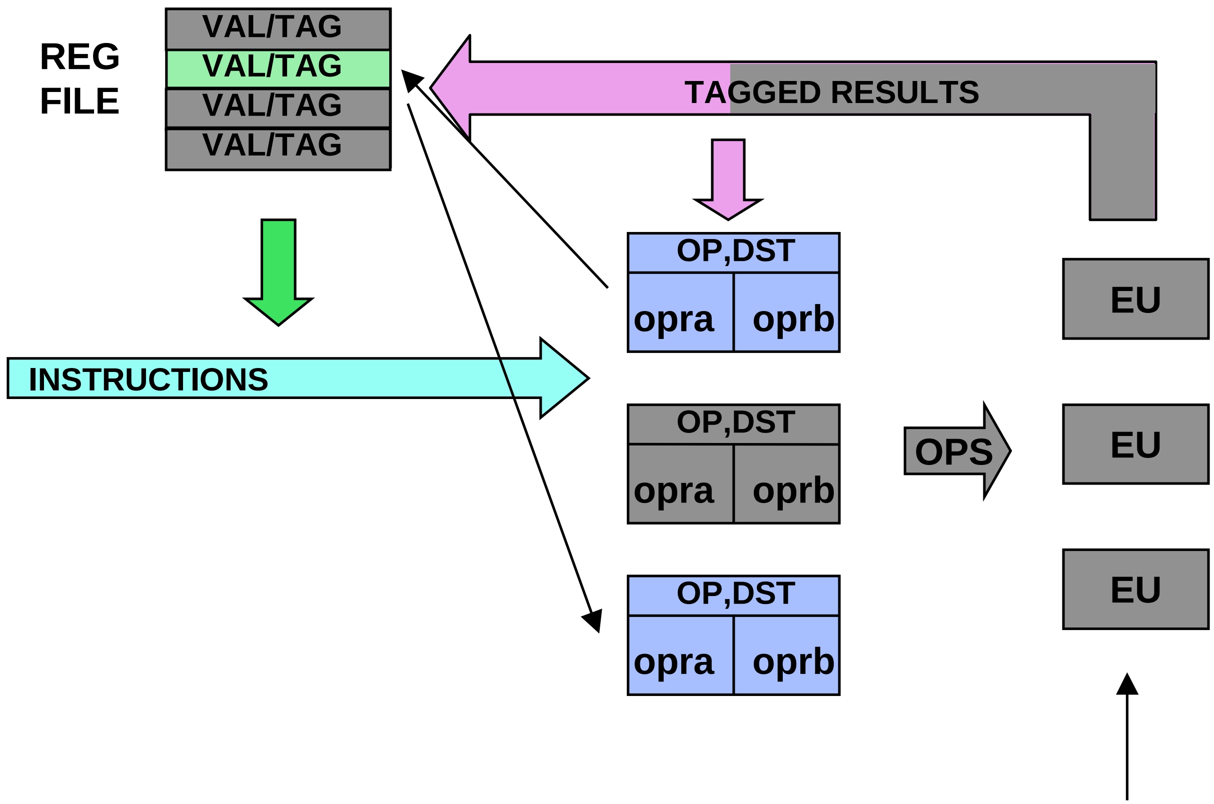
```
forall (k in TAG)
  layer lemma1 :
    if (rs[k].valid & rs[k].opra.valid)
      rs[k].opra.val := aux[k].opra;
```

- Result correctness

```
forall (i in TAG)
  layer lemma2[i] :
    if (rb.tag = i & rb.valid)
      rb.val := aux[i].res;
```

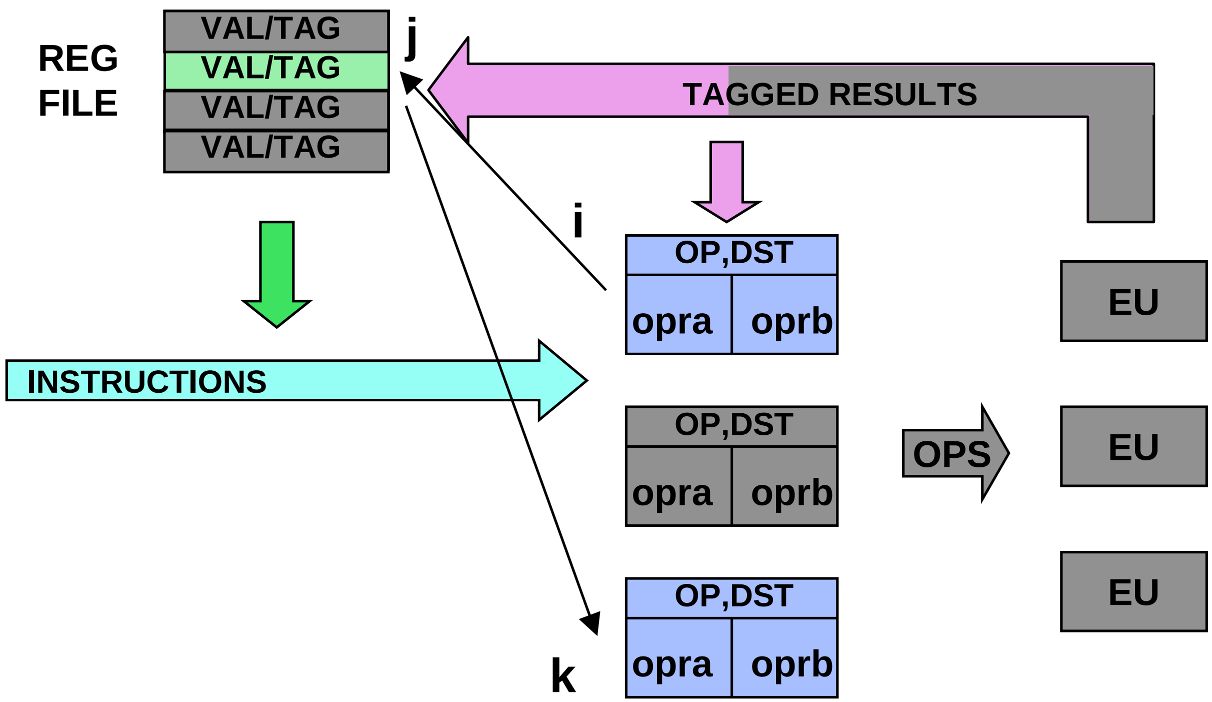
Note: only two signals specified in proof

Path splitting in Tomasulo



"cone of influence" eliminates

Symmetry reduction in operands lemma



- Symmetry reduces unbounded cases to two cases:

$$(i = 0,\ \ j = 0,\ \ k = 0)$$

$$(i = 0,\ \ j = 0,\ \ k = 1)$$

Type reduction

- Reduce types of undefined range to finite types
- Specific reduction is chosen based on property parameters

  TAG $\rightarrow$ {i,k,~{i,k}}

  REG $\rightarrow$ {j,~{j}}

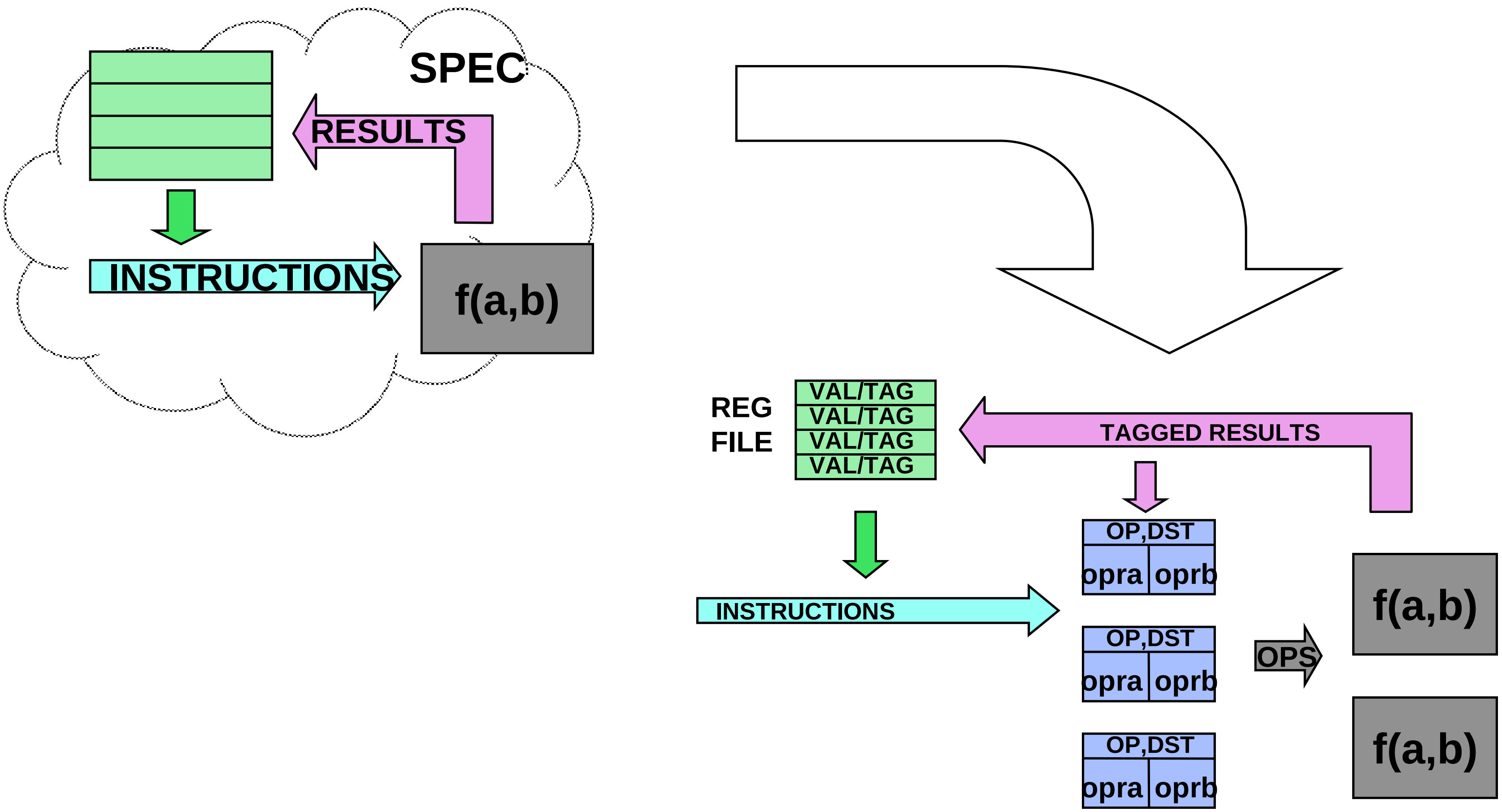
  i.e., only include values we care about in reduced types

- Reduces variable encodings to
  - 1 bit per variable of type REG
  - 2 bits per variable of type TAG

What about data words?

Uninterpreted functions

- Verify Tomasulo for arbitrary EU function f(a,b).



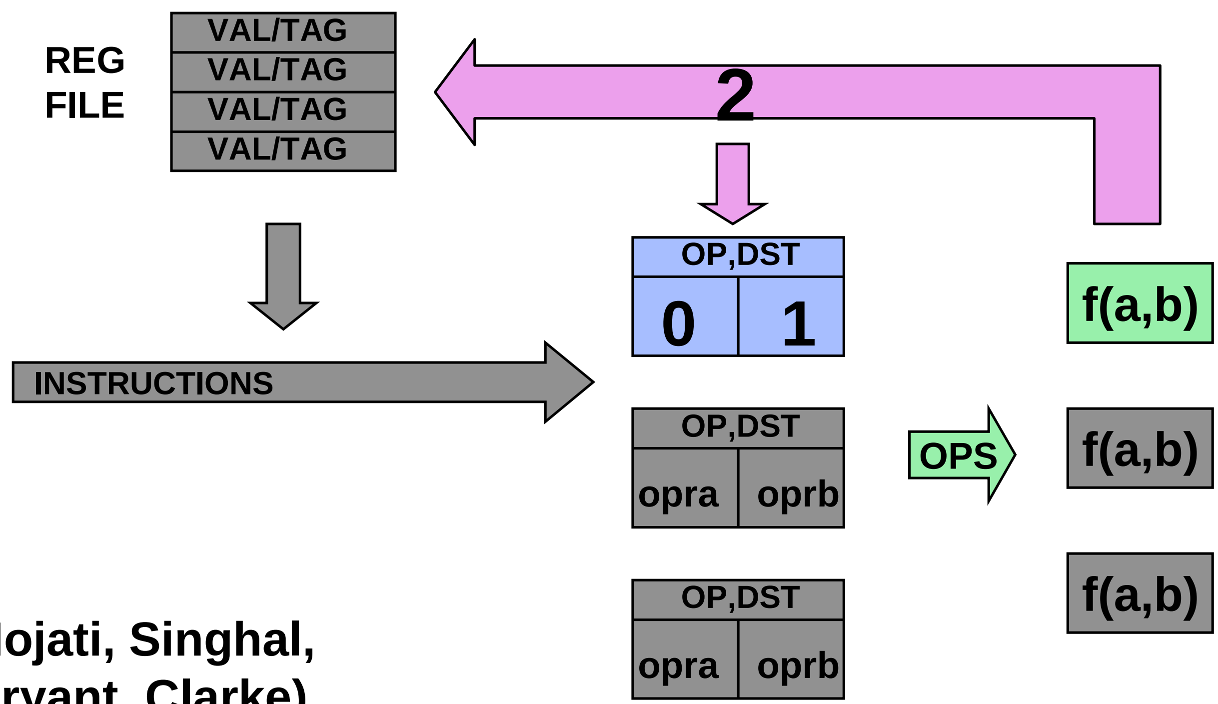
(related: Burch, Dill, Jones, etc...)

Case splitting

- Prove result correctness only for specific cases, e.g.

  opra = 0, oprb = 1, f[0][1] = 2

3! = 6 cases verified



(related: Hojati, Singhal, Bryant, Clarke)

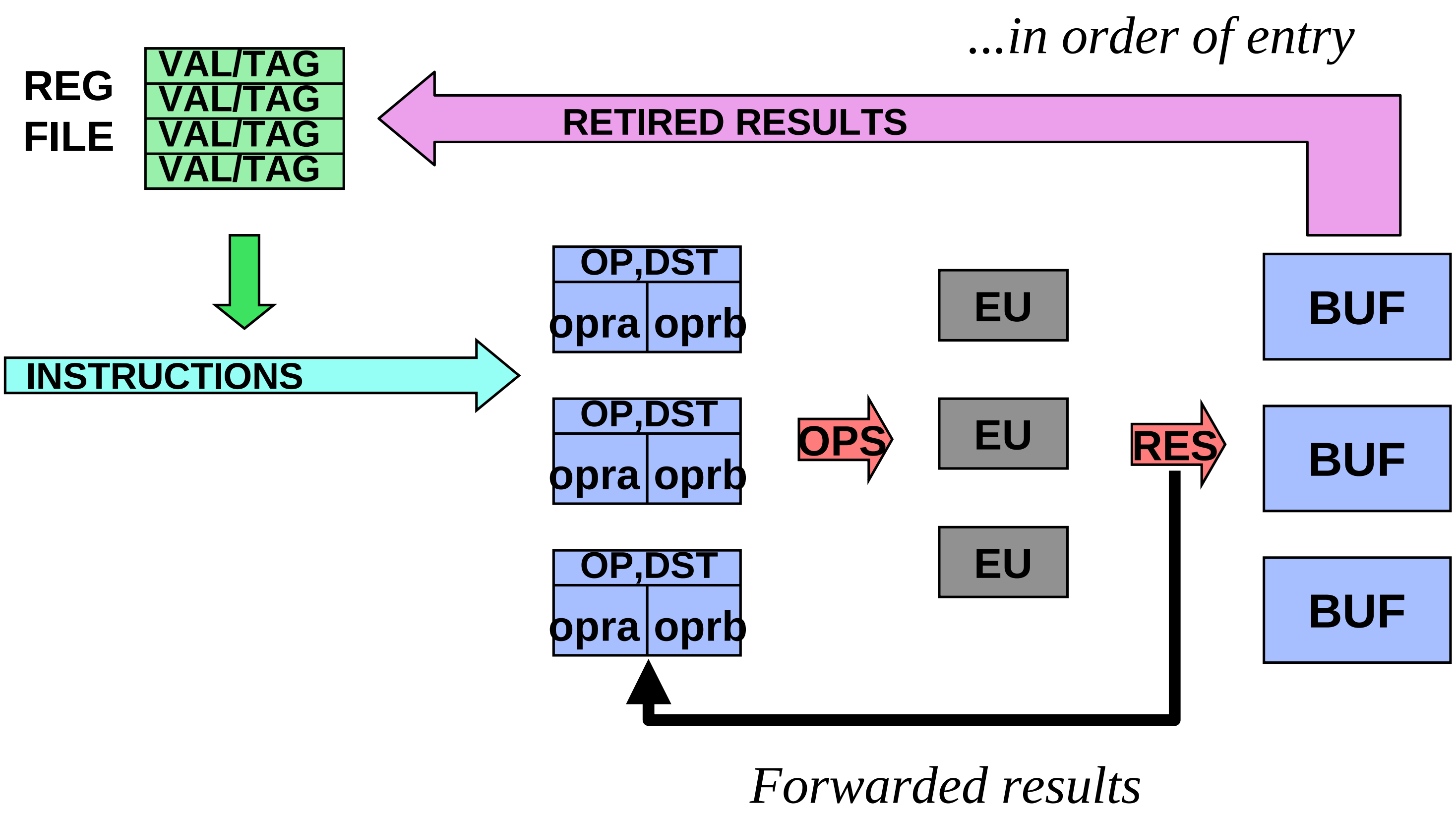
Result

- Verification problem reduced to tractable finite problems
  - Max 25 state bits
  - 11 cases of lemmas to verify after symmetry
  - Verification time less than 4 seconds
- Tomasulo implementation proved for
  - Arbitrary number of registers, reservation stations
  - Arbitrary data word size and EU function

    note -- *unbounded EU's requires one more lemma*
- The proof is concise and robust

    details in the paper...but in what way robust?

Adding in-order retirement



Reusing the proof

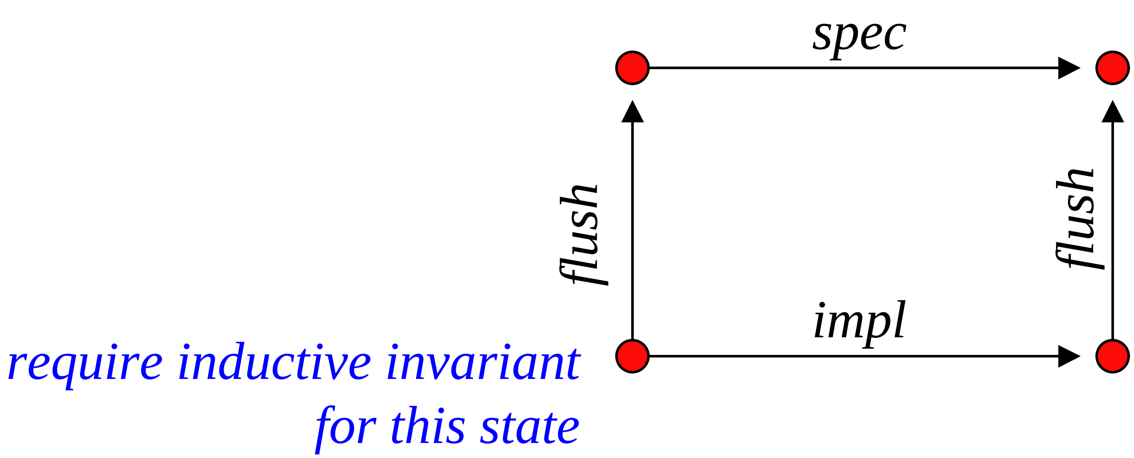
New version is verified without modifying one proof line...

- Why?
  - Refinement relations only specify two signals:

    operand registers and result bus
  - Function of these signals did not change

Because a model checker computes the reachable states,
we do not have to specify invariants of most signals.
This improves reusability of the proof.

Comparison to symbolic simulation



- Use uninterpreted functions to abstract out arithmetic
- Generating flushing function *can* be automatic
  - in practice, manual decompositions are required
- Inductive invariants can be complex
  - reduces reusability of proof

Other related work

- Data independence (Wolper)
  - require control to be independent of data
  - does not reduce the size of arrays
- Lazic and Roscoe
  - complete procedure (for determinism)
  - limitations to avoid undecidability:

    no comparison of array elements (i.e., no cache tags)

    no array indices stored in arrays (i.e., no Tomasulo)
  - present method is (necessarily) incomplete

Other applications

- Data type reductions
  - One-address abstraction in cache coherence
  - Content-addressable memories
  - Infinite packet streams
  - Circular buffers

using ordered types, inductive proofs

- Uninterpreted functions
  - Routing tables, address decoding
  - Hash functions (e.g., in caches)
  - Replace any symmetry-breaking function

Conclusion

- Hardware verification can involve large or unbounded structures
  - problem for finite state methods
- Case splitting and data type reduction:
  - combine with compositional methods
  - reduce to *small* finite scalars and arrays

    fewer manual decompositions
  - allows MC with uninterpreted functions

    avoid writing inductive invariants
  - concise, robust proofs with high degree of automation