Formal Verification of Mobile Network Protocols

Paolo Milazzo

Dipartimento di Informatica, Università di Pisa, Italy
milazzo@di.unipi.it

Pisa – April 26, 2005

Introduction

Design validation

- ensuring the correctness of systems at design time
- simulation

Complex systems

- digital circuits
- communication protocols
- software

Formal verification

- exhaustive exploration of all possible behaviours

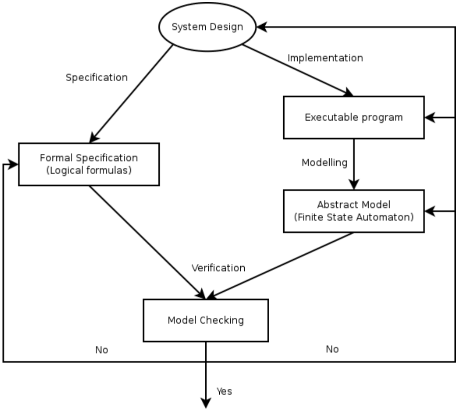
Formal verification techniques

- model checking
- theorem proving
- ...

Model checking

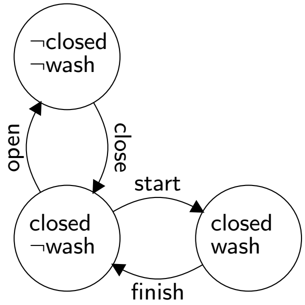
- fully automatic
- systems modelled by finite state automata (Kripke structures)
- specifications given as logical formulas
- model checking algorithms
  - return *true* if the system satisfies the specification
  - give a counterexample otherwise

The Process of Model Checking



Modelling Systems: Kripke Structures

- Kripke structures are finite state automata in which states are labelled with atomic propositions
- Example: a washing machine



variables:
boolean closed, wash;

initial state:
closed = wash = false;

transitions:
[closed=false] → closed = true;
[closed=true and wash=false] → wash = true;
[closed=true and wash=false] → closed = false;
[wash=true] → wash = false;

Formal Specification

A specification is a set of properties that the system must satisfy

Temporal logics can assert how the behavior of a system must evolve over time

- LTL (Linear Temporal Logic)
- CTL, CTL* (Computational Tree Logics)

Examples of properties

- an execution exists such that p is always true
- for all the possible executions, p is eventually true

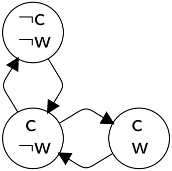
LTL – Linear Temporal Logic

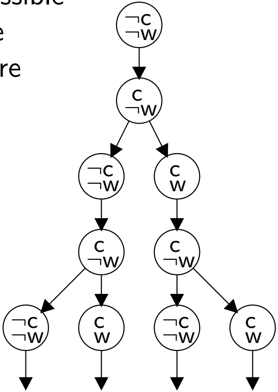
Path Quantifiers	**Path Operators**
$\mathbf{A}f$ ("always")	$\mathbf{X}f$ ("next time")
$\mathbf{E}f$ ("exists")	$\mathbf{F}f$ ("in the future")
	$\mathbf{G}f$ ("globally")
	$f_1\ \mathbf{U} f_2$ ("until")

- Atomic propositions are properties of a state (tests on the values of variables)
- The standard operators $\vee, \wedge, \neg, \rightarrow$ are included.

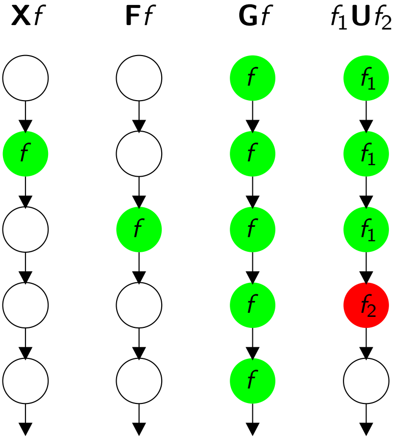
LTL – Linear Temporal Logic

- The (infinite) tree of all the possible paths (computation tree) is the unfolding of the Kripke structure
- **A** and **E** quantify over paths

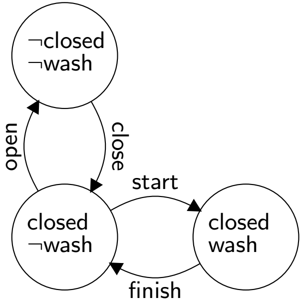




LTL – Linear Temporal Logic



Examples



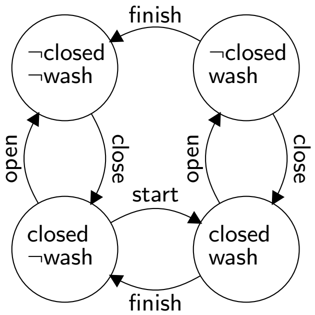
$\mathbf{E}(\mathbf{F}\ \text{wash})$ $\surd$

$\mathbf{A}(\mathbf{G}\ (\text{wash} \rightarrow \text{closed}))$ $\surd$

$\mathbf{A}(\mathbf{G}\ (\text{wash} \rightarrow \mathbf{X}\ \neg\ \text{wash}))$ $\surd$

$\mathbf{A}(\neg\ \text{wash}\ \mathbf{U}\ (\text{closed} \wedge \text{wash})$ $\surd$

Examples



$\mathbf{E}(\mathbf{F}\ \mathrm{wash})$ $\surd$

$\mathbf{A}(\mathbf{G}\ (\mathrm{wash} \rightarrow \mathrm{closed}))$ $\times$

$\mathbf{A}(\mathbf{G}\ (\mathrm{wash} \rightarrow \mathbf{X}\ \neg\ \mathrm{wash}))$ $\times$

$\mathbf{A}(\neg\ \mathrm{wash}\ \mathbf{U}\ (\mathrm{closed} \wedge \mathrm{wash})$ $\surd$

Counterexample for both the false properties

- close,start,open,$\times$

Model Checking Algorithms

Several techniques for verifying properties exist

- the first algorithms used an explicit representation of the Kripke structure, and visited it
- many other approaches use automata for specifications as well as for implementations
  - automata can be sinthesized from logical formulas
  - the system $\mathcal{A}$ satisfies the specification $\mathcal{S}$ when $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{S})$, that is when $\mathcal{L}(\mathcal{A}) \cap \overline{\mathcal{L}(\mathcal{S})} = \varnothing$
- other approaches use observational equivalence or refinement relations

Verifying Network Protocols

A network is a set of concurrent systems

- Problem: state space (exponential) explosion

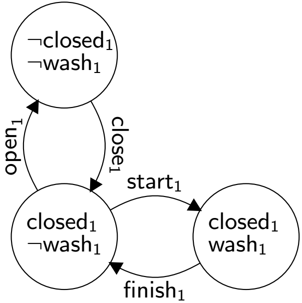
Concurrent systems communicate through message passing

- it can be modelled
- Problem: state space explosion

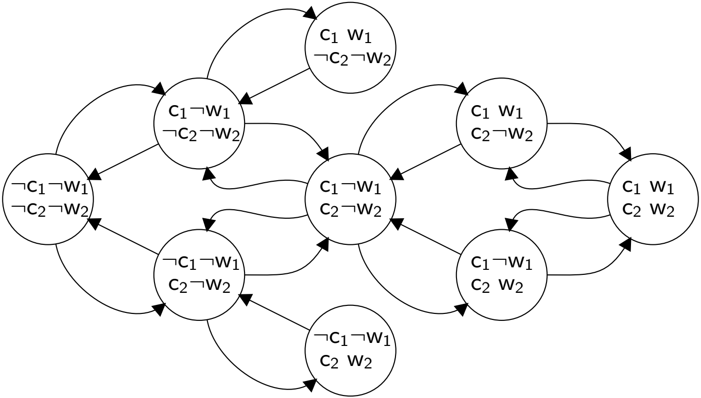
The number of systems involved in a protocol can be unbounded

- Problem: infinite state space

A Network of Washing Machines



A Network of Washing Machines



State space explosion

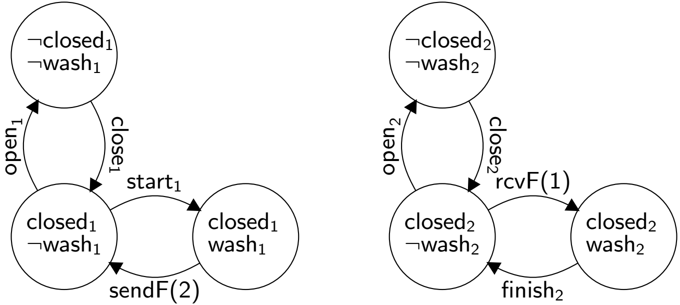
Problem:

- given a network of n finite state systems, the number of global states grows exponentially with n.

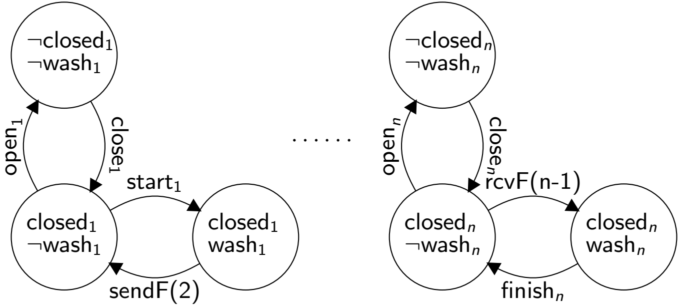
(Partial) solutions:

- symbolic model checking
- on–the–fly model checking
- partial order reduction
- abstractions
- compositional reasoning
- ...

Communicating Washing Machines



Communicating Washing Machines



Unbounded Number of Concurrent Systems

Suppose that we want to prove that machine $i+1$ starts after machine i for every i and for every number n of concurrent systems

$$\forall n \quad \mathbf{A}\left((\neg \mathrm{wash}_1 \mathbf{U} \mathrm{wash}_0) \wedge \cdots \wedge (\neg \mathrm{wash}_n \mathbf{U} \mathrm{wash}_{n-1})\right)$$

We should verify the property once for every possible value of n, but n is unbounded

Solutions (semi–automatic):

- abstractions
- reduce the property to an invariant for pairs of components

Ad Hoc On Demand Distance Vector Protocol (AODV)

AODV:

- routing protocol for mobile ad hoc networks
- frequent changes in topology $\Rightarrow$ routes on demand
- analysis based on the version 2 draft specification
- property to verify: LOOP FREEDOM
- model checking tool: SPIN

Ad Hoc On Demand Distance Vector Protocol (AODV)

Each node maintains 2 counters

- `seqno` incremented at each local topology change
- `broadcast_id` incremented each time the node makes a route–discovery broadcast

and one route record for every currently active destination

- $\mathtt{next}_d$ next node on the route
- $\mathtt{hops}_d$ hops count to d
- $\mathtt{seqno}_d$ last known sequence number of the destination
- $\mathtt{lifetime}_d$ remaining time before route expiration

Ad Hoc On Demand Distance Vector Protocol (AODV)

When a node wants to communicate with a destination d, it broadcast a route request message (RREQ).

RREQ(hops_to_src , broadcast_id , d, seqno , s, src_seq_no)

- hops_to_src hops to the initiating node s
- broadcast_id broadcast id of the initiating node s (if a node receives two messages from the same source and the same broadcast id, the second message is discarded)
- seqno least accepted sequence number for the route
- src_seq_no sequence number of the initiating node s

Ad Hoc On Demand Distance Vector Protocol (AODV)

When a node t receives an RREQ

- if t has not a fresh enough route to d, it rebroadcast the RREQ with incremented hops_to_src field
- if t has a fresh enough route to d, it replies with

  $$\mathtt{RREP}(\mathtt{hops}_d,\ d,\ \mathtt{seqno}_d,\ \mathtt{lifetime}_d)$$

  where $\mathtt{hops}_d$, $\mathtt{seqno}_d$ and $\mathtt{lifetime}_d$ are the corresponding attributes of t's route to d
- if $t = d$, it replies with

  $$\mathtt{RREP}(0,\ d,\ \mathtt{big\_seq\_no}\ ,\ \mathtt{MY\_ROUTE\_TIMEOUT}\ )$$

  where big_seq_no is max(seqno , seqno in RREQ) and MY_ROUTE_TIMEOUT is the default timeout, locally configured at d

Ad Hoc On Demand Distance Vector Protocol (AODV)

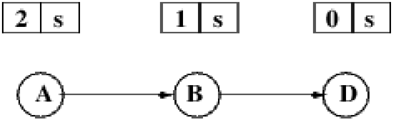
If a route expires (i.e. $\texttt{lifetime}_d$ generates a timeout event)

- it is marked invalid
- $\texttt{hops}_d$ is set to infinity
- $\texttt{lifetime}_d$ is reset BAD_LINK_LIFETIME, which is a locally configured constant

When the BAD_LINK_LIFETIME timer expires, if the route is still invalid it is erased

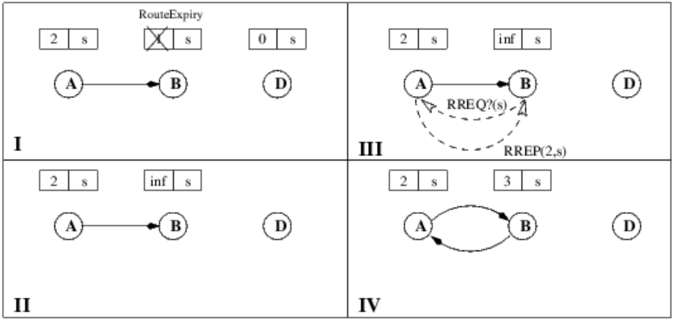
Instance Verification

Simple 3–node scenario

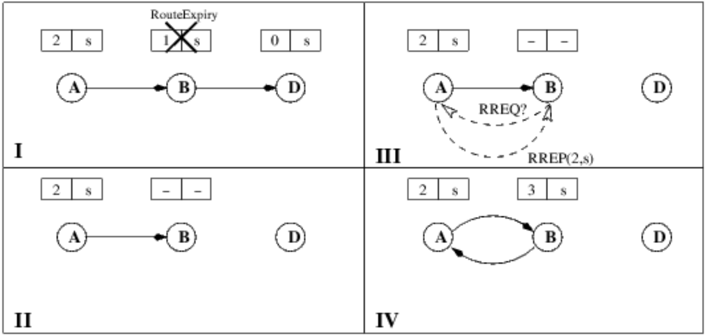


SPIN found a number of looping scenarios !!!

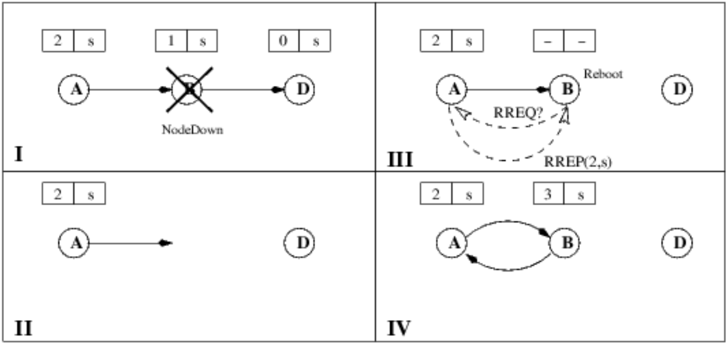
Loop Conditions



Loop Conditions



Loop Conditions



Assumption for loop freedom

Assumption (suggested by the discovered scenarios) under which we prove loop freedom:

- **A1.** When a node discovers and expired or broken route, it increments the sequence number for that route
- **A2.** Nodes never delete routes
- **A3.** Nodes always detect when a neighbor reboots

Theorem 1 (Loop–freedom). Consider an **arbitrary** network of nodes running AODV. If all nodes conform to the assumptions A1–A3, there will be no routing loops formed.

Loop–freedom invariant

The following is an invariant (over time) of the AODV process at a node n, for every destination d

Theorem 2 (Loop–freedom invariant). If $\mathtt{next}_d(n) = n'$, then

1. $\mathtt{seqno}_d(n) \leq \mathtt{seqno}_d(n')$, and
2. $\mathtt{seqno}_d(n) = \mathtt{seqno}_d(n') \Rightarrow \mathtt{hops}_d(n) > \mathtt{hops}_d(n')$

This invariant implies:

1. **Loop Freedom (Theorem 1)** At each hop either the sequence number must increase or the hop–count must decrease
2. **Route Validity** If all the sequence numbers along a path are the same, hop–counts must strictly decrease

Loop–freedom invariant

Theorem 2 can be translated in

- **Lemma 3.** If $t_1 \leq t_2$, then $\texttt{seqno}_d(n)(t_1) \leq \texttt{seqno}_d(n)(t_2)$
- **Lemma 4.** If $t_1 \leq t_2$ and $\texttt{seqno}_d(n)(t_1) = \texttt{seqno}_d(n)(t_2)$, then $\texttt{hops}_d(n)(t_1) \geq \texttt{hops}_d(n)(t_2)$
- **Lemma 5.** If $\texttt{next}_d(n)(t) = n'$, then $\texttt{seqno}_d(n)(t) = \texttt{seqno}_d(n')(lut)$ and $\texttt{hops}_d(n)(t) = 1 + \texttt{hops}_d(n')(lut)$

Where *lut* is the last time before t when $\texttt{next}_d(n)$ changed to n'

Conclusions

The loop–freedom property (Theorem 1) has been reduced to an invariant on pairs on nodes (Theorem 2).

As a consequence an unbounded n–node verification has been reduced to a 2–node verification

The 2–node verification can be performed automatically with SPIN

Loop freedom of AODV under assumptions A1–A3 is verified

References

- D. Obradovic. **Formal Analysis of Routing Protocols**. PhD Thesis, University of Pennsylvania, 2002.
- K. Bhargavan, D. Obradovic and C.A. Gunter. **Formal verification of standards for distance vector routing protocols**. Journal of ACM, 2000.
- E.M. Clarke Jr, O. Grumberg and D.A. Peled. **Model Checking**. MIT Press, 1999.