Parameterized Verification of Deadlock Freedom in Symmetric Cache Coherence Protocols

Brad Bingham[1] Jesse Bingham[2] Mark Greenstreet[1]

[1]University of British Columbia, Canada

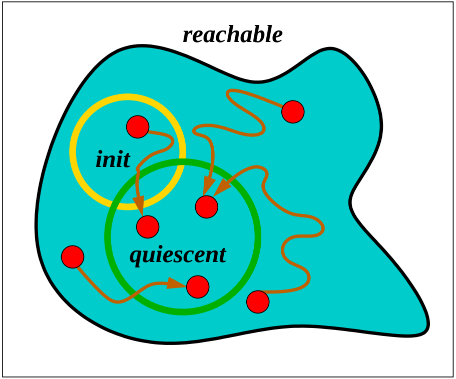
[2]Intel Corporation, U.S.A.

November 2, 2011

FMCAD

Outline

The Problem: Deadlock-Freedom

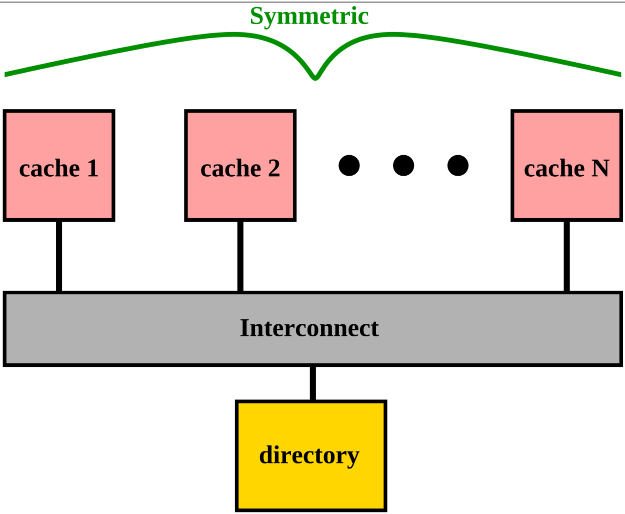


- "Is it deadlock-free?" $\equiv$ "Is there a path from each reachable state to a quiescent state?"
  - "quiescent" $\equiv$ "nothing is pending"
  - In CTL: $\mathrm{AG}\,\mathrm{EF}\,q$ (more generally, $\mathrm{AG}\,(p \rightarrow \mathrm{EF}\,q)$)
  - Cheap to model check; rules out some liveness bugs; avoids fairness

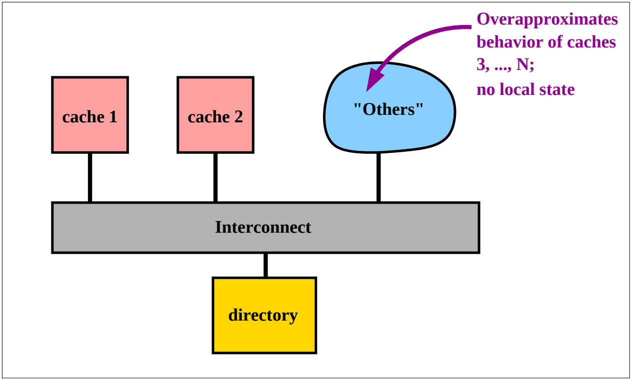
Overview: Parameterized Systems

- A system $\mathcal{S} = (S, I, T)$ is a tuple of states S, initial states I and transitions T
- A parameterized system is a mapping from the naturals to systems. $\mathcal{S}(N) = (S(N), I(N), T(N))$.
  - In cache coherence protocols, the parameter might correspond to "number of caches", "number of address", "length of some buffer", etc. In our examples, it's "number of caches".
- Verifying a safety property of $\mathcal{S}(N)$ for all N is algorithmically undecidable.
- Previous work addresses this problem. One promising approach is based on compositional reasoning (CEGAR + Human Ingenuity).
  - [McMillan99], [Chou+04], [O'Leary+09]

Parameterized Cache

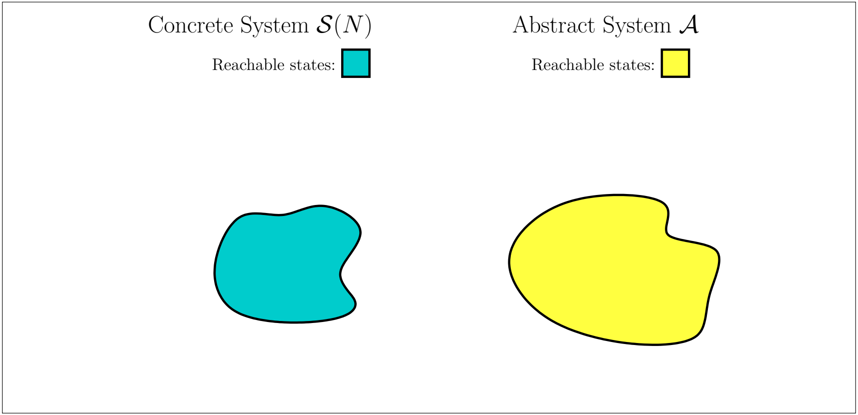


Parameterized Cache Abstraction

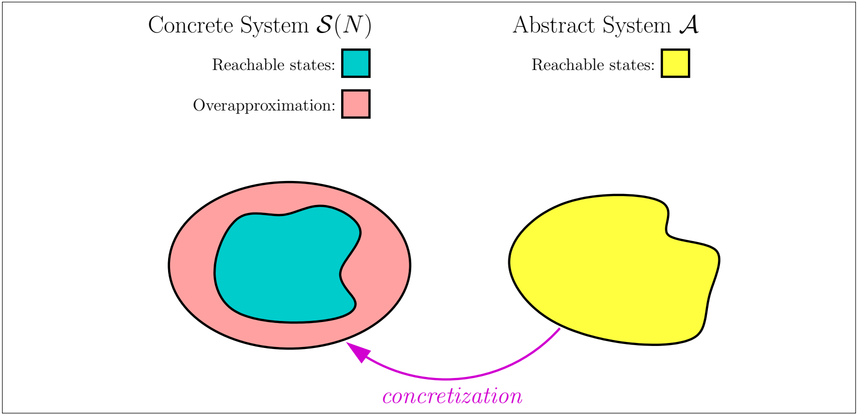


- Finite-state, overapproximate abstraction of $\mathcal{S}(N)$ for all $N > 2$
- Suitable for model checking

Abstraction Relation

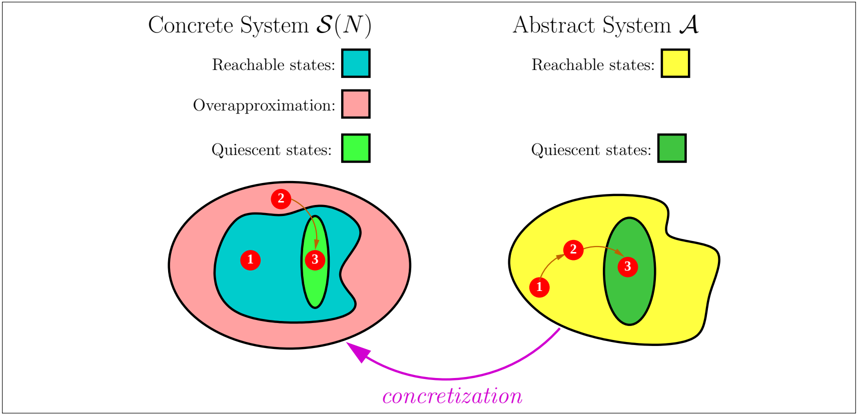


Abstraction Relation



- ✓ Abstraction allows us to infer concrete safety properties ☺

Abstraction Relation



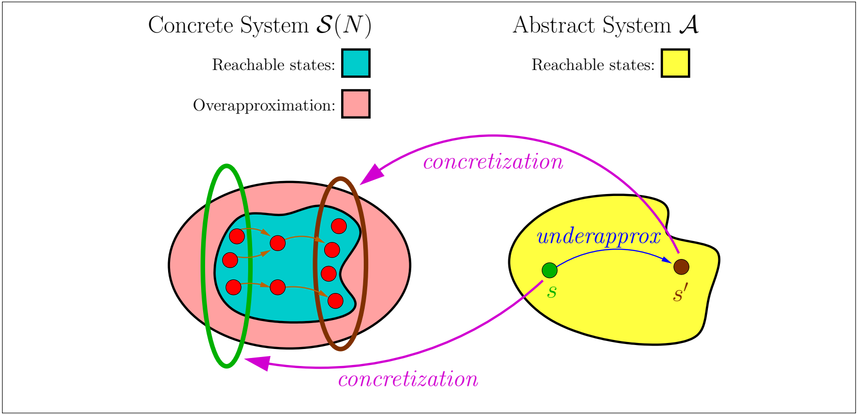
- ✓ Abstraction allows us to infer concrete safety properties ☺
- ✗ Cannot infer concrete deadlock-freedom properties ☹

Paths don't (necessarily) concretize

Underapproximate Transitions

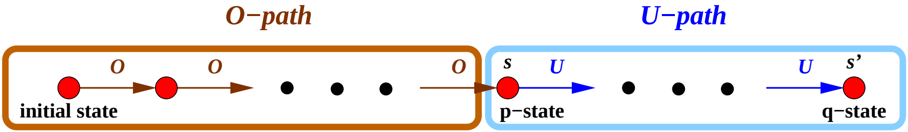
Suppose (s, s') is an abstract transition where every reachable state in the concretization of state s has a path to some state in the concretization of state s'.

This transition is called **underapproximate**.



Mixed Abstraction

- A Mixed Abstraction[LT88][Dams+97] is like an abstract transition system, but has two sets of transitions: overapproximate (O) and underapproximate (U).
- Model checking $\mathrm{AG}\,(p \rightarrow \mathrm{EF}\,q)$ in mixed abstraction $\mathcal{M}$: for each O-reachable p-state, find a U-path to some q-state.

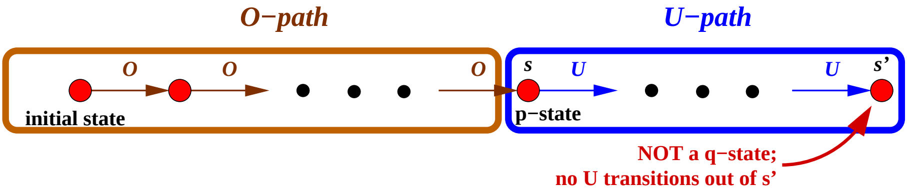


Theorem

If $\mathcal{M} \models \mathrm{AG}\,(p \rightarrow \mathrm{EF}\,q)$*, then* $\mathcal{S}(N) \models \mathrm{AG}\,(p \rightarrow \mathrm{EF}\,q)$*.*

Insufficiency

What if model checking fails?

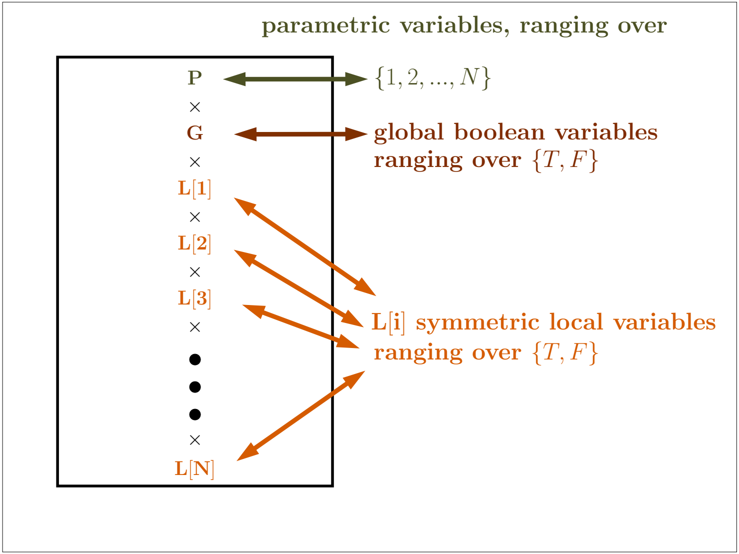


1. Perhaps O is too weak
   - State s has no reachable concretization in $\mathcal{S}(N)$
   - Remedied by strengthening O (covered by previous literature in parameterized safety)
2. Perhaps U is too strong
   - A U-path from s gets "stuck" before a q-state is reached
   - Proving that transitions are underapproximate is not addressed by extensive previous work; this is our focus

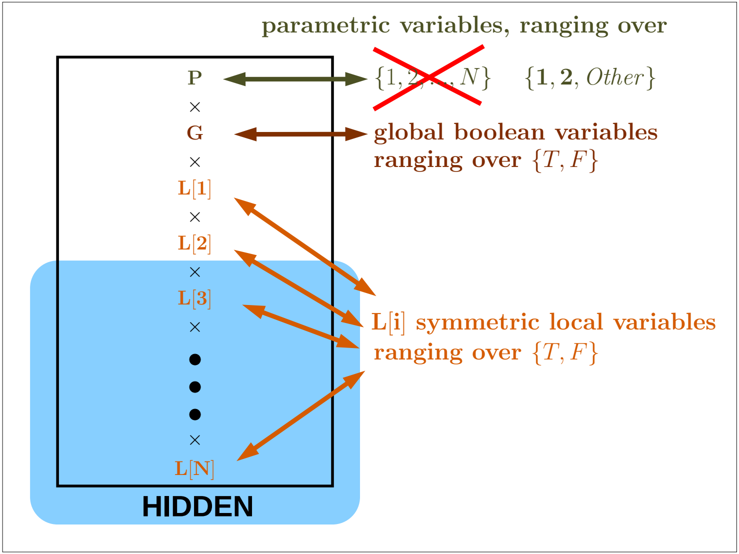
Strategy

- Assume a symmetric, parameterized system $\mathcal{S}(N)$ expressed with guarded commands (or "rules"); assume an overapproximate abstraction of $\mathcal{S}(N)$
  - Some restrictions to syntactic form
- Use the abstraction as a starting point for the mixed abstraction
- Approach: Use syntactic analysis to find "trivially" underapproximate transitions U
- Then: Prove selected guarded commands of O are in fact underapproximate by leveraging symmetry and model checking the mixed abstraction.
  - The approach depends on the syntactic form of the rule
  - All of our methods rely on "path symmetry"

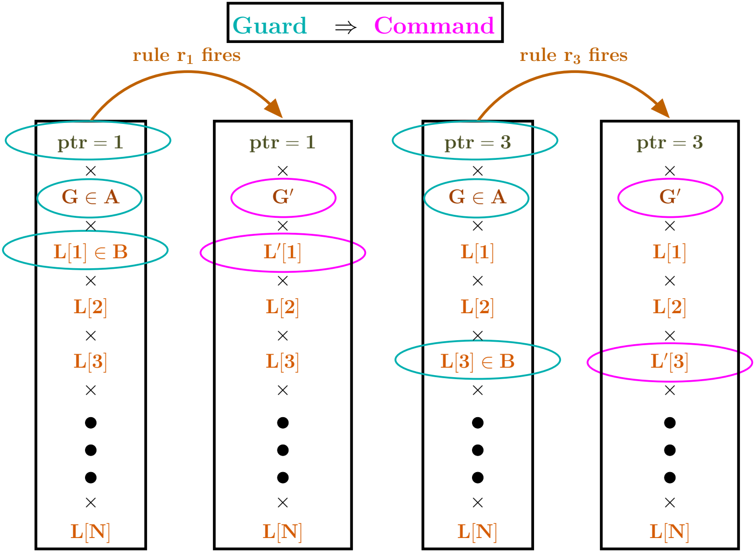
Concrete States



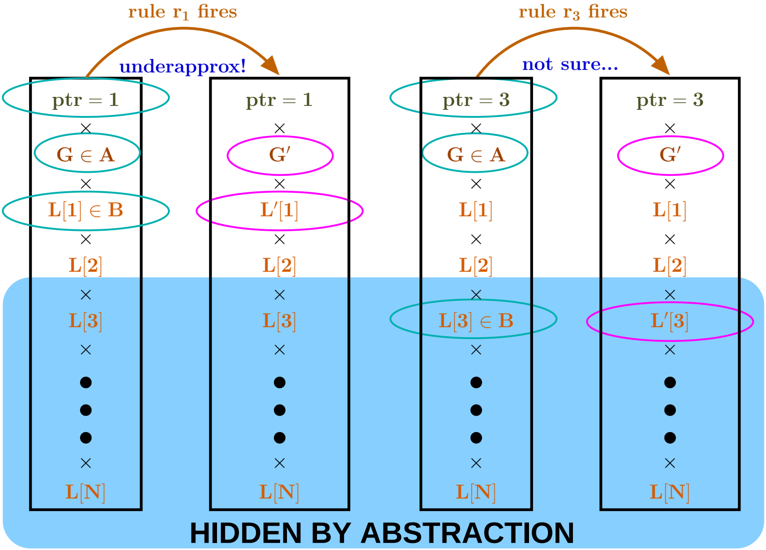
Abstract States



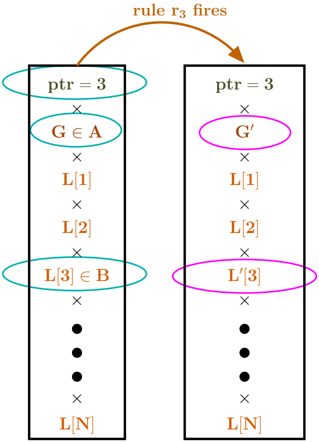
(Symmetric) Guarded Commands



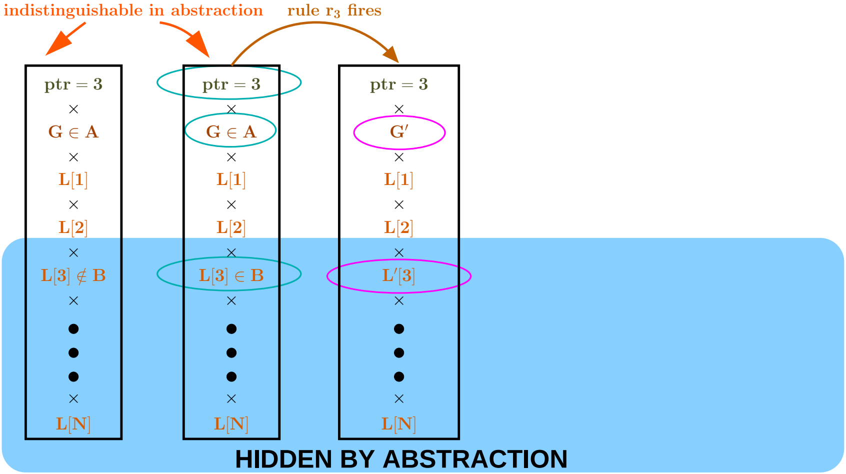
(Symmetric) Guarded Commands



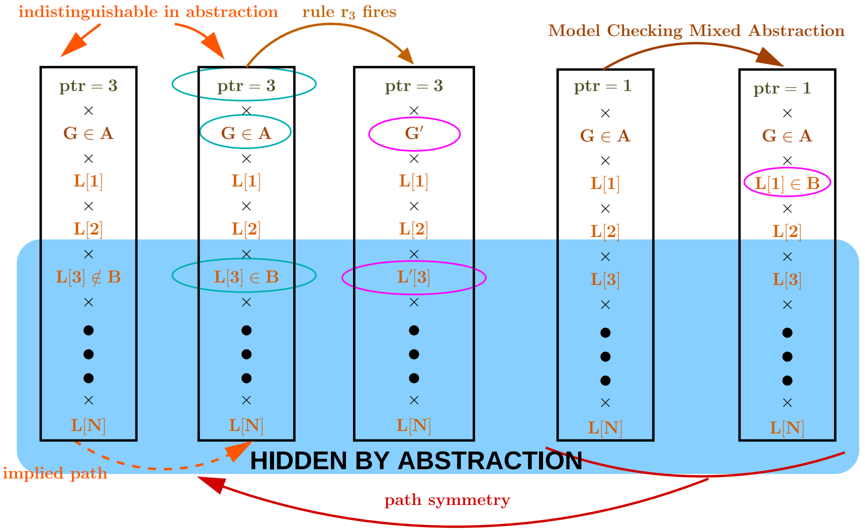
Abstracted Local State: $\mathbf{L[ptr]} \in \mathbf{B} \wedge \mathbf{G} \in \mathbf{A}$



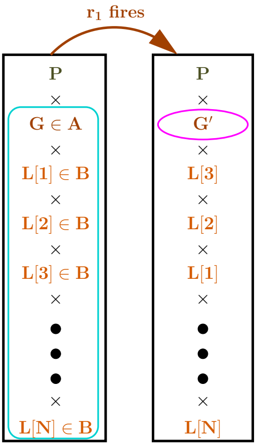
Abstracted Local State: $\mathbf{L[ptr]} \in \mathbf{B} \wedge \mathbf{G} \in \mathbf{A}$



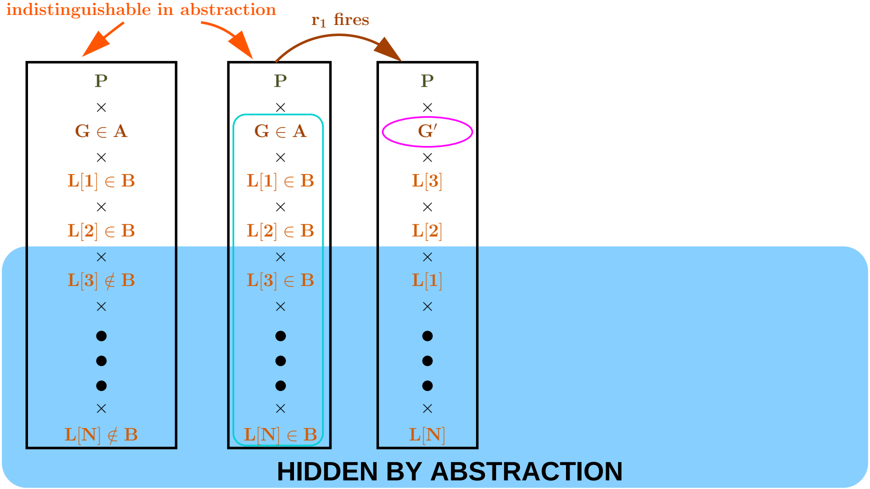
Abstracted Local State: $\mathbf{L[ptr]} \in \mathbf{B} \wedge \mathbf{G} \in \mathbf{A}$



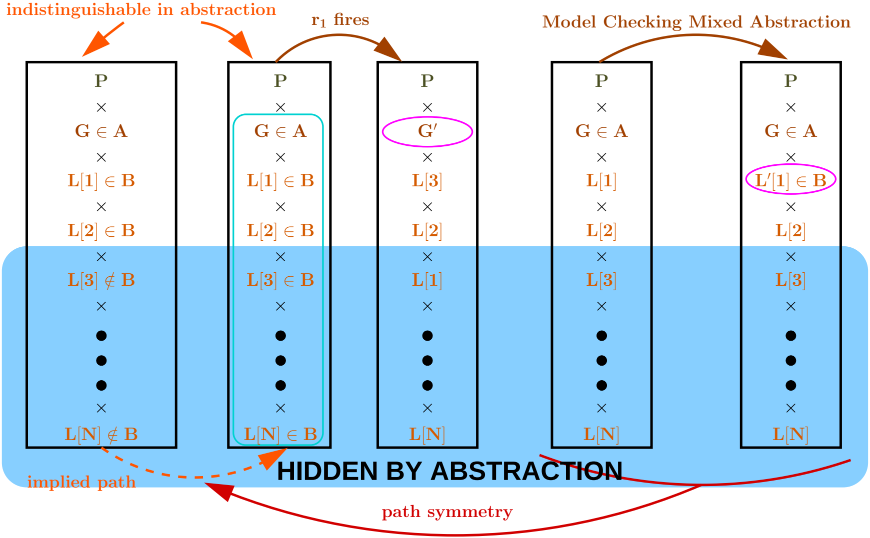
Abstracted Universal Quantifier: $\mathbf{G} \in \mathbf{A} \wedge \forall \mathbf{i}.\ \mathbf{L[i]} \in \mathbf{B}$



Abstracted Universal Quantifier: $\mathbf{G} \in \mathbf{A} \wedge \forall \mathbf{i}.\ \mathbf{L[i]} \in \mathbf{B}$

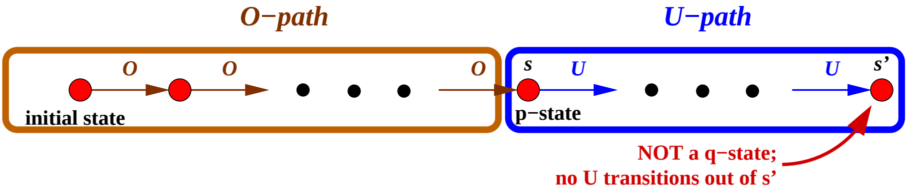


Abstracted Universal Quantifier: $\mathbf{G} \in \mathbf{A} \wedge \forall \mathbf{i}.\ \mathbf{L[i]} \in \mathbf{B}$



Case Studies

- German and Flash cache coherence protocols
- Proved "For any number of caches, the system can always clear the communication channels and directory is not in a waiting state"
- Overapproximate transitions from Murφ models of strengthened abstractions borrowed from [Chou+04]
- Underapproximate transitions proven "on-demand"
  - Some transitions are trivially underapproximate by syntactic analysis
  - Others are proven underapproximate with our methods, when the model checker indicates a rule will help, i.e., enabled transitions of O at s'



Automation?

Can this process be automated?

- YES: Detection of a "useful" rule to prove underapproximate
- YES: Application of model checking for the appropriate reasoning (depends on the form of the guard)
- UNSURE: What to do if our tricks fail
- HOWEVER: When our tricks don't work, it's a sign that the rule may NOT be underapproximate.
- WHAT THEN?: Perform some manual strengthening similar to previous work!

Future Work

- Automation: As mentioned, in a theorem proving environment.
  - Automatically extract from O the weakest U supported by our methods
- Other Problems: Parameterize over addresses? (OpenSPARC)
  - Still symmetric, but guards of rules take different syntactic form
- Other Properties: Consider request *req* and response *resp*:
  - Prove "When *req* is outstanding, there exists a path to *resp*"
  - $\mathsf{AG}\,(\mathit{req\text{-}pend} \rightarrow \mathsf{EF}\,\mathit{resp})$

Wrap-Up

- Presented a tractible method for proving parameterized deadlock-freedom
- Builds directly on previous work in parameterized safety ([McMillan99,Chou+04])
- Expectation: Method offers low-hanging deadlock-freedom result following application of these methods, leveraging a tight overapproximation
- Thank-you! Questions?