Some strategies for proving theorems using a model checker

Ken McMillan

Cadence Berkeley Labs

Introduction

- Model checking provides a convenient, automated proof technique for finite-state systems, *however*...
  - state explosion problem $\rightarrow$ limited scale
  - cannot handle parameterized/unbounded systems
- Theorem provers are very general, *however*...
  - require detailed guidance
  - require an inductive invariant (can be quite complex!)
  - proof scripts are fragile

In principle, we would like to combine the two approaches to let a model checker handle the "details" of a proof

Application areas

- Would like to reason about
  - processor micro-architectures
  - cache coherence protocols
  - network systems, etc...
- Two important characteristics
  - very large, complex
  - unbounded state space

thus, cannot apply model checking directly...

Hybrid proof approaches

- Combine theorem prover and model checker
  - proof techniques that reduce problem quickly to finite state
  - implementation in a special purpose proof assistant

- Gains
  - less manual effort guiding prover
  - fewer manually constructed invariants
  - more robust proofs (will see what this means later)

will consider proof principles in abstract, and not implementation

Context

- For illustration, we will use F.O. temporal logic

  (for both specifications and implementations)

- We are given:
  - A universe of values *U*,
  - Inflexible variables i, j,... and function symbols f, g...
  - Flexible variables x, y,... and function symbols a,b,...
  - Boolean operators and F.O. quantifiers, ∀i,∃i,...
  - Temporal operators *F*, *G*, *X*, *U*.

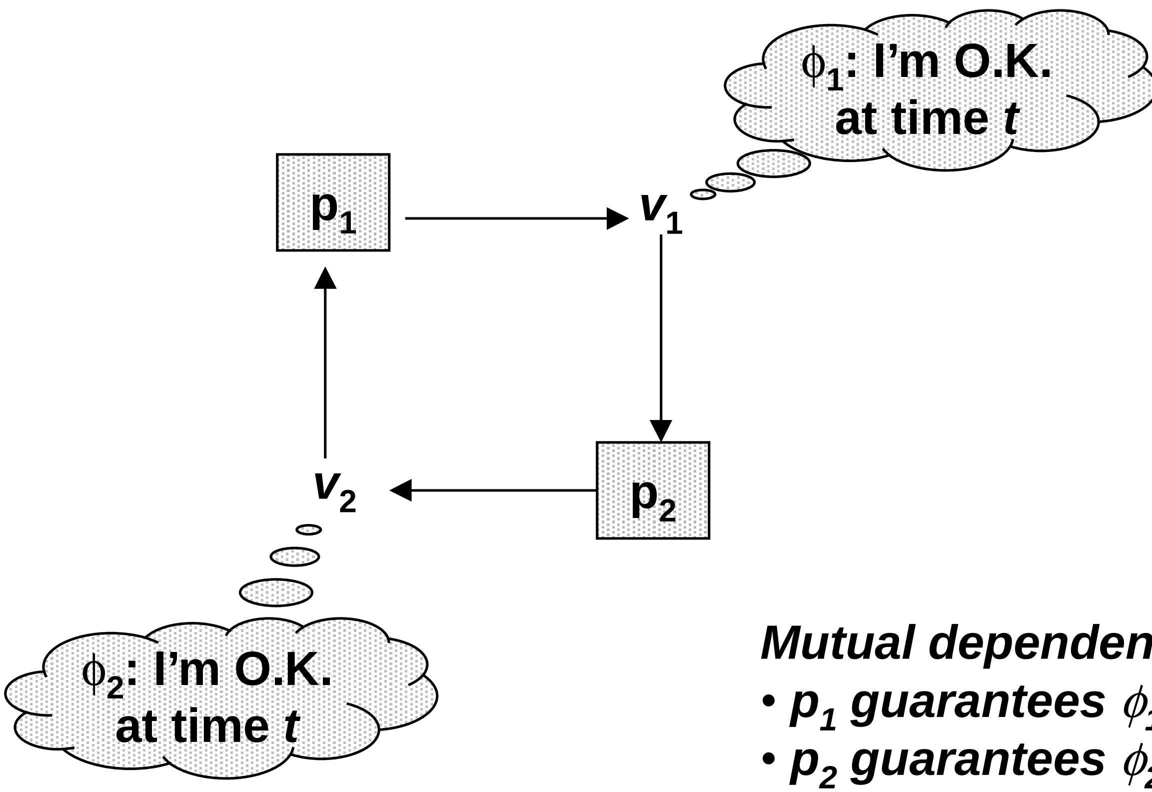
If U is finite, we can check validity in PSPACE (model checking)
but, in general, U can be infinite and logic is undecidable.

Problems in reducing to finite state

- Problem too large, complex
  - apply circular compositional reasoning to decompose proof
- Variables of unbounded range (i.e., U is infinite)
  - parameterize the problem
  - reduce to finite parameter instances (e.g., by symmetry)
  - abstract U to finite set for each instance

we will consider this approach for two particular theories:
• uninterpreted functions with equality
• successor and inequality
...and consider applications...

Circular compositional reasoning



Mutual dependence:

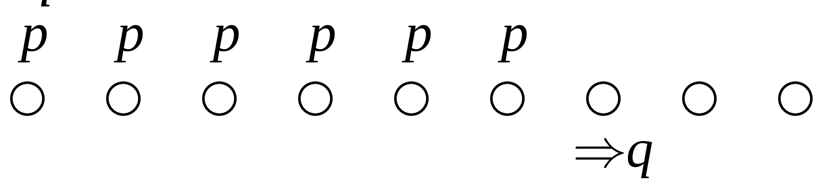
- ***p_1 guarantees ϕ_1 as long as ϕ_2 holds***
- ***p_2 guarantees ϕ_2 as long as ϕ_1 holds***

Circular compositional reasoning (cont.)

- Abbreviation: "if p up to t-1, then q at t"

$$p \mapsto q \quad \equiv \quad \neg(p U \neg q)$$

$p \mapsto q$



- "Circular" proof:

$$\frac{\begin{array}{c} p_1 \Rightarrow (\phi_2 \mapsto \phi_1) \\ p_2 \Rightarrow (\phi_1 \mapsto \phi_2) \end{array}}{(p_1 \wedge p_2) \Rightarrow G(\phi_1 \wedge \phi_2)}$$

- *Hypotheses can be model checked*
- *Checks are local*
- *Differs from std. invariant checking:*
  - *"course of values" induction*
  - *not state properties*

General rule

- **Given:**

$$\Delta_i \subseteq \{\phi_1, \phi_2, \ldots\}$$ **(assumed up to t-1 to prove ϕ_i at t)**

$$\Theta_i \subseteq \{\phi_1, \phi_2, \ldots\}$$ **(assumed at t to prove ϕ_i at t)**

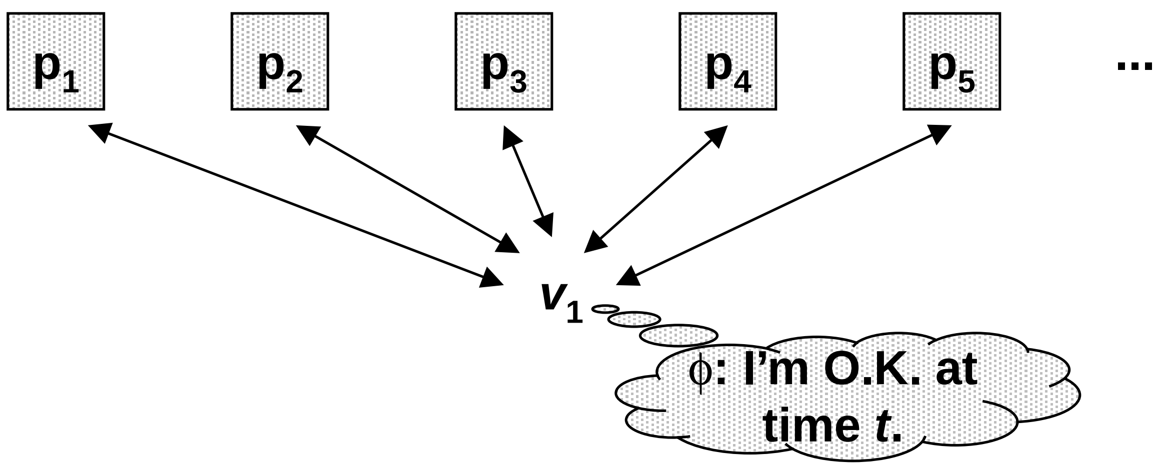
$$\Gamma_i \subseteq \{p_1, p_2, \ldots\}$$ **(assumed to prove ϕ_i)**

where $\phi_j \in \Theta_i$ **implies** $j < i$

$$\frac{\forall i\colon \Gamma_i \Rightarrow (\Delta_i \mapsto (\Theta_i \Rightarrow \phi_i))}{\wedge_i\ p_i \Rightarrow \wedge_k\ G\phi_i}$$

- ***each instance is a PLTL tautology***
- ***typically entire proof is one application of this rule***

Temporal case splitting

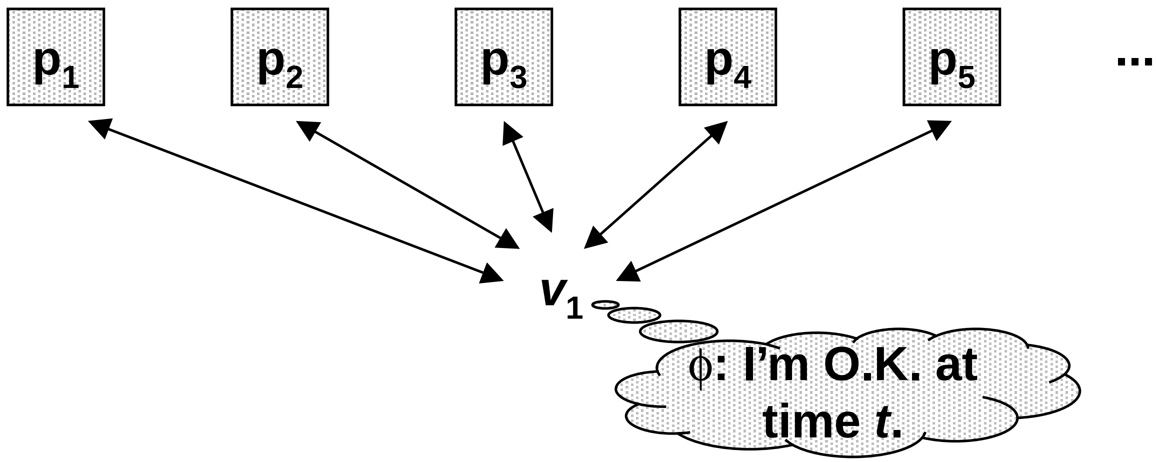


Idea:
Split cases on most recent writer *w* at time *t*.

$$\frac{\forall i:\ p_i \Rightarrow G((w=i)\Rightarrow \phi)}{\wedge_i\ p_i \Rightarrow G\phi}$$

Rule can be used to decompose large arrays

Combine with circular reasoning



To prove case *w*=*i* at time *t*, assume general case up to *t*-1:

$$\frac{\forall i\colon\ p_i \Rightarrow G(\phi \mapsto (w = i) \Rightarrow \phi)}{\wedge_i\ p_i \Rightarrow G\phi}$$

still have unbounded cases to prove...

Reduce cases by symmetry

- Consider formulas with uninterpreted functions and equality
  - validity of formulas preserved under permutations of U
  - that is, if

    I is an interpretation of free parameters

    π is a permutation (bijection) on U,

$$(U, I) \models f \qquad \textbf{iff} \qquad (U, \pi \circ I) \models f$$

Hence, a finite set of representative cases is sufficient to prove f is valid for all interpretations I.

(one for each partition of the parameters)

A simple example

- Suppose we want to prove

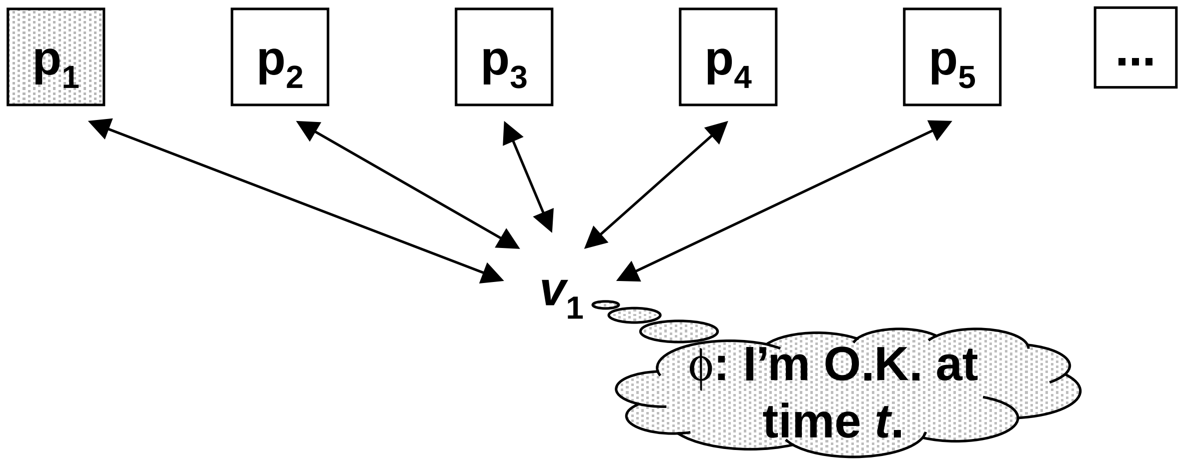
$$\forall i, j: G(i = j) \Rightarrow X(j = i)$$

  - two cases suffice, e.g.: (i=0, j=0) and (i=0, j=1)
  - since both are valid, i.e.,

$$G(0 = 0) \Rightarrow X(0 = 0)$$
$$G(0 = 1) \Rightarrow X(1 = 0)$$

we know the proposition is true in general

An array of processes



By symmetry, suffices to prove that writes by p_1 are O.K.:

$$\frac{p_1 \Rightarrow G(\phi \mapsto (w = 1) \Rightarrow \phi)}{\bigwedge_i \; p_i \Rightarrow G\phi}$$

Data type reductions

- Problem: variables ranging over unbounded set U
- Solution: reduce U to finite set by abstraction, e.g.,

$$\widehat{U} = \{\{i\}, U \setminus i\}$$

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

- Need a sound abstract interpretation, such that

$$(\widehat{U}, \widehat{I}) \models \widehat{\phi} \quad \textbf{implies} \quad (U, I) \models \phi$$

Data type reduction in SMV

- Abstract values represent sets of concrete values

$$\widehat{U} = \{\{i\}, U \setminus i\}$$

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

$$\widehat{f}\,\widehat{x} \leq \widehat{f\,x}$$

- Examples:
  - Equality

$\widehat{=}$	$\{i\}$	$U \setminus i$
$\{i\}$	1	0
$U \setminus i$	0	$\perp$

Abstraction, continued...

– Function symbol application

x	$\{i\}$	$U \setminus i$
$\widehat{f}(x)$	$\widehat{f(i)}$	$\perp$

Unbounded array reduced to one fixed element!

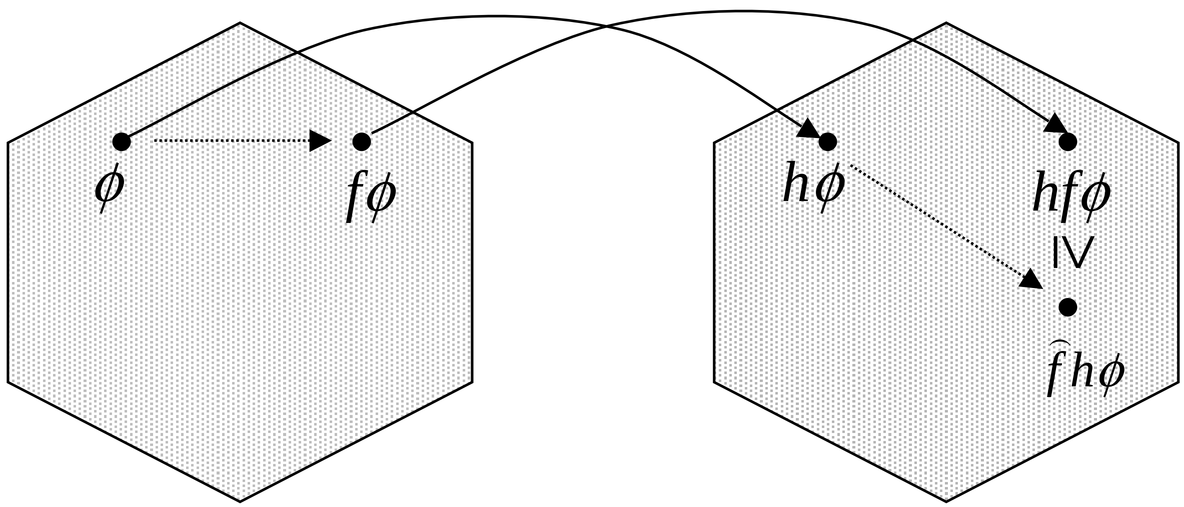
– Other operators ...

$$0 \mathbin{\widehat{\wedge}} \perp = 0 \qquad 1 \mathbin{\widehat{\wedge}} \perp = \perp \qquad \textit{etc...}$$

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

Property preserving abstractions

- Homomorphism from concrete to abstract domains



- By induction, $h\phi \geq \hat{\phi}$ thus if $\hat{\phi}$ is valid, then ϕ is valid.

...but we may get $\hat{\phi} = \perp$

A simple example

- Suppose we want to prove

$$G(f(x) = x \Rightarrow f(f(x)) = x)$$

  – Introduce parameters $i = x$, $j = f(i)$

  – Choose representative cases (i=0, j=0) and (i=0, j=1)

  – In first case: U abstracts to $\widehat{U} = \{\{0\}, U \setminus 0\}$

  two finite state variables: $x, f(0) \in \widehat{U}$

  – Both cases are valid

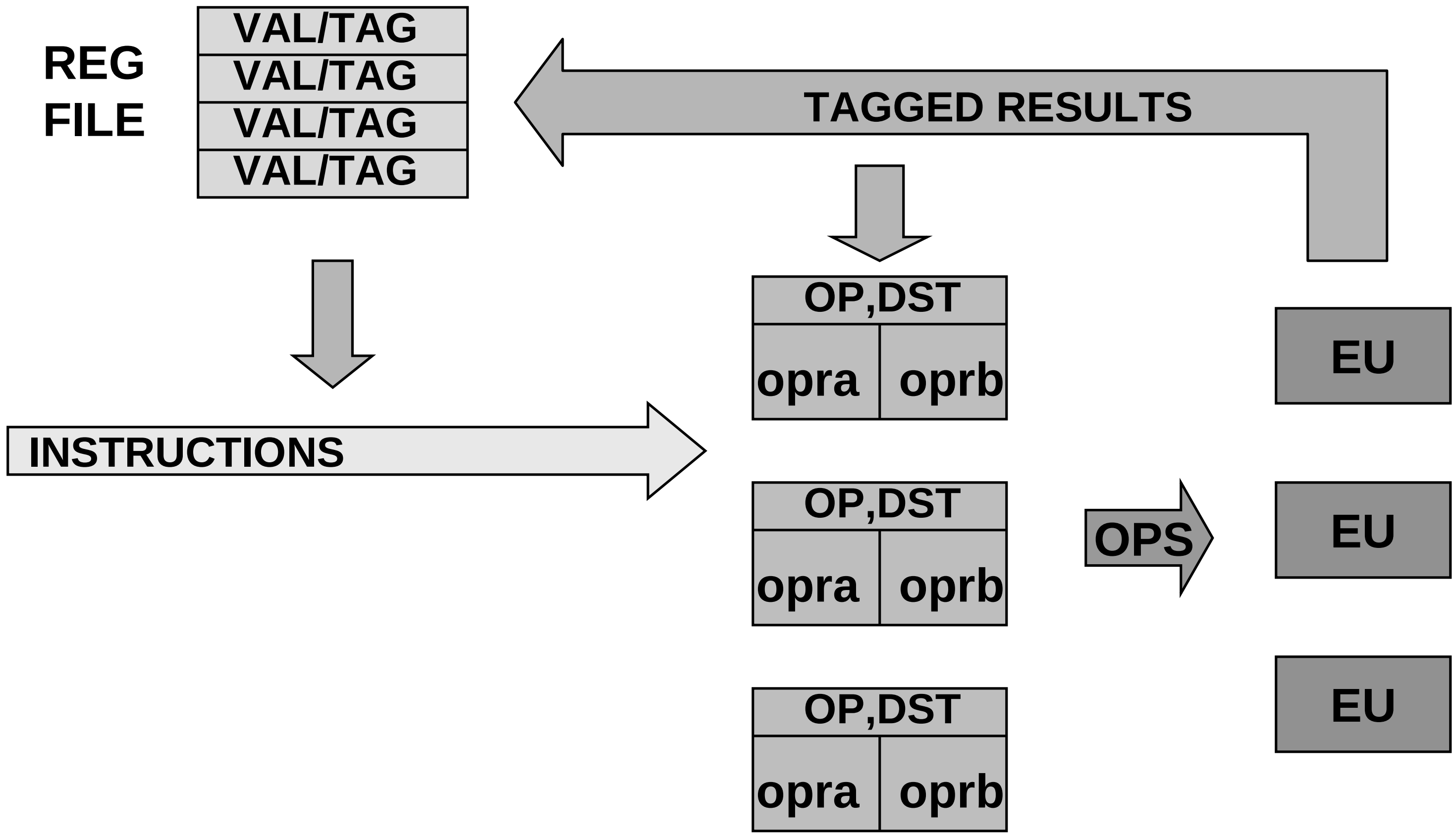
Note: this gives a decision procedure for non-temporal case, but temporal case is undecidable.

Overview of proof strategy

- Decompose structurally using circular compositional proof
- Parameterize to decompose large structures
- Reduce to finite number of cases using symmetry
- Reduce to bounded variables using data abstraction
- Check validity with model checker

Illustration: Tomasulo's algorithm

- Execute instructions in data flow order



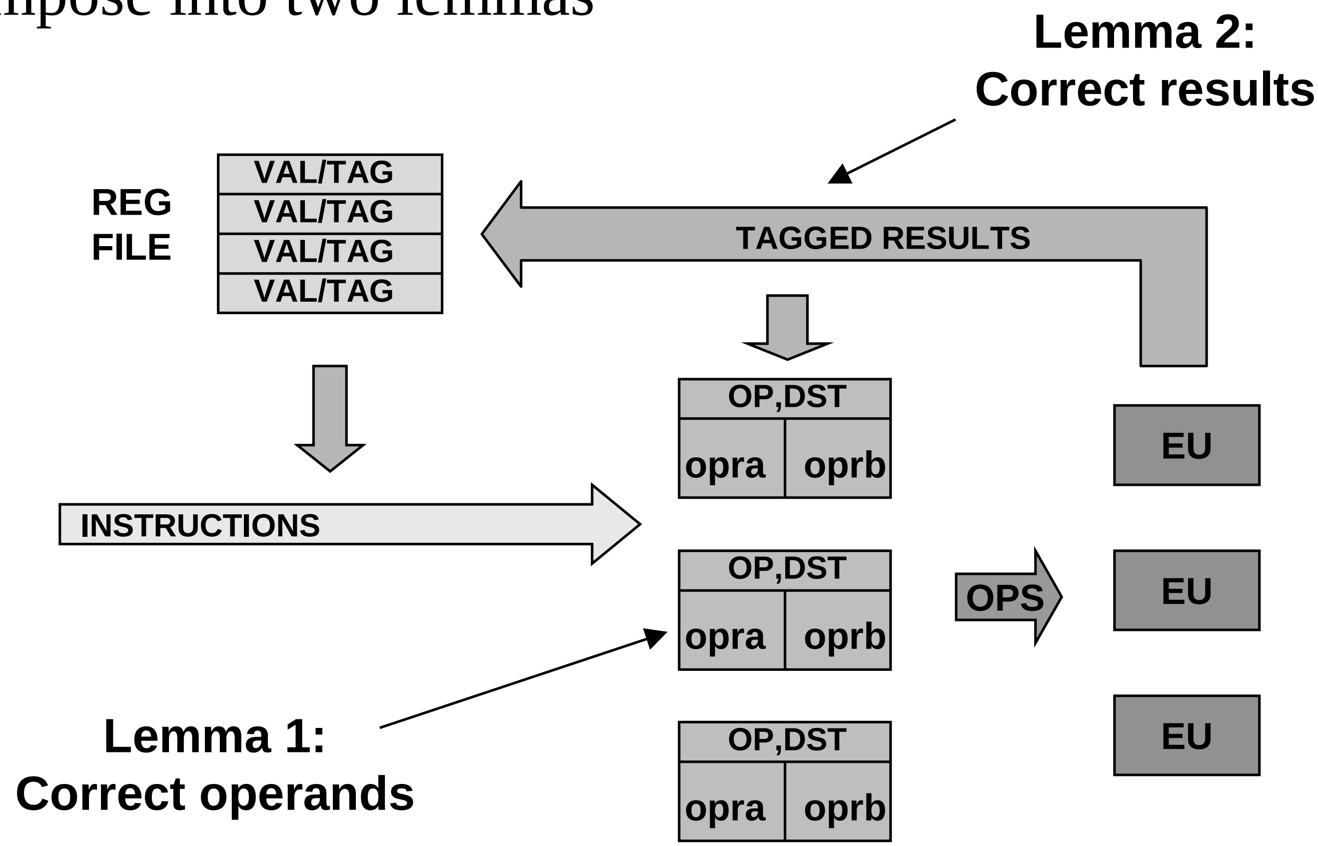
Unbounded resources

- The following types are unbounded:
  - 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.

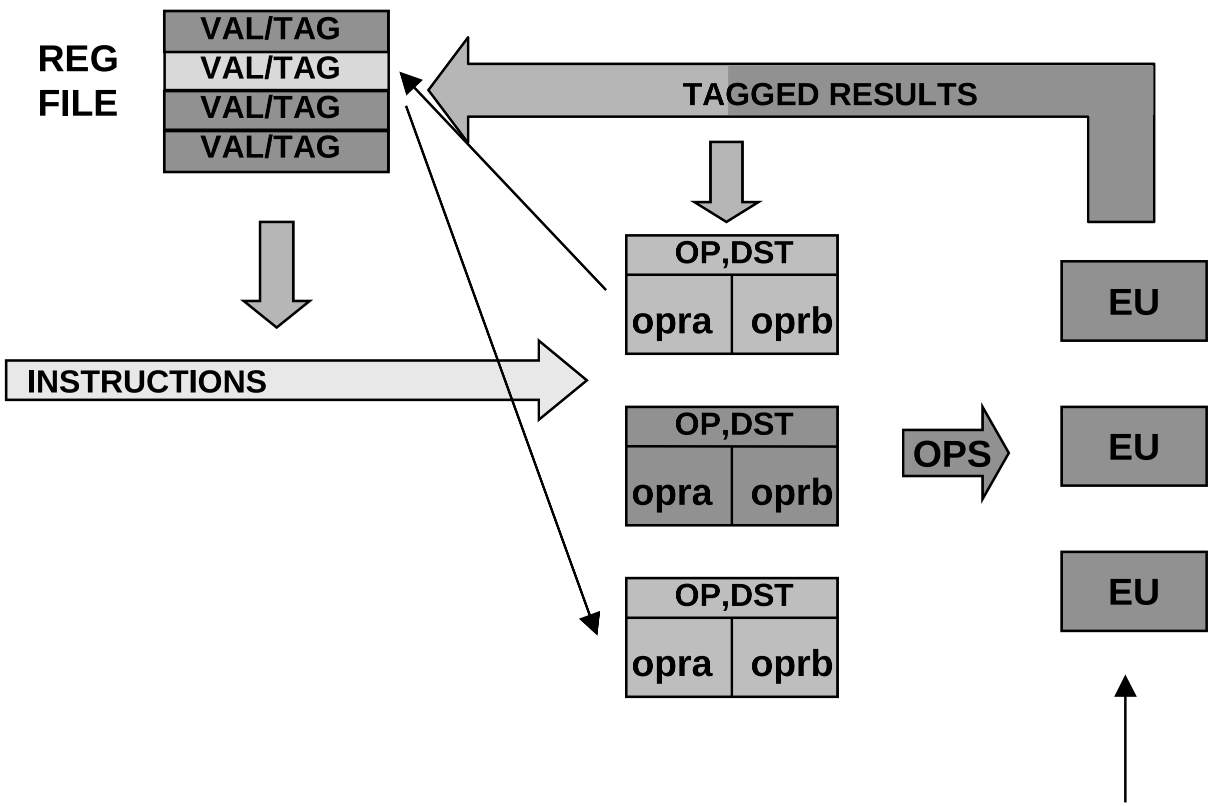
Compositional rule

- Decompose into two lemmas



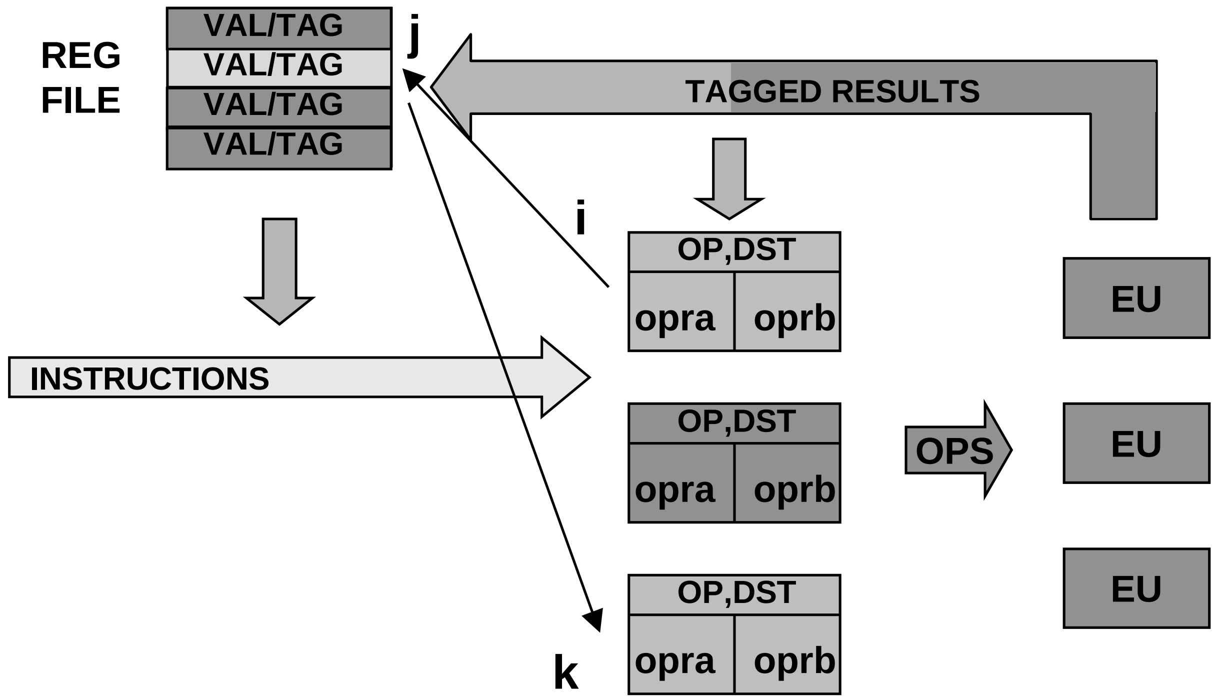
Computing each result and operand uses finite resources

Temporal case splitting



"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 unbounded range to finite types
- Specific reduction is chosen based on property parameters

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

  REG $\rightarrow$ {j, REG\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

(And we can handle results correctness lemma...)

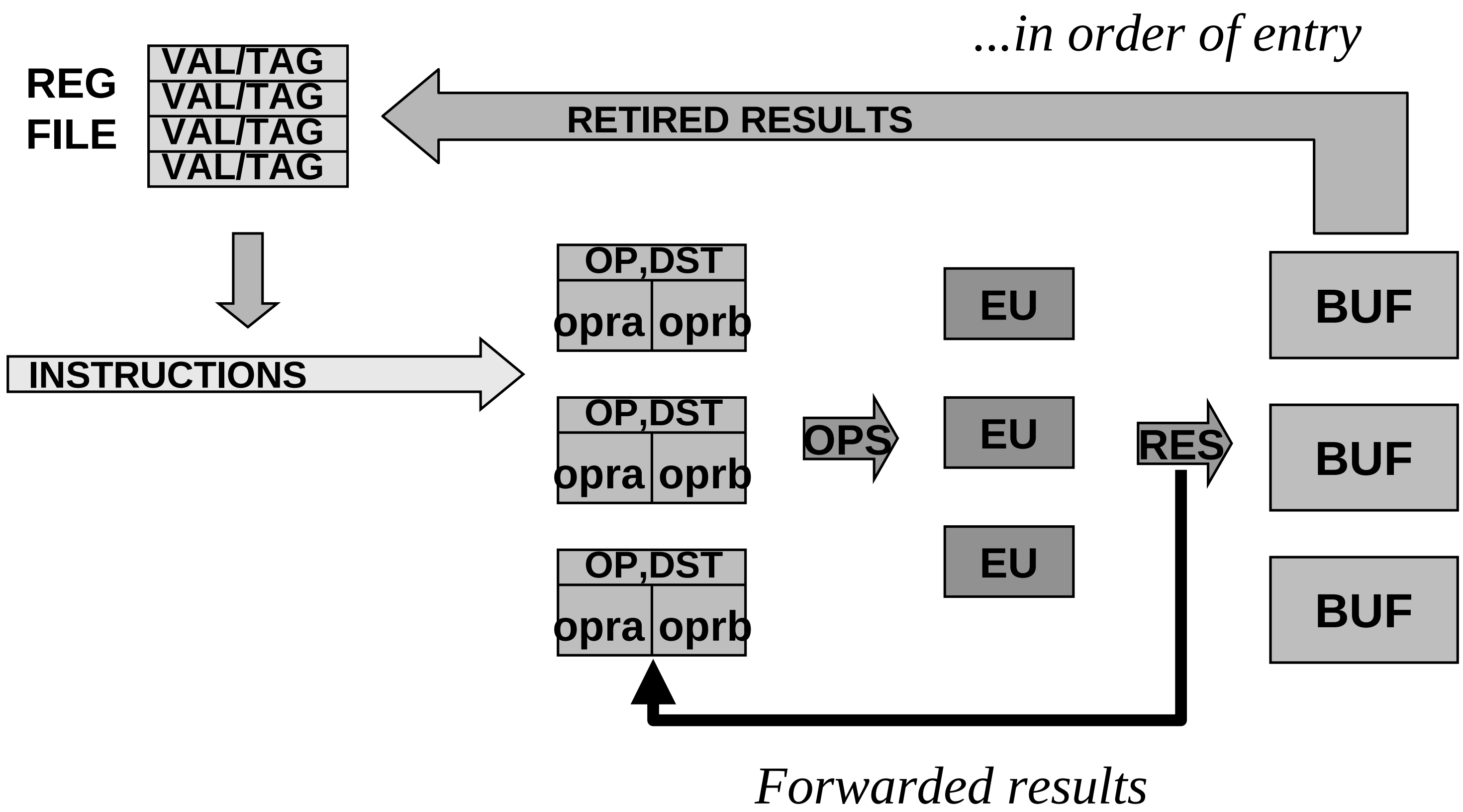
Result

- 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

...but in what way robust?

Adding in-order retirement



Reusing the proof

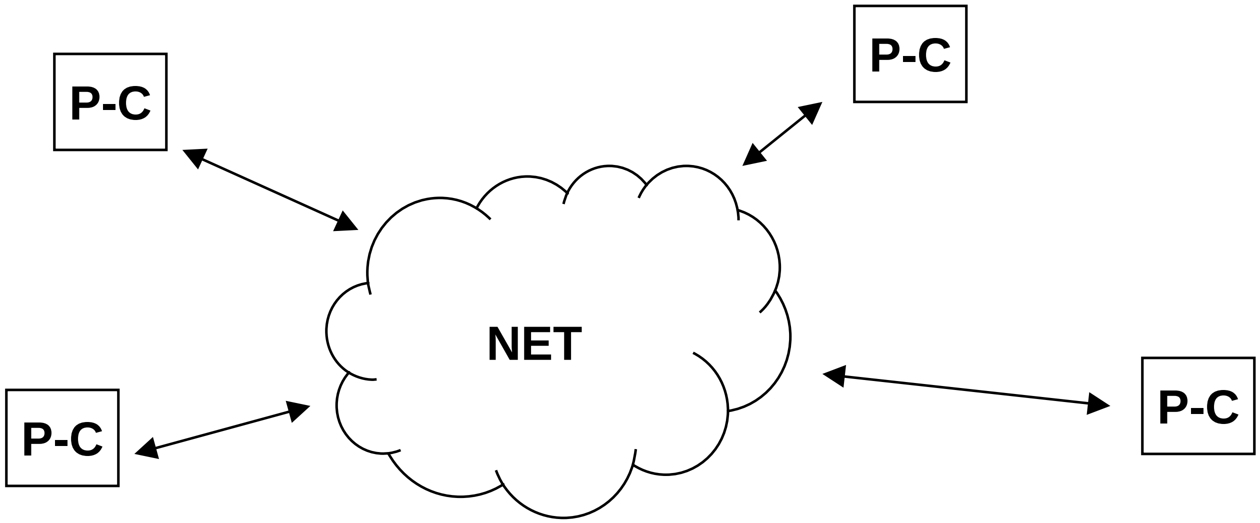
New version is verified without modifying one proof line...

- 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 of
the abstract models, we do not have to specify invariants of most signals.
This improves reusability of the proof.

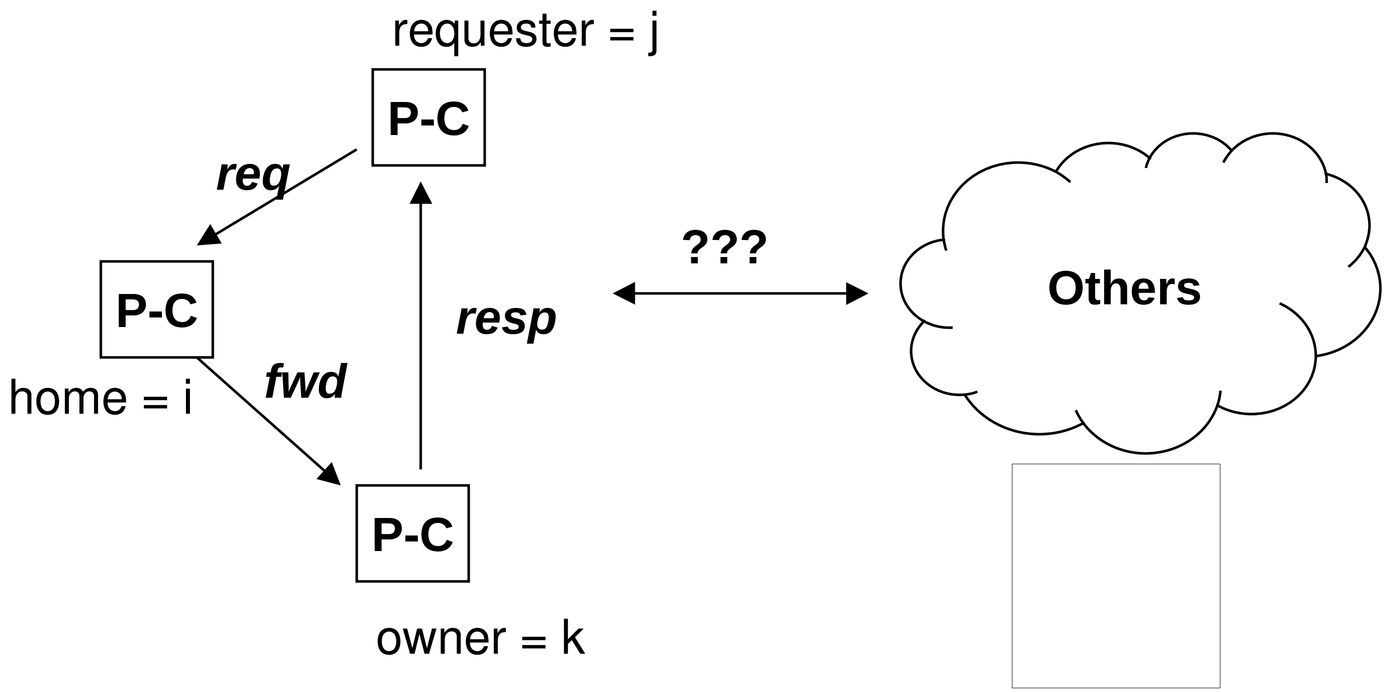
Illustration: FLASH protocol



- Message passing protocol maintains cache consistency
- Want to prove liveness, consistency with abstract memory model

Proof strategy

- Temporal case split based on three major protocol roles:



Non-interference

- Must assume that "others" do not interfere:
  - do not execute writes to store
  - do not send fwd and resp messages
  - etc...
- Discharge non-interference conditions
  - split cases on sender of bad message
  - assume general non-interference condition up to time t-1

Note: this proof is more difficult than Tomasulo example, because of more flexible interconnection, allowing more possible interference.

Natural numbers and induction

- Suppose we introduce a successor function (+1) and inequality relation < over *U*...
  - Given a set of terms of the form

$$c, \quad i \pm c$$

  where i is a parameter, and c is a natural number, enumerate all the feasible inequality relationships.

For example, given the terms 0, *i*-1 and *i*, we have three cases:

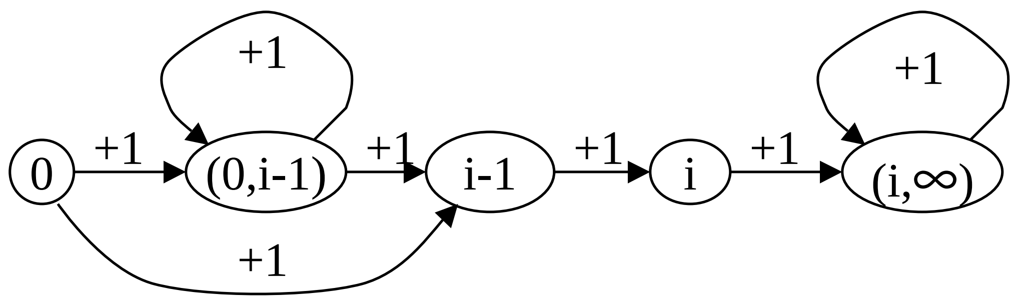
$$i - 1 < 0 = i$$

$$0 = i - 1 < i$$

$$0 < i - 1 < i$$

Abstractions of the naturals

- For each case, choose an abstraction of U, where the abstract values are fixed values, or intervals on U...
  - for example, in the case $0 < i-1 < i$,



  - note:

    abstract model is finite

    original model homomorphic to this model for any i satisfying $0 < i-1 < i$

Simple example

- A counter increments at each time step...

$$x = 0 \wedge G(x' = x + 1)$$

- Prove that every natural number is eventually reached...

$$\forall i: F(x = i)$$

  – by induction on i, prove

$$F(x = 0) \qquad \forall i > 0: F(x = i - 1) \Rightarrow F(x = i)$$

  – choose abstraction using the terms 0, *i*-1 and *i*.

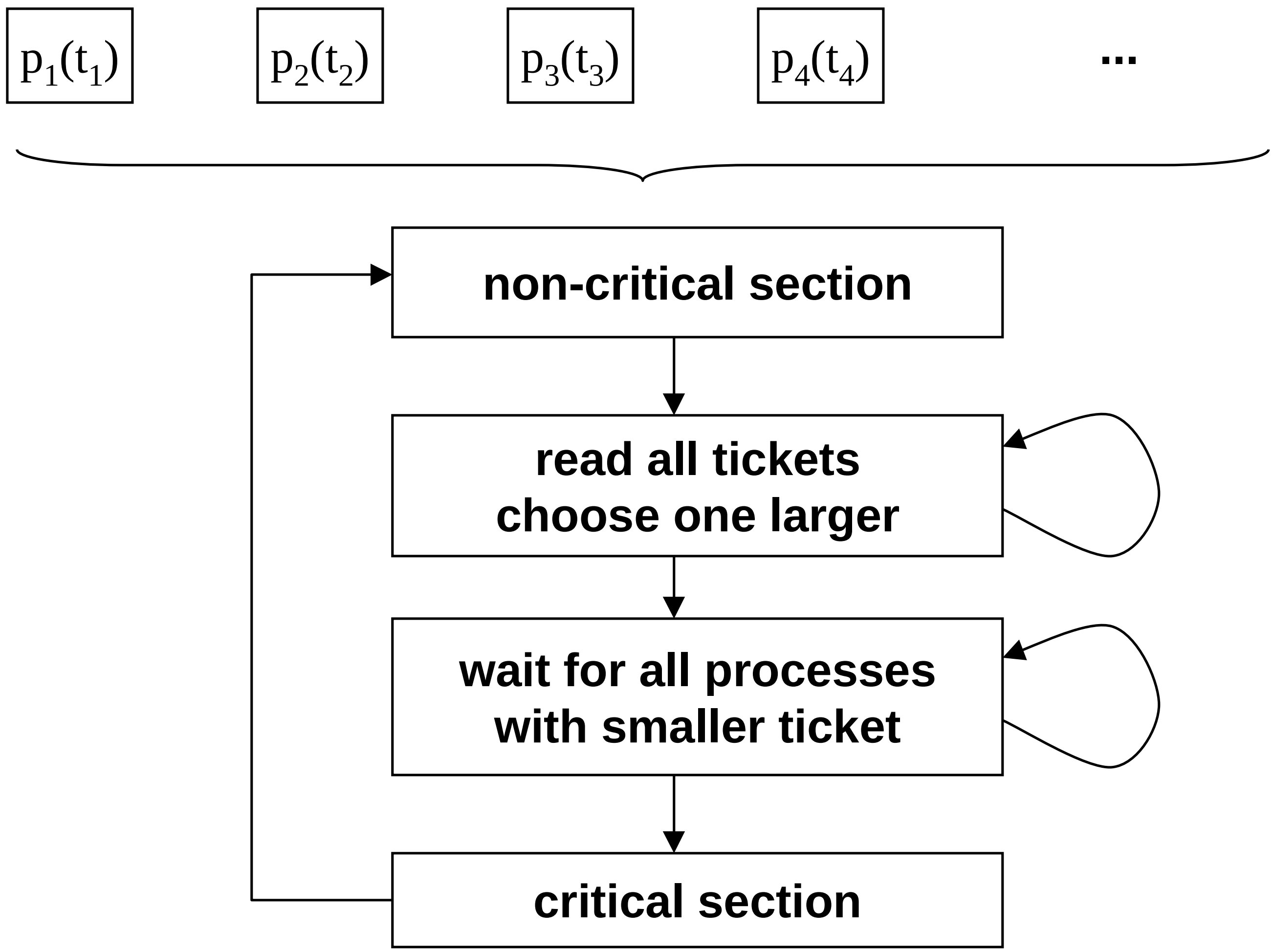
  – must check three finite abstract models:

$$i - 1 < 0 = i$$

$$0 = i - 1 < i$$

$$0 < i - 1 < i$$

Illustration: Lamport's Bakery algorithm



Proof overview (Qadeer and Saxe)

- Prove liveness by induction on ticket numbers
  - Split cases on process p_j that process p_i is waiting for...

    ... $\boxed{p_j(t_j)}$ ... $\boxed{p_i(t_i)}$ ...

  - Assume by induction that j terminates if $t_j<t_i$.

    Note, p_j must get higher ticket than p_i on next iteration*
  - By induction on j, process p_i exits wait loop

Note:
- **reduction to finite number of processes**
- **model checking gets property * automatically**

Some observations

- Protocol not "control dominated"
  - i.e., not ideal application for model checking
  - nonetheless, some significant temporal reasoning handled by model checking
- Proof effort considerably greater than previous examples, if measured per line of verified code.

This problem is perhaps at the extreme end of what you would want to do with these techniques

Recap

- Proof strategy
  - Structurally decompose using circular compositional rule

    note, internal specifications usually w.r.t. reference model
  - Parameterize (yields unbounded set of interpretations)
  - Map to finite set of abstract finite interpretations

    based on some case analysis, such as equality/inequality
  - Use model checking to verify these finite interpretations

Last slide (really)

- We observed in some examples
  - concise proofs
  - fewer manually constructed invariants
  - more robust proofs (w.r.t. design changes)
- This strategy implemented in automatic proof assistant on top of SMV model checker

http://www-cad.eecs.berkeley.edu/~kenmcmil/smv