Automated Refinement Checking of Asynchronous Processes

Rajeev Alur

University of Pennsylvania
www.cis.upenn.edu/~alur/

Intel Formal Verification Seminar, July 2001

Refinement Checking

Problem

Given two descriptions of the same design Imp and Spec, check if every behavior of Imp is allowed by Spec:

Imp < Spec

Why relevant ?

Writing Spec as another state machine may be easier than listing all temporal logic formulas of interest

Promotes hierarchical design by successive refinements

Examples

Cache-coherent memory < Abstract serial memory

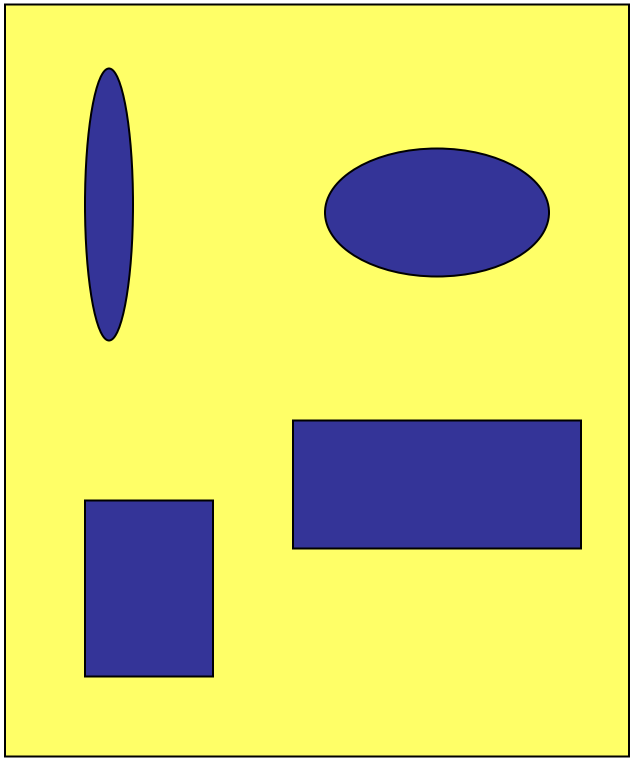
Pipelined implementation < ISA spec

Point-to-Point Protocol

- Popular networking protocol for establishing connections remotely
- Goal: To verify the actual implementation
- Specification: RFC 1661 (standard)
  - Specified in tabular format
- Implementation: ppp version 2.4.0
  - available in various Linux distributions
  - C code

Why Modular Reasoning?

- Behavior of a component can be computed from behaviors of its parts
- Components can be analyzed in isolation
- Assume-guarantee rules -> Scalable analysis



Goal: Composable Behavioral Interfaces!

Model Checker MOCHA

Joint project with UC Berkeley

Innovations aimed at exploiting modularity

- Modeling language: Reactive Modules
- Requirements: Alternating Temporal Logic
- Symbolic model checking
- Game-based abstractions
- Hierarchical reduction algorithms
- Assume guarantee reasoning

Available from www.cis.upenn.edu/~mocha
www.eecs.berkeley.edu/~mocha

Talk Outline

- ✓ Motivation
- ❑ Refinement check as Reachability
- ❑ Assume-Guarantee Reasoning
- ❑ Hierarchical Reduction
- ❑ Case studies

Reactive Modules

- Hierarchic modeling using composition, hiding, and instantiation
- Well-typed communication interface
- Compositional semantics

  module = (inputs, outputs, traces)
- Proof calculus for simplifying verification goals
- Both synchronous and asynchronous systems
- Modeling of open systems

Weak Refinement

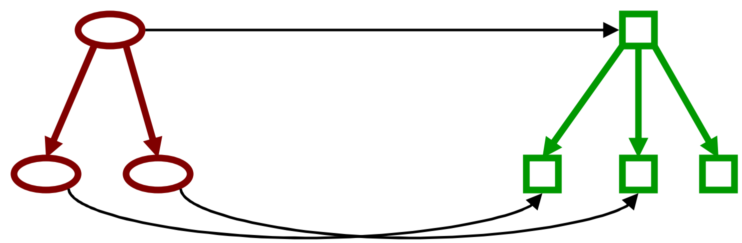
- **Standard Refinement:** inclusion of trace sets (set of traces of Imp is included in the set of traces Spec)
- **Modeling of Asynchrony:** a process may idle or take a step in each round (speeds of different components are independent)
- **Weak refinement:** Trace inclusion, but traces differing only due to stuttering are equivalent

Refinement Check by Search

Suppose all vars of Spec are part of Imp

$$Imp\,(x_1,x_2,\ldots,x_k,y_1,y_2,\ldots\,y_m) < Spec(x_1,x_2,\ldots,x_k)$$

Then, for every reachable state s of Imp, check



Search can be performed enumeratively or symbolically

If Spec has additional variables, user must supply their definitions in terms of Imp variables

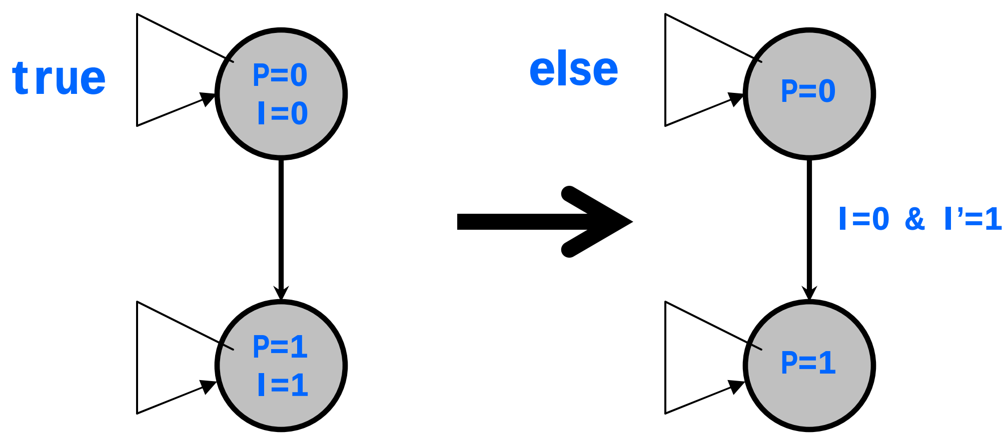
Imp inherits all properties of Spec

Refinement -> Reachability

- Goal: To check I < S
- If S has private variables, complexity of automatic check too high
- Solution: Introduce a module W that defines private vars of S in terms of vars of I (Cospan, Mocha, SMV)
- Checking I || W < S involves reachability analysis
- Can construction of W be automated ?

Automatic Witness Construction

- First strategy: Pick W to be Priv(S) (part of S that controls its private vars)
- Doesn't work for asynchronous processes
- Our strategy: Pick W to be Eager(Priv(S)) (stutter only when all else is disabled)



Spec transition

Witness transition

Witness Construction

- Eager(P) can be viewed as a locally determinized version of P
- If S is deterministic, checking I<S reduces to reachability of analysis of IxS (i.e. update rules for union of variables of I and S)
- Sound, but incomplete, method
- Eager(P) can be constructed easily by syntactic transformation
- Works surprisingly well

Talk Outline

✓ Motivation

✓ Refinement check as Reachability

❑ Assume-Guarantee Reasoning

❑ Hierarchical Reduction

❑ Case studies

Decomposing Refinement Check

- Goal: Reduce I < S to simpler subgoals
- Strategy: I is a composition of many components, so exploit that structure
- If I is I1||I2, rewrite S as S1||S2, so that S1 is abstraction of I1 and S2 is abstraction of I2
- Powerful technique, but requires expertise and “clean” interfaces

Compositional Rule

To prove

It suffices to prove

Assume Guarantee

- Intuition: Proving I1<S1 may require assumptions about the inputs to I1
- Strategy: Use S2 (the specification of I2) as the assumption about the inputs to I1
- Circularity: S1 is established assuming S2 and S2 is established assuming S1
- Not always valid! (key to proof is “non-blocking” interaction, and non-empty trace-sets)
- Long history: Starks85, ChandyMisra88, AbadiLamport93, AlurHenzinger96, McMillan97…

Assume-Guarantee Rule

To prove

It suffices to prove

and

Talk Outline

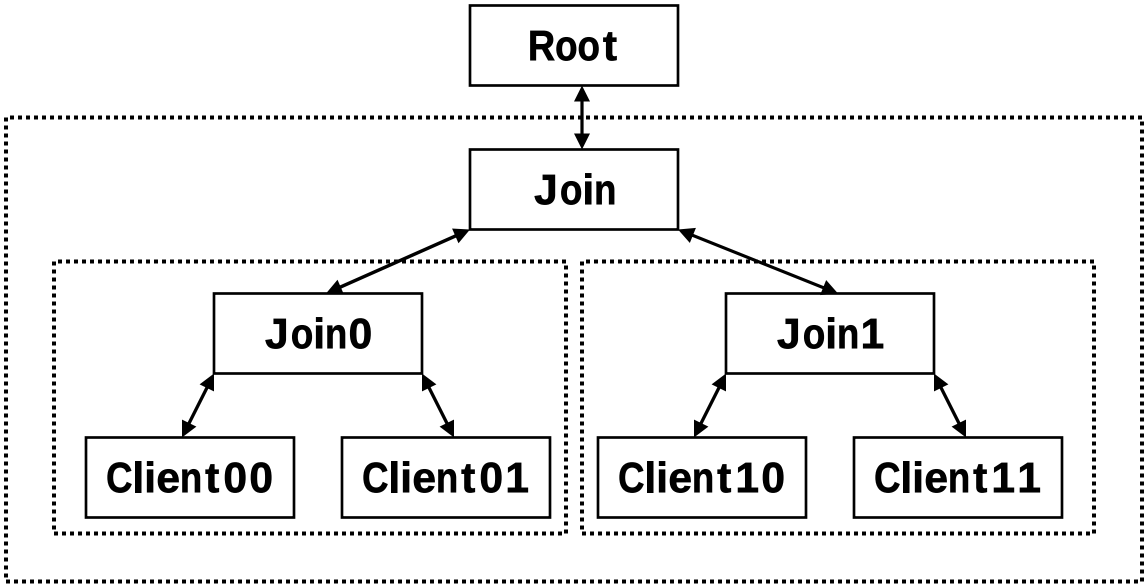
- ✓ Motivation
- ✓ Refinement check as Reachability
- ✓ Assume-Guarantee Reasoning
- ❑ Hierarchical Reduction
- ❑ Case studies

Hierarchic Reduction

- Typical on-the-fly search strategies do not exploit architectural hierarchy
- Compositional minimization works bottom-up
- Can we combine the advantages of the two?
- Solution based on transition hierarchies

Simple reduction strategy based on compressing internal moves (AW: Concur'99)

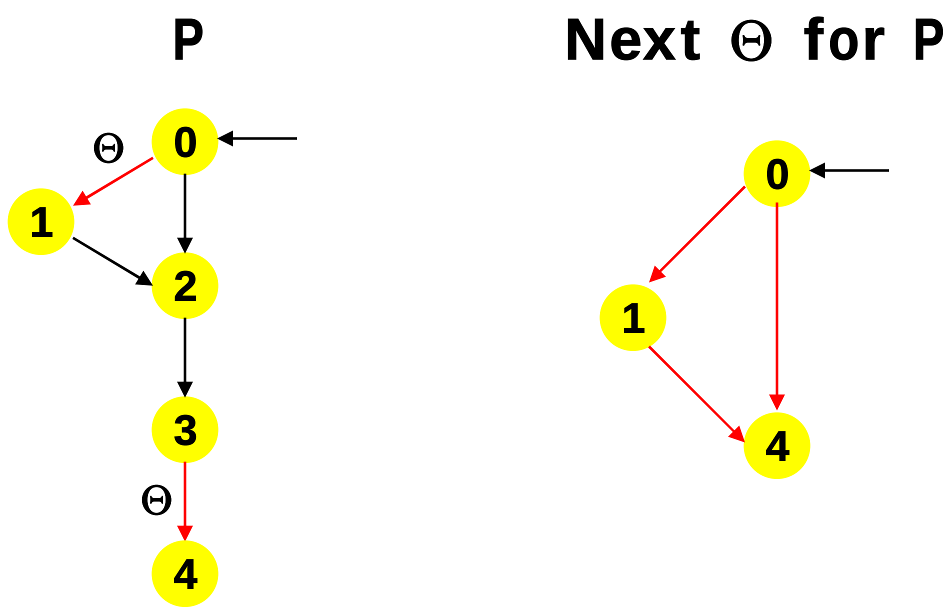
Tree Architecture



E = Root||Join||Join0||Join1||Client00||Client01||Client10||Client11

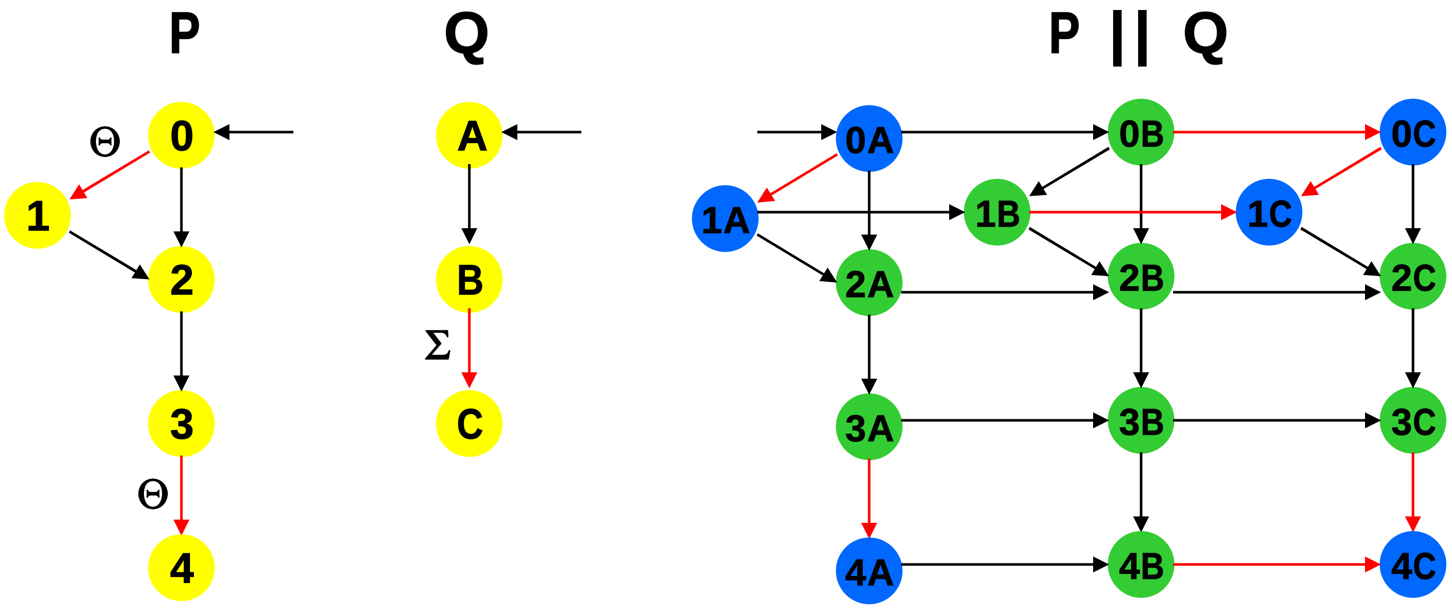
Definition of next operator

Next Θ for P executes P transitions until it encounters a transition in Θ.



Correctness: If Θ includes all “visible” transitions
Then P and Next Θ for P are weakly-similar

State-Space Reduction



R(next Θ for P || next Σ for Q) has blue nodes

R(P || Q) has blue and green nodes

Methodology

- Given E = P || Q || R … and an invariant ϕ,

  check whether "E satisfies ϕ"?
- Transform E to E' by inserting hide and next so

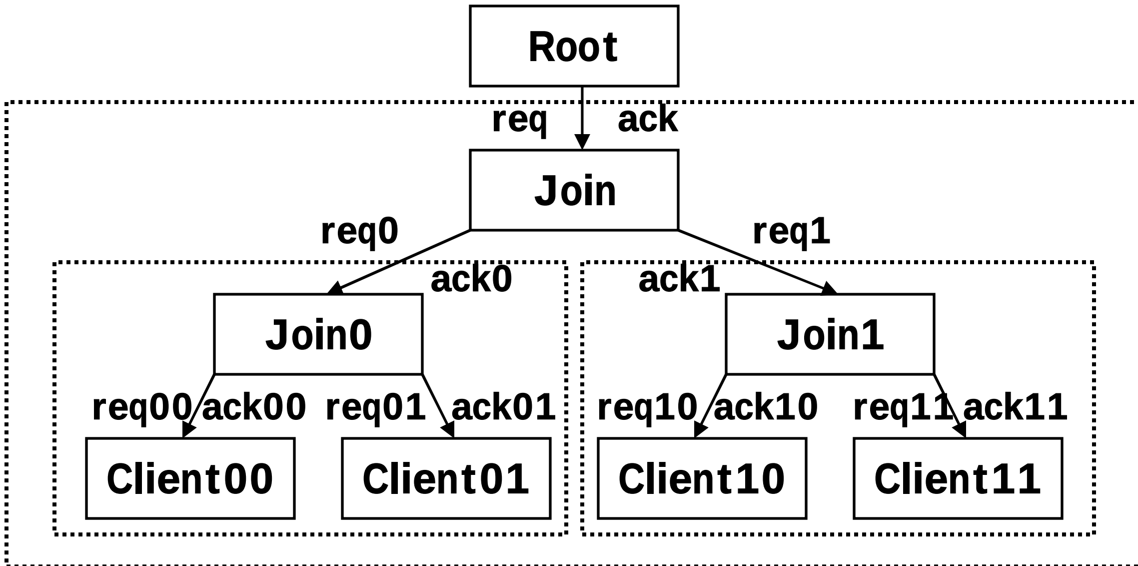
  that "E satisfies ϕ" reduces to "E' satisfies ϕ"
- Basis for computation of E':

  weak simulation preorder and what is visible.
- Search algorithm is used to solve "E' satisfies ϕ"

Symbolic Search Algorithm

- **Problem:**
  - How to search an expression with nested applications of next and parallel composition?
- **Goals:**
  - On-the-fly: states should be explored on demand
  - Avoid precomputing transitive closures
  - Store only states
  - Early detection of violation of requirements

Parity Computer Example



$$E' = Root||NEXT\left[Join||\begin{matrix}NEXT(Join0||Client00||Client01)\\NEXT(Join1||Client10||Client11)\end{matrix}\right]$$

Space Comparison (MDD nodes)

PPP

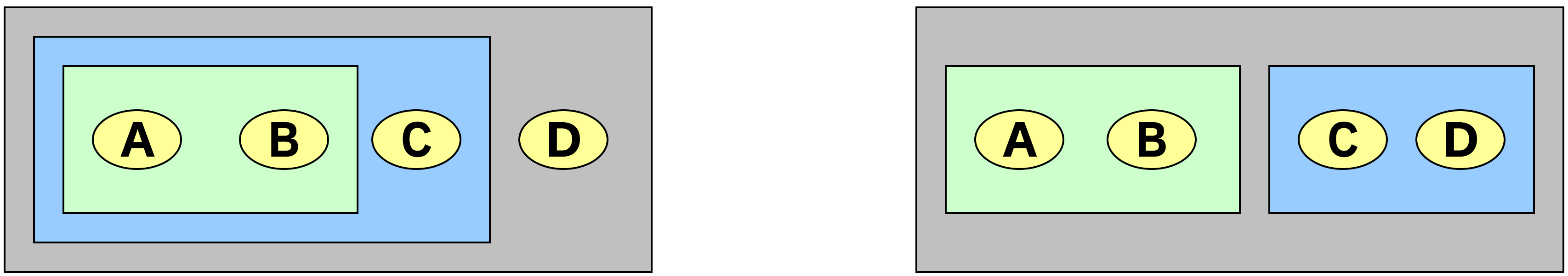
PPP	H.R.	IWLS
No sift	1,007,719	1,171,598
sift	186,589	166,320

DME

	4	5	6	7
H.R.	3804	7200	8971	9516
IWLS	3452	12133	25536	15725

Automatic Hierarchical Partitioning

For a set of processes, which architectural hierarchy is “better”?



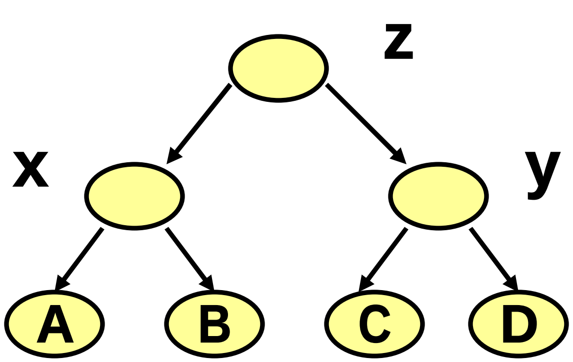
Influences the order of composition and hiding

- Relevant for compositional minimization
- Affects performance of hierarchical reduction

Optimization Problem

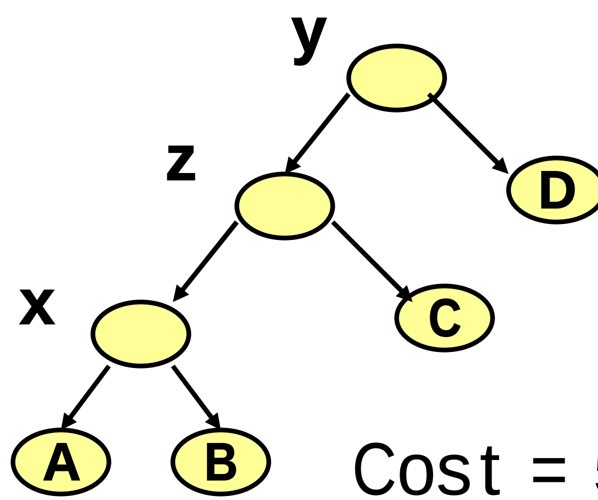
Processes as vertices: { A, B, C, D}

Variables as hyperedges: x:{A,B}   y:{C,D}   z:{A,B,C}



Cost = 4

x:1, y:1, z:2



Cost = 5

x:1, y:3, z:1

Hierarchical Partitioning

- Input: Hypergraph (V,E)
  - V: set of processes
  - Each edge corresponds to a variable, and is a subset of V (processes that access it)
- Output: Tree T over V (hierarchical partition)
- Cost of T: sum of heights of all edges
  - Ht of e: ht of lowest node where e is visible
- Goal: Optimize cost of T
- Problem is NP-hard
- Greedy heuristic (implementation + experiments)

Talk Outline

- ✓ Motivation
- ✓ Refinement check as Reachability
- ✓ Assume-Guarantee Reasoning
- ✓ Hierarchical Reduction
- ❑ Case studies

Refinement Verification

- **Goal:** Given two models Imp and Spec, verify that Imp refines Spec
- **Methodology:**

  Step 1: Using compositional rules, generate simpler subgoals

  Step 2: For each subgoal I < S, introduce a witness module W, and reduce the check to reachability analysis of I || W

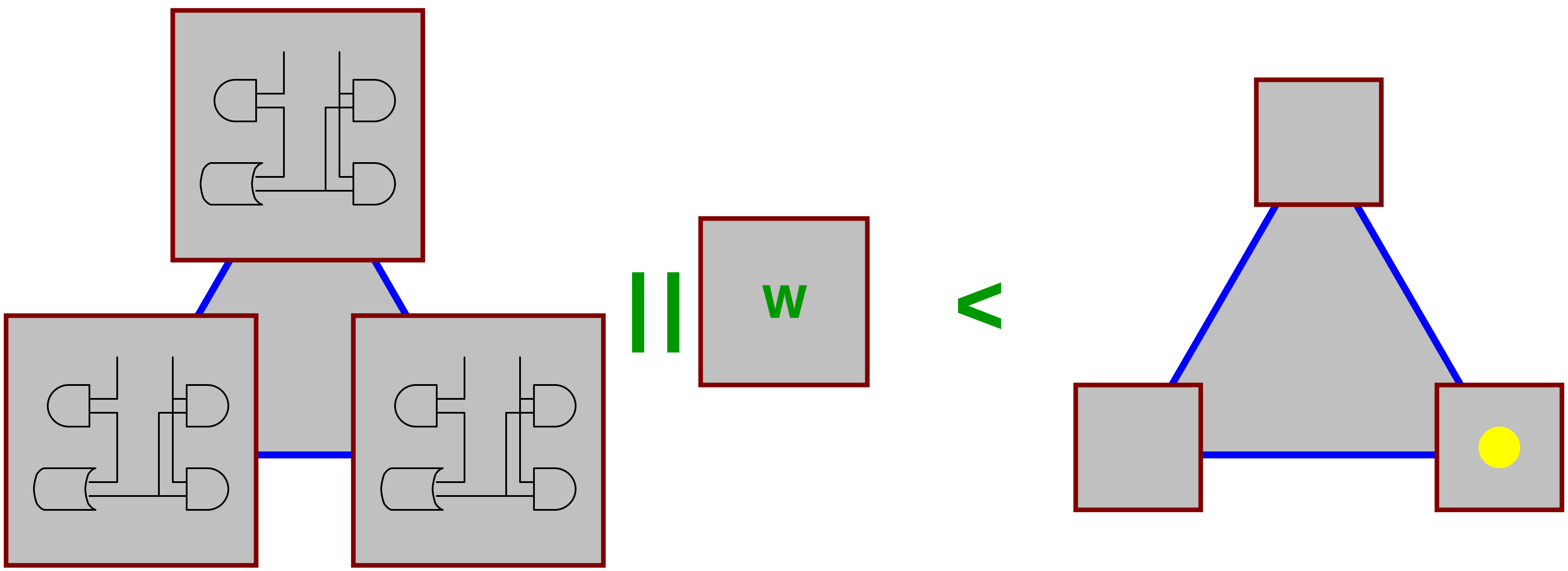
  Step 3: Apply an efficient reachability check

DME Example

- High-level description:
  - A virtual token is passed around a ring of cells.
  - Any cell which gets the token has the right to access the critical section.
  - A cell asks its right neighbor for the token.
  - A cell passes the token to the left when done.
- Low-level description:
  - The implementation is built on logic gates.
  - No virtual token is defined in the implementation.

DME Refinement

Automatic witness construction works



Point-to-Point Protocol

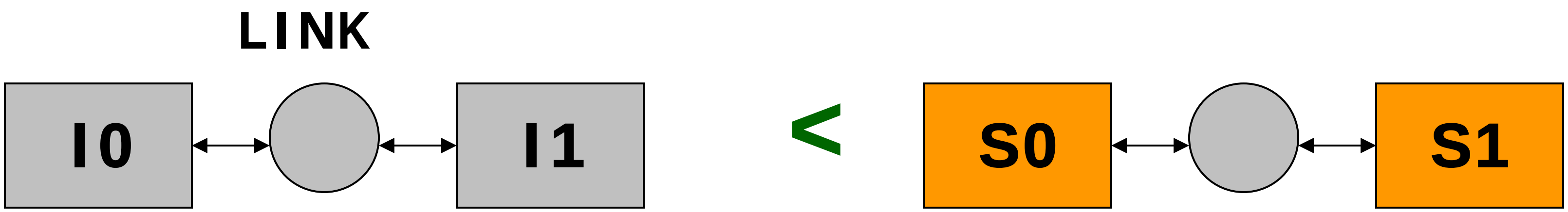
- Popular networking protocol for establishing connections remotely
- Goal: To verify the actual implementation
- Specification: RFC 1661 (standard)
  – Specified in tabular format
- Implementation: ppp version 2.4.0
  – available in various Linux distributions
  – C code

PPP Verification

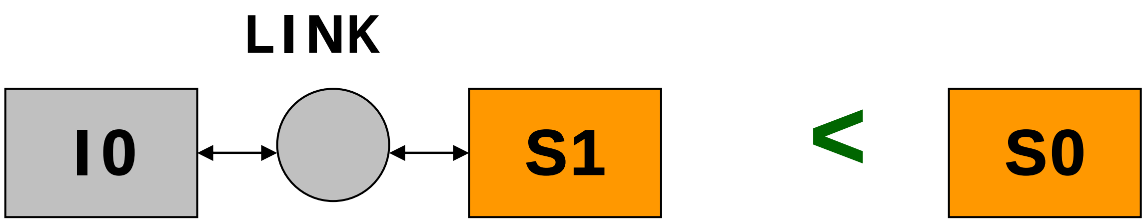
- Focus on option negotiation aspect of protocol
- Manually constructed module I from C-code
- Manually translated RFC spec to module S
- Goal: To verify I < S
- Result: Discovered an inconsistency in the code wrt specification

Assume Guarantee Reasoning

GOAL:



REDUCES TO



More Case Studies

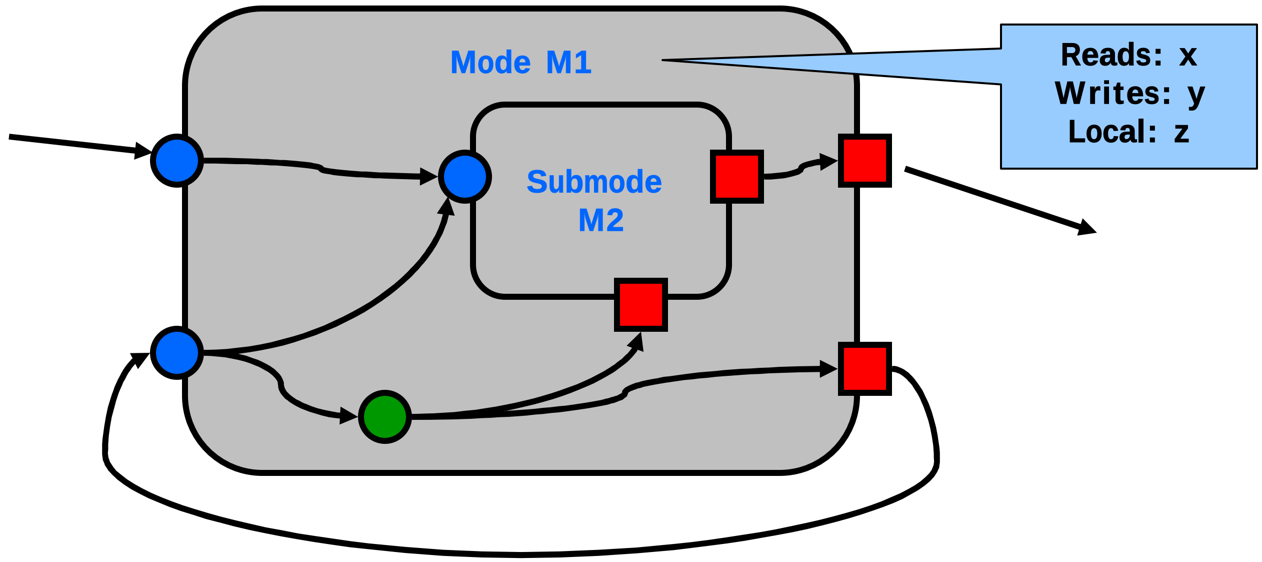
- DHCP: Dynamic host configuration protocol for mobile networks
  - Specification: RFC 2131
  - Implementation: dhcp version 2.0 patch 5
- Traditional examples in refinement setting
  - DME
  - Leader election
  - Tree-structured req-ack template
- Hierarchical reduction can be beneficial

References

- Mocha: A model checker that exploits design structure (ICSE'01)
- Automatic refinement checking for asynchronous processes (A,Grosu,Wang, FMCAD'00)
- Verifying network protocol implementations by symbolic refinement checking (A,Wang, CAV'01)
- Heuristics for hierarchical partitioning (A, Moller, CHARME'01)

Analysis of hierarchical state machines

Modes:



- ❑ **Transition relation is indexed by control points**
  - generalization of conjunctively partitioned bdds,
- ❑ **Transition type exploited**
  - for early quantification in the symbolic search,
- ❑ **Reached state space indexed by control points**
  - pool of variables is not global,