A Model Checking Case Study

Flooding Time Synchronization Protocol

Ocan Sankur

Univ Rennes, CNRS, Inria

Formal Methods

Development cycle for general-purpose systems

1. Write code
2. Code review, testing
3. Deploy
4. Fix bugs, and provide regular updates

Formal Methods

Development cycle for general-purpose systems

1. Write code
2. Code review, testing
3. Deploy
4. Fix bugs, and provide regular updates

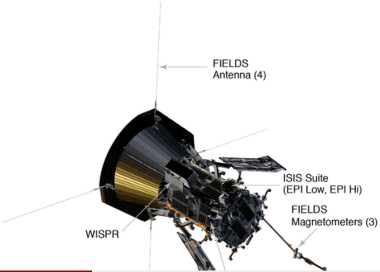
Is step 4 feasible for all systems?

Formal Methods

Development cycle for general-purpose systems

1. Write code
2. Code review, testing
3. Deploy
4. Fix bugs, and provide regular updates

Is step 4 feasible for all systems?
Aerospatial:



Formal Methods

Development cycle for general-purpose systems

1. Write code
2. Code review, testing
3. Deploy
4. Fix bugs, and provide regular updates

Is step 4 feasible for all systems?
Health:

Formal Methods

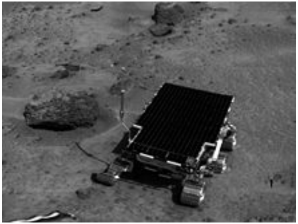
Development cycle for general-purpose systems

1. Write code
2. Code review, testing
3. Deploy
4. Fix bugs, and provide regular updates

Is step 4 feasible for all systems?

Transportation (train networks, autonomous vehicles, airplanes)

Mars Rover



- Developed in 3 years, for about 150 million dollars
- The rover landed successfully in Mars but had several total system resets
- Loss of mission time
- Engineers were able to fix the bug by an update!

Ariane 5

- An Ariane rocket launched in 1996 has exploded shortly after the launch
- Previous missions were successful
- 370 million euros

Ariane 5

- An Ariane rocket launched in 1996 has exploded shortly after the launch
- Previous missions were successful
- 370 million euros

Other famous bugs: Hardware bugs, Toyota unintended acceleration bug(?), infusion pumps, train systems

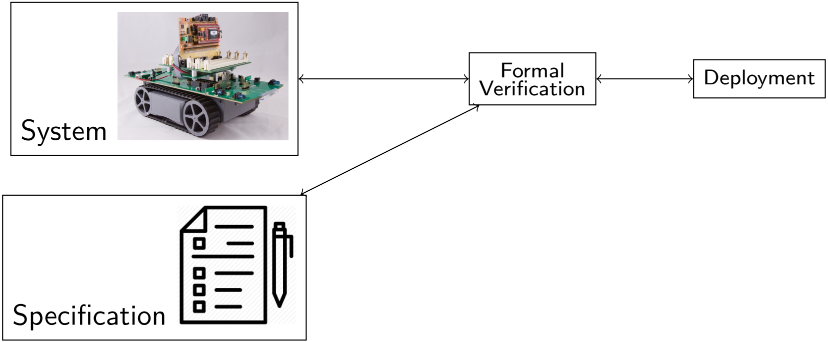
Formal Verification

Formal verification

Input: System or a model

Output: Check whether *all* possible behaviors are correct

$\sim$ Exhaustive testing



Application Domains

- Hardware industry
- Embedded systems
- Communication systems
- Transportation (Automative, aerospatial, trains)

Critical areas such as aerospatial industry require certification:
A rigorous development methodology including formal verification must be followed

Application Domains

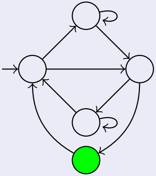
- Hardware industry
- Embedded systems
- Communication systems
- Transportation (Automative, aerospatial, trains)

Critical areas such as aerospatial industry require certification:
A rigorous development methodology including formal verification must be followed

More and more used in non-critical software development!

Model Checking

Model-checking



$\stackrel{?}{\models}$

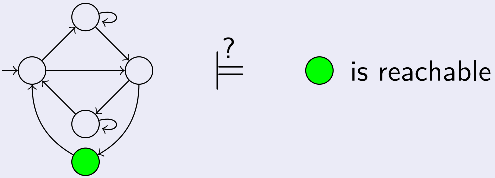
 is reachable

The model is often a transition system: **graph of configurations**

Goal: Check the specification on **all paths** in this graph

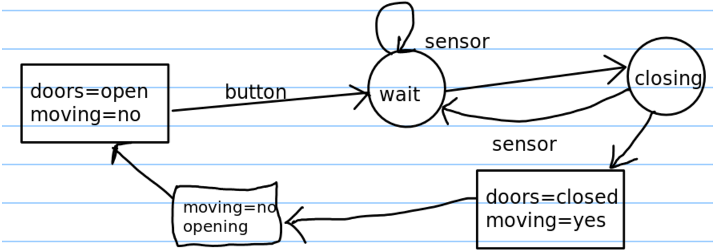
Model Checking

Model-checking



The model is often a transition system: **graph of configurations**

Goal: Check the specification on **all paths** in this graph



Model Checking

Theoretical Definition

Given a transition system T, and specification ϕ, check whether the executions of T satisfy ϕ.

When T is an automaton, and ϕ safety condition, model checking is a simple graph traversal.

Model Checking

Theoretical Definition

Given a transition system T, and specification ϕ, check whether the executions of T satisfy ϕ.

When T is an automaton, and ϕ safety condition, model checking is a simple graph traversal.

However, in practice, state-space explosion due to

- Use of Boolean or discrete variables (think of a 64-bit integer variable)
- Parallel composition of components
- Time constraints, ...

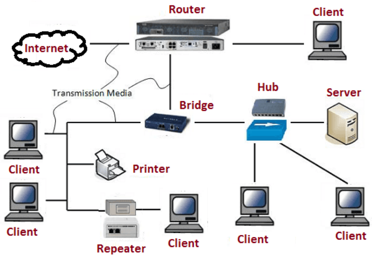
Model Checking - 2

Model checking is about controlling the state space explosion.
Each algorithm and application must justify how this is handled. E.g.

- Choice of an efficient state-space representation
- Reduction of the state space: abstractions
- ...

Case Study: Clock Synchronization Protocol

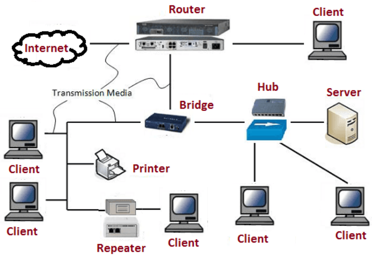
- **Clocks** on all electronics are not identical and sensitive to temperature
- Algorithms are used to synchronize clocks over networks



▶ This makes sure machines agree on a common time: collaborative platforms, social networks, wireless sensor networks

Case Study: Clock Synchronization Protocol

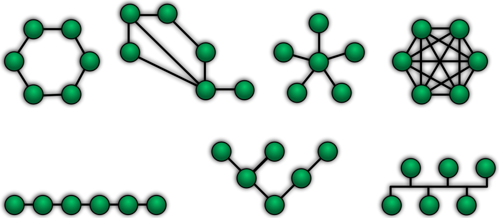
- **Clocks** on all electronics are not identical and sensitive to temperature
- Algorithms are used to synchronize clocks over networks



▶ This makes sure machines agree on a common time: collaborative platforms, social networks, wireless sensor networks

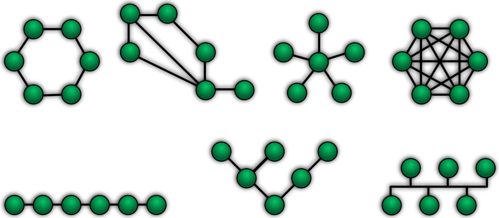
Parameterized Model Checking

Goal: Model check a given protocol on **all possible** network topologies



Parameterized Model Checking

Goal: Model check a given protocol on **all possible** network topologies



For **all** number of participants, and **all** topologies, check **all** executions

Flooding-Time Synchronization Protocol (FTSP)

Leader Election

From all possible configurations a **unique leader machine** is eventually elected

FTSP

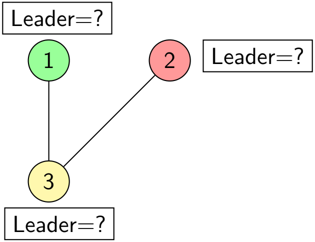
- Maintains a unique leader, recovers in case of link/node failures
- Smoothly synchronizes the clocks over the network with the clock of the leader

We consider the leader election part of FTSP: Verify that a unique leader is eventually elected

Flooding-Time Synchronization Protocol (FTSP)

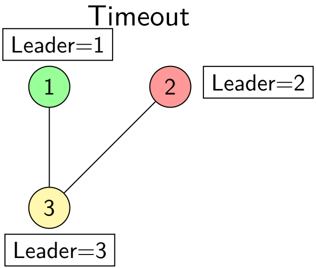
– Nodes have unique identifiers but execute the same program
– Each node wakes up with period P and sends a message to its neighbors
– The network eventually elects the node with the least ID as the **leader**
– Fault tolerant: any node that hasn't heard from the leader for a while timeouts and declares itself leader

Initially



Flooding-Time Synchronization Protocol (FTSP)

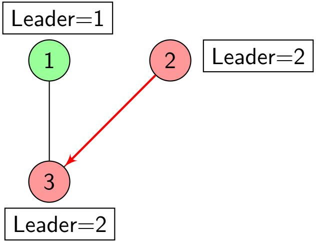
– Nodes have unique identifiers but execute the same program
– Each node wakes up with period P and sends a message to its neighbors
– The network eventually elects the node with the least ID as the **leader**
– Fault tolerant: any node that hasn't heard from the leader for a while timeouts and declares itself leader



Flooding-Time Synchronization Protocol (FTSP)

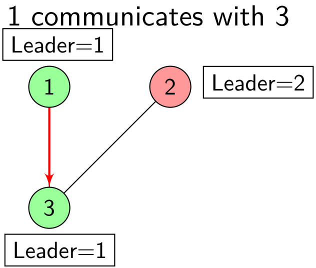
– Nodes have unique identifiers but execute the same program
– Each node wakes up with period P and sends a message to its neighbors
– The network eventually elects the node with the least ID as the **leader**
– Fault tolerant: any node that hasn't heard from the leader for a while timeouts and declares itself leader

2 communicates with 3



Flooding-Time Synchronization Protocol (FTSP)

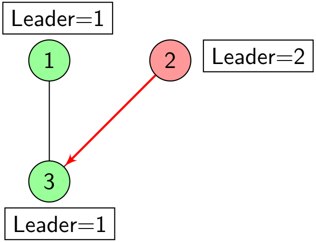
– Nodes have unique identifiers but execute the same program
– Each node wakes up with period P and sends a message to its neighbors
– The network eventually elects the node with the least ID as the **leader**
– Fault tolerant: any node that hasn't heard from the leader for a while timeouts and declares itself leader



Flooding-Time Synchronization Protocol (FTSP)

– Nodes have unique identifiers but execute the same program
– Each node wakes up with period P and sends a message to its neighbors
– The network eventually elects the node with the least ID as the **leader**
– Fault tolerant: any node that hasn't heard from the leader for a while timeouts and declares itself leader

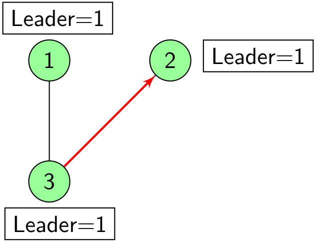
2 communicates with 3: Ignored!



Flooding-Time Synchronization Protocol (FTSP)

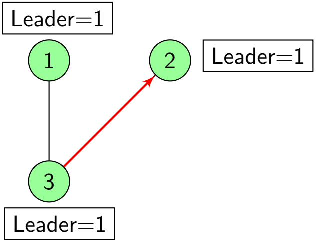
– Nodes have unique identifiers but execute the same program
– Each node wakes up with period P and sends a message to its neighbors
– The network eventually elects the node with the least ID as the **leader**
– Fault tolerant: any node that hasn't heard from the leader for a while timeouts and declares itself leader

3 communicates with 2: Convergence!



Flooding-Time Synchronization Protocol (FTSP)

– Nodes have unique identifiers but execute the same program
– Each node wakes up with period P and sends a message to its neighbors
– The network eventually elects the node with the least ID as the **leader**
– Fault tolerant: any node that hasn't heard from the leader for a while timeouts and declares itself leader



Message content: (leader ID ℓ, sequence number s)

Process ignores message if its leader is $< \ell$ or if its leader is ℓ but has already seen a message with $s' \geq s$.

Simplified code

```
#define TIMEOUT 8
extern byte ID;
byte heartBeats;
byte myleader;
byte myseq;
chan out;

void receive (byte li, byte si) {
  if(li < myleader || (li == myleader && si > myseq)){
     myleader = li;
     myseq = si;
     heartBeats = 0;
     /* Add time sample */
  }
}

void activated () {
  if(heartBeats >= TIMEOUT){
    myleader = ID;
    myseq = 0;
    heartBeats = 0;
  } else heartBeats++;
  o!(myleader, myseq);
  if(myleader == ID){ myseq++; }
}
```

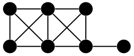
Our model: Derived from the TinyOS implementation

Omitted here but available in the model: sample threshold, ignore period

Previous Verification Results

Previous work: Model checking that a unique leader is eventually elected (Spin, FDR3, Uppaal).

- A few fixed topologies. The largest verified topology (in 1 hour):



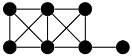
- Perfectly synchronized clocks, no clock deviations!
- Synchronous message broadcast: when a process sends a message, all other nodes stop and listen

Kusy, Abdelwahed 2006, McInnes 2009, Tan, Zhao, Wang 2010

Previous Verification Results

Previous work: Model checking that a unique leader is eventually elected (Spin, FDR3, Uppaal).

- A few fixed topologies. The largest verified topology (in 1 hour):



- Perfectly synchronized clocks, no clock deviations!
- Synchronous message broadcast: when a process sends a message, all other nodes stop and listen

Kusy, Abdelwahed 2006, McInnes 2009, Tan, Zhao, Wang 2010

Present work

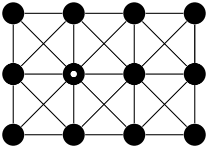
- Arbitrary network topology within given diameter K (we will go up to $K = 13$)
- Deviating clocks
- Synchronous or asynchronous broadcast

Overview of the Talk

1. FTSP
2. Previous model checking attempts
3. Abstraction Idea 1: Anonymization
4. Abstraction Idea 2: Network abstraction
5. Clock Deviations
6. Results
7. Incremental Proof and Custom Semi-Algorithm
8. Abstraction Refinement

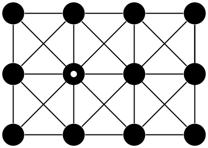
Main Abstraction Idea for Parameterized Verification

How the leader is propagated:



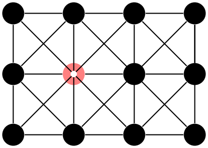
Main Abstraction Idea for Parameterized Verification

How the leader is propagated:



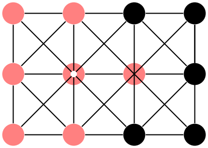
Main Abstraction Idea for Parameterized Verification

How the leader is propagated:



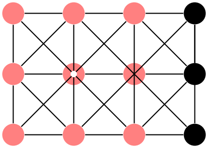
Main Abstraction Idea for Parameterized Verification

How the leader is propagated:



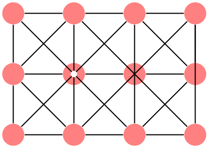
Main Abstraction Idea for Parameterized Verification

How the leader is propagated:



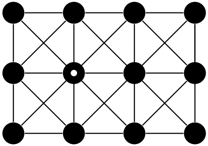
Main Abstraction Idea for Parameterized Verification

How the leader is propagated:



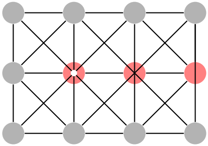
Main Abstraction Idea for Parameterized Verification

Abstracting the network:



Main Abstraction Idea for Parameterized Verification

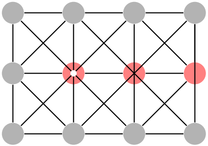
Abstracting the network:



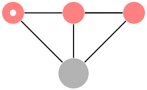
Pick a shortest path from the future leader to some node

Main Abstraction Idea for Parameterized Verification

Abstracting the network:



$\sqsubseteq$



On the right, the gray node can send any message (ℓ, s) to anyone, as long as ℓ is different than the IDs of the red processes.

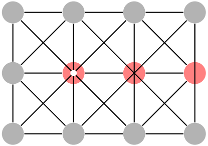
If the least ID is 1, we check the following property in the small model:

$$\Diamond\Box(\texttt{P1.myleader} = 1 \ \& \ \texttt{P2.myleader} = 1 \ \& \ \texttt{P3.myleader} = 1).$$

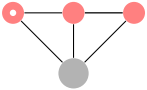
This would imply that the property holds on the left for this **particular topology**

Main Abstraction Idea for Parameterized Verification

Abstracting the network:



$\sqsubseteq$



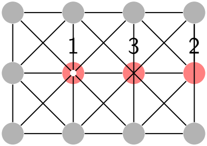
On the right, the gray node can send any message (ℓ, s) to anyone, as long as ℓ is different than the IDs of the red processes.

If the least ID is 1, we check the following property in the small model:

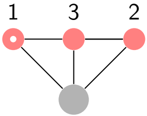
$$\Diamond\Box(\texttt{P1.myleader} = 1 \; \& \; \texttt{P2.myleader} = 1 \; \& \; \texttt{P3.myleader} = 1).$$

Generalization: We want the abstract model to be the same for all possible choices of the red paths of given length D

Main Abstraction Idea for Parameterized Verification



$\sqsubseteq$



On the right, the gray node can send any message (ℓ, s) to anyone, as long as ℓ is different than the IDs of the red processes.

If the least ID is 1, we check the following property in the small model:

$$\Diamond\Box(\texttt{P1.myleader} = 1 \ \& \ \texttt{P2.myleader} = 1 \ \& \ \texttt{P3.myleader} = 1).$$

Problem: The verification result is valid for a given path and a given configuration of the identifiers

1. Anonymization through data abstraction

Goal: Map "unbounded" variables to finite domains

Let `FLEAD` denote the identifier of the *future leader*.

Let `NONFLEAD` be a symbol to represent any other id.

Node Identifiers

Map all identifier variables i.e. `myleader` and `li` to $\{\mathtt{FLEAD}, \mathtt{NONFLEAD}\}$.

Some expressions and assignments become non-deterministic:

$\rightarrow$ Expression "$\mathtt{li} < \mathtt{myleader}$" becomes:

$$\begin{cases} \mathtt{li} = \mathtt{FLEAD} \wedge \mathtt{myleader} = \mathtt{NONFLEAD} & : \quad \mathtt{true} \\ \mathtt{li} = \mathtt{NONFLEAD} \wedge \mathtt{myleader} = \mathtt{FLEAD} & : \quad \mathtt{false} \\ \mathtt{li} = \mathtt{myleader} = \mathtt{FLEAD} & : \quad \mathtt{false} \\ \text{otherwise} & : \{\mathtt{true}, \mathtt{false}\}. \end{cases}$$

(We also map integer variables to finite domains)

1. Anonymization through data abstraction

Goal: Map "unbounded" variables to finite domains
Let FLEAD denote the identifier of the *future leader*.
Let NONFLEAD be a symbol to represent any other id.

Node Identifiers

Map all identifier variables i.e. myleader and li to $\{\mathtt{FLEAD}, \mathtt{NONFLEAD}\}$.

Some expressions and assignments become non-deterministic:
$\rightarrow$ Expression "$\mathtt{li} < \mathtt{myleader}$" becomes:

$$\begin{cases} \mathtt{li} = \mathtt{FLEAD} \wedge \mathtt{myleader} = \mathtt{NONFLEAD} & : \quad \mathtt{true} \\ \mathtt{li} = \mathtt{NONFLEAD} \wedge \mathtt{myleader} = \mathtt{FLEAD} & : \quad \mathtt{false} \\ \mathtt{li} = \mathtt{myleader} = \mathtt{FLEAD} & : \quad \mathtt{false} \\ \text{otherwise} & : \{\mathtt{true}, \mathtt{false}\}. \end{cases}$$

(We also map integer variables to finite domains)

- The abstract protocol is an over-approximation

1. Anonymization through data abstraction

Goal: Map "unbounded" variables to finite domains

Let FLEAD denote the identifier of the *future leader*.

Let NONFLEAD be a symbol to represent any other id.

Node Identifiers

Map all identifier variables i.e. myleader and li to $\{\texttt{FLEAD}, \texttt{NONFLEAD}\}$.

Some expressions and assignments become non-deterministic:

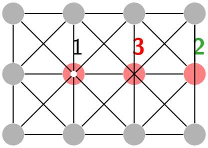
$\rightarrow$ Expression "$\texttt{li} < \texttt{myleader}$" becomes:

$$\begin{cases} \texttt{li} = \texttt{FLEAD} \wedge \texttt{myleader} = \texttt{NONFLEAD} & : \quad \texttt{true} \\ \texttt{li} = \texttt{NONFLEAD} \wedge \texttt{myleader} = \texttt{FLEAD} & : \quad \texttt{false} \\ \texttt{li} = \texttt{myleader} = \texttt{FLEAD} & : \quad \texttt{false} \\ \text{otherwise} & : \{\texttt{true}, \texttt{false}\}. \end{cases}$$

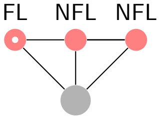
(We also map integer variables to finite domains)

- The abstract protocol is an over-approximation
- The protocol does not depend on precise identifiers but only on FLEAD

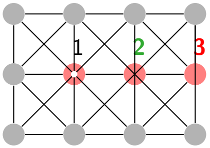
Back to Shortest-Path Abstraction



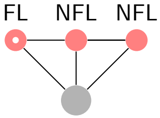
$\sqsubseteq$



Back to Shortest-Path Abstraction



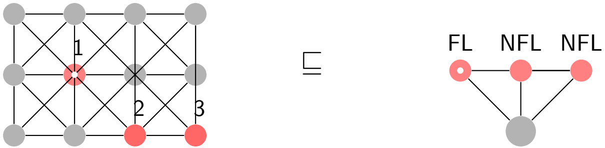
$\sqsubseteq$



The abstraction is identical:

- For any configuration of the identifiers

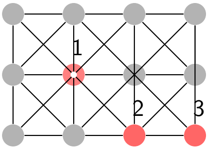
Back to Shortest-Path Abstraction



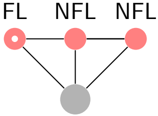
The abstraction is identical:

- For any configuration of the identifiers

Back to Shortest-Path Abstraction



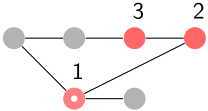
$\sqsubseteq$



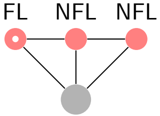
The abstraction is identical:

- For any configuration of the identifiers
- For any chosen path

Back to Shortest-Path Abstraction



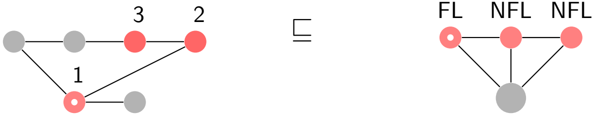
$\sqsubseteq$



The abstraction is identical:

- For any configuration of the identifiers
- For any chosen path

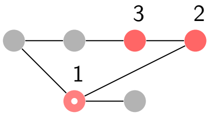
Back to Shortest-Path Abstraction



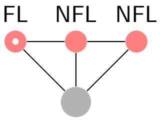
The abstraction is identical:

- For any configuration of the identifiers
- For any chosen path in **any** graph

Back to Shortest-Path Abstraction



$\sqsubseteq$



The abstraction is identical:

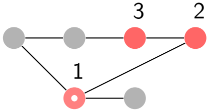
- For any configuration of the identifiers
- For any chosen path in **any** graph

Fix parameter D as the max. distance from the future leader.

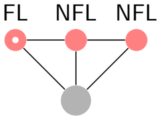
Abstract model $\mathcal{A}_D$ over-approximates the nodes within distance of D from future leader in all networks

Then, $\mathcal{A}_D \models \phi$ means ϕ holds at all nodes at distance $\leq D$.

Back to Shortest-Path Abstraction



$\sqsubseteq$



The abstraction is identical:

- For any configuration of the identifiers
- For any chosen path in **any** graph

Fix parameter D as the max. distance from the future leader.

Abstract model $\mathcal{A}_D$ over-approximates the nodes within distance of D from future leader in all networks

Then, $\mathcal{A}_D \models \phi$ means ϕ holds at all nodes at distance $\leq D$.

Main idea but needs several other tricks to work

Asynchrony and Clock Deviations

Distributed system: we need to define a **scheduler**

– Each process is activated with “identical” period $P \pm \epsilon$

– **Not** synchronous, **not** completely asynchronous neither

Asynchrony and Clock Deviations

Distributed system: we need to define a **scheduler**

– Each process is activated with "identical" period $P \pm \epsilon$

– **Not** synchronous, **not** completely asynchronous neither

Approximate Synchrony (Caspi 2000, Desai et al. CAV2015)

Fix parameter Δ. Let $N_i(t)$ denote the number of times process i has been activated at time t.

Scheduler: Allow all interleavings between processes so that $|N_i(t) - N_j(t)| \leq \Delta$ for all i, j, t.

Asynchrony and Clock Deviations

Distributed system: we need to define a **scheduler**

– Each process is activated with "identical" period $P \pm \epsilon$

– **Not** synchronous, **not** completely asynchronous neither

Approximate Synchrony (Caspi 2000, Desai et al. CAV2015)

Fix parameter Δ. Let $N_i(t)$ denote the number of times process i has been activated at time t.

Scheduler: Allow all interleavings between processes so that $|N_i(t) - N_j(t)| \leq \Delta$ for all i, j, t.

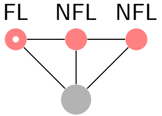
Given P, ϵ, Δ, there exists $N = f(P, \epsilon, \Delta)$ such that the above scheduler over-approximates system behaviors given by deviating clocks.

Summary of Abstractions and Specification

1. Unbounded variables and identifier variables

   $\rightarrow$ Nodes `NONFLEAD` become anonymous

2. Shortest-Path Abstraction:



3. Approximately Synchronous Scheduler

   $\Delta = 1, P \in [29.7, 30.3]$ for $N = 110$ steps

4. Specification: Given D, find N such that

$$\mathcal{A}_D \models \mathsf{F}_{\leq N}\mathsf{G}(\bigwedge_{i=1}^{D} \texttt{Pi.myleader} = \texttt{FLEAD})$$

Experimental Results for FTSP

Tool: A custom algorithm implemented within **NuSMV**
`https://github.com/osankur/nusmv/tree/ftsp`

(Other tools we tried: Spin, CMurphi, ITS-tools)

	synchronous		asynchronous	
D	*N*	time	*N*	time
1	8	0s	8	0s
2	14	1s	14	1s
3	23	1s	25	28s
4	35	3s	39	130s
5	54	16s	63	65mins
6	67	76s	TO	TO
7	107	13mins	TO	TO

D: Max distance from FLEAD
N: Number of steps to convergence

E.g. 2D grids with 169 nodes, or 3D grids in 2197 nodes.
– Clock rates within 1 ± 10^{-2} (period $[29.7, 30.3]$).

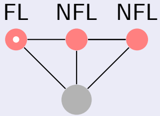
Error recovery Our models are initialized at arbitrary states: in case of any failure, the protocol recovers in N steps

Next: Incremental verification technique + a custom algorithm

Optimization: Incremental Verification Strategy

Observation

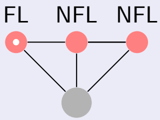
The abstraction $\mathcal{A}_D$ proves the property for all nodes within D of the future leader in all network topologies.



Optimization: Incremental Verification Strategy

Observation

The abstraction $\mathcal{A}_D$ proves the property for all nodes within D of the future leader in all network topologies.



D-radius: the nodes within a distance of D from the future leader.

Incremental Verification For Increasing D

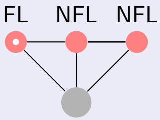
For all $D = 1\ldots$,

- Initialize the system $\mathcal{A}_D$ **nondeterministically** at states where the $(D-1)$-radius already satisfies $\wedge_{i \leq D-1} \texttt{Pi.myleader = FLEAD}$.
- Model check $\mathcal{A}_D \models \mathsf{F}_{\leq N_D}\mathsf{G}(\texttt{PD.myleader = FLEAD})$.

Optimization: Incremental Verification Strategy

Observation

The abstraction $\mathcal{A}_D$ proves the property for all nodes within D of the future leader in all network topologies.



D-radius: the nodes within a distance of D from the future leader.

Incremental Verification For Increasing D

For all $D = 1...$,

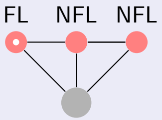
- Initialize the system $\mathcal{A}_D$ **nondeterministically** at states where the $(D-1)$-radius already satisfies $\wedge_{i \leq D-1} \texttt{Pi.myleader = FLEAD}$.
- Model check $\mathcal{A}_D \models \mathsf{F}_{\leq N_D}\mathsf{G}(\texttt{PD.myleader = FLEAD})$.

Substantial gain in time and memory: processes $1, \ldots, D-1$ are simplified since they were proven to satisfy the spec forever

Optimization: Incremental Verification Strategy

Observation

The abstraction $\mathcal{A}_D$ proves the property for all nodes within D of the future leader in all network topologies.



D-radius: the nodes within a distance of D from the future leader.

Incremental Verification For Increasing D

For all $D = 1...,$

- Initialize the system $\mathcal{A}_D$ **nondeterministically** at states where the $(D-1)$-radius already satisfies $\wedge_{i \leq D-1}\texttt{Pi.myleader = FLEAD}$.
- Model check $\mathcal{A}_D \models \mathsf{F}_{\leq N_D}\mathsf{G}(\texttt{PD.myleader = FLEAD})$.

Then in $N = N_1 + N_2 + \ldots + N_D$ number of steps, the whole D-radius agree on FLEAD

Optimization: Semi-Algorithm for $\mathtt{F}_{\leq N}\mathtt{G}\phi$

Standard algorithm to check $\mathtt{FG}\phi$

Convert formula to Buchi automaton, forward exploration, keep all seen states to guarantee termination.

– $R_1, R_2, \ldots, R_k$ where R_i are states reachable in i steps

– Stop when $R_k \subseteq \cup_i R_i$, or when an accepting lasso is found

Optimization: Semi-Algorithm for $\mathtt{F}_{\leq N}\mathtt{G}\phi$

Standard algorithm to check $\mathtt{FG}\phi$

Convert formula to Buchi automaton, forward exploration, keep all seen states to guarantee termination.

– $R_1, R_2, \ldots, R_k$ where R_i are states reachable in i steps

– Stop when $R_k \subseteq \cup_i R_i$, or when an accepting lasso is found

– Even the lasso starts after 1000 steps, we keep $R_1, \ldots, R_{1000}$.

– Consumes memory and impairs BDD reordering

– To compute best N, need to run the tool $\log(N)$ times

Optimization: Semi-Algorithm for $\mathtt{F}_{\leq N}\mathtt{G}\phi$

Standard algorithm to check $\mathtt{FG}\phi$

Convert formula to Buchi automaton, forward exploration, keep all seen states to guarantee termination.

– $R_1, R_2, \ldots, R_k$ where R_i are states reachable in i steps

– Stop when $R_k \subseteq \cup_i R_i$, or when an accepting lasso is found

– Even the lasso starts after 1000 steps, we keep $R_1, \ldots, R_{1000}$.

– Consumes memory and impairs BDD reordering

– To compute best N, need to run the tool $\log(N)$ times

Custom Semi-Algorithm

– Start exploring but forget previous states: R_i (delete $R_1, \ldots, R_{i-1}$)

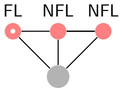
– Whenever $R_i \subseteq \phi$, start remembering $R_i, R_{i+1}, \ldots, R_j$

  – If $R_j \subseteq \cup_{i \leq k \leq j} R_k$, RETURN i

  – If $R_j \not\subseteq \phi$, delete $R_i, \ldots, R_{j-1}$, and continue

Significant performance improvement

Non-Interference Lemma for FTSP



Counterexample to FGϕ

– $S(C_1) = S(C_2) = S(C_3) = \texttt{FL}$, `P1.myseq = P2.myseq = P3.myseq=1`.

– (Some **outside** node sends a message $(\texttt{FLEAD}, 32)$ to P3)

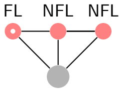
– $S(C_1) = S(C_2) = S(C_3) = \texttt{FL}$, `P1.myseq = P2.myseq = 1, P3.myseq=32`.

– (P3 ignores all messages from the root until its sequence number reaches 32)

$\rightarrow$ P3 timeouts before this happens

– $S(C_1) = S(C_2) = \texttt{FL}$, $S(C_3) = \texttt{NFL}$, `P1.myseq = P2.myseq = 1, P3.myseq=0`.

Non-Interference Lemma for FTSP



Counterexample to FGϕ

– $S(C_1) = S(C_2) = S(C_3) = \texttt{FL}, \texttt{P1.myseq = P2.myseq = P3.myseq=1}$.
– (Some **outside** node sends a message $(\texttt{FLEAD}, 32)$ to P3)
– $S(C_1) = S(C_2) = S(C_3) = \texttt{FL}, \texttt{P1.myseq = P2.myseq = 1, P3.myseq=32}$.
– (P3 ignores all messages from the root until its sequence number reaches 32)
 $\rightarrow$ P3 timeouts before this happens
– $S(C_1) = S(C_2) = \texttt{FL}, S(C_3) = \texttt{NFL}, \texttt{P1.myseq = P2.myseq = 1, P3.myseq=0}$.

Non-interference lemma

$$\psi = \forall i, \texttt{Pi.myleader = FL} \Rightarrow \texttt{Pi.myseq} \leq \texttt{P1.myseq}.$$

Theorem [McMillan 2001, Chou, Mannavan, Park 2004]

If all transitions of the concrete model are strengthened by non-interference lemma ψ, then both the specification ϕ, and the lemma ψ can be model checked in the absraction of the strengthening.

Conclusion

- Few results on parameterized model checking of non-identical non-symmetric systems with arbitrary topologies
- Decidability versus efficiency
- An efficient solution that combines several ideas
- Other protocols whose spec depends on an information being propagated

Next objectives:

- Also prove clock precision bounds under hypotheses on environment conditions
- Extend the theory of abstraction & refinement to probabilistic systems
- Automatize abstractions

Related Works

Parameterized **symmetric** systems: cache coherence protocols

- **Isolating two components** (among K), and applying existential abstraction

  FLASH Cache coherence protocol [McMillan 2001].

Related Works

Parameterized **symmetric** systems: cache coherence protocols

- **Isolating two components** (among K), and applying existential abstraction

  FLASH Cache coherence protocol [McMillan 2001].
- **Counter Abstraction:** Count how many components are at a given state, and abstract as $\{0, 1, \infty\}$

  [Pnueli, Xu, Zuck 2002]

Related Works

Parameterized **symmetric** systems: cache coherence protocols

- **Isolating two components** (among K), and applying existential abstraction

  FLASH Cache coherence protocol [McMillan 2001].
- **Counter Abstraction:** Count how many components are at a given state, and abstract as $\{0, 1, \infty\}$

  [Pnueli, Xu, Zuck 2002]
- **Environment Abstraction**: the isolated components are seen as reference points and can change

  [Clarke, Talupur, Veith 2008]

Related Works

Parameterized **symmetric** systems: cache coherence protocols

- **Isolating two components** (among K), and applying existential abstraction

  FLASH Cache coherence protocol [McMillan 2001].
- **Counter Abstraction:** Count how many components are at a given state, and abstract as $\{0, 1, \infty\}$

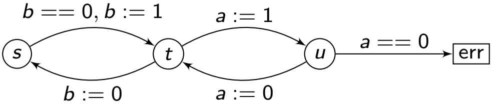
  [Pnueli, Xu, Zuck 2002]
- **Environment Abstraction**: the isolated components are seen as reference points and can change

  [Clarke, Talupur, Veith 2008]
- Similar abstraction + refinement by **non-interference lemmas**

  [Chou, Mannava, Park 2004]
  - Given a spurious counterexample, guess an invariant ϕ that excludes it
  - The model is constrained by ϕ which yields a finer abstraction
  - "Lemma" ϕ itself can be proven on the constrained model

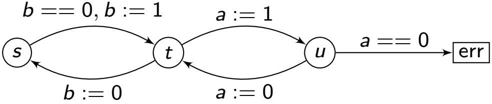
Automatic computation of the best refinement [Bingham 2008]

System: Shared variables a, b and identical components $C_1, \ldots, C_k, \ldots$:



On Refinement with Non-Interference Lemmas [CMP 2004]

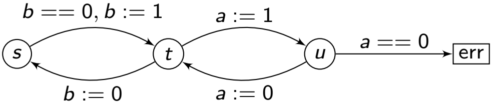
System: Shared variables a, b and identical components $C_1, \dots, C_k, \dots$:



Environment Abstraction: Keep variables a, b and one component C_1
States are tuples: $\boxed{v_a, v_b, q}$

On Refinement with Non-Interference Lemmas [CMP 2004]

System: Shared variables a, b and identical components $C_1, \ldots, C_k, \ldots$:



Environment Abstraction: Keep variables a, b and one component C_1
States are tuples: $\boxed{v_a, v_b, q}$

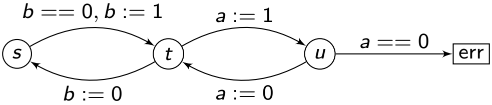
Initial abstract state: $\boxed{0,0,s}$ represents all states

$$a = 0, b = 0, S(C_1) = s, S(C_2) = x_2, S(C_3) = x_3, \ldots, S(C_k) = x_k,$$

for all $k \geq 1$, and all $x_2, \ldots, x_k$.

On Refinement with Non-Interference Lemmas [CMP 2004]

System: Shared variables a, b and identical components $C_1, \ldots, C_k, \ldots$:



Environment Abstraction: Keep variables a, b and one component C_1
States are tuples: $\boxed{v_a, v_b, q}$

Initial abstract state: $\boxed{0,0,s}$ represents all states

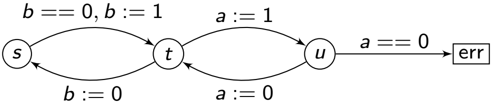
$$a = 0, b = 0, S(C_1) = s, S(C_2) = x_2, S(C_3) = x_3, \ldots, S(C_k) = x_k,$$

for all $k \geq 1$, and all $x_2, \ldots, x_k$.

Existential Abstraction: $\boxed{v_a, v_b, q} \rightarrow \boxed{v'_a, v'_b, q'}$
iff $\exists$ a transition in a concrete system that maps to this abstraction

On Refinement with Non-Interference Lemmas [CMP 2004]

System: Shared variables a, b and identical components $C_1, \ldots, C_k, \ldots$:



Environment Abstraction: Keep variables a, b and one component C_1
States are tuples: $\boxed{v_a, v_b, q}$

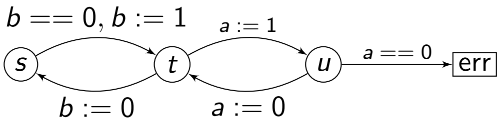
Initial abstract state: $\boxed{0,0,s}$ represents all states

$$a = 0, b = 0, S(C_1) = s, S(C_2) = x_2, S(C_3) = x_3, \ldots, S(C_k) = x_k,$$

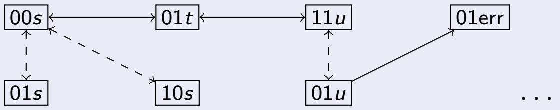
for all $k \geq 1$, and all $x_2, \ldots, x_k$.

Existential Abstraction: $\boxed{v_a, v_b, q} \rightarrow \boxed{v'_a, v'_b, q'}$
iff $\exists k, \exists x_2, x'_2, \ldots, x_k, x'_k.\ (v_a, v_b, q, x_2, \ldots, x_k) \rightarrow (v'_a, v'_b, q', x'_2, \ldots, x'_k)$.

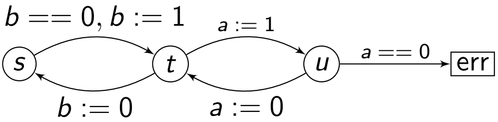
On Refinement with Non-Interference Lemmas [CMP 2004]



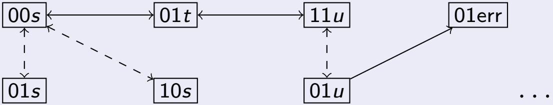
The Abstract System



On Refinement with Non-Interference Lemmas [CMP 2004]



The Abstract System

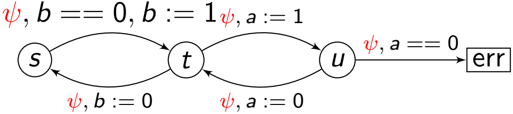


Counterexample present in all abstractions for all $k \geq 1$

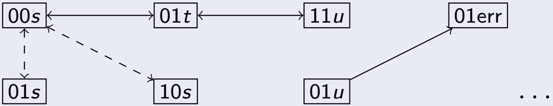
The following invariant explains why the counterex is spurious:

$$\psi = \forall i, j, i \neq j \Rightarrow \neg(S(C_i) \in \{t, u\} \wedge S(C_j) \in \{t, u\}).$$

On Refinement with Non-Interference Lemmas [CMP 2004]



The Abstract System



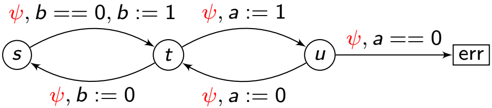
Counterexample present in all abstractions for all $k \geq 1$

The following invariant explains why the counterex is spurious:

$$\psi = \forall i, j, i \neq j \Rightarrow \neg(S(C_i) \in \{t, u\} \wedge S(C_j) \in \{t, u\}).$$

Strengthened Abstraction: $\boxed{v_a, v_b, q} \to \boxed{v'_a, v'_b, q'}$
iff $\exists k, \exists x_2, x'_2, \ldots, x_k, x'_k.\ (v_a, v_b, q, x_2, \ldots, x_k) \models \psi$
and $(v_a, v_b, q, x_2, \ldots, x_k) \to (v'_a, v'_b, q', x'_2, \ldots, x'_k)$.

On Refinement with Non-Interference Lemmas [CMP 2004]

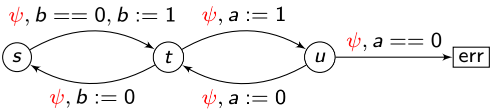


Remark

If ψ is invariant in the concrete system $\mathcal{A}$ (i.e. Reach($\mathcal{A}$) $\subseteq \psi$), then

$$\text{strengthen}(\mathcal{A}, \psi) \models \phi \Leftrightarrow \mathcal{A} \models \phi$$

On Refinement with Non-Interference Lemmas [CMP 2004]

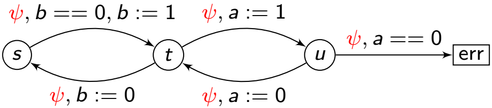


Remark

If ψ is invariant in the concrete system $\mathcal{A}$ (i.e. Reach($\mathcal{A}$) $\subseteq \psi$), then

$$\text{abstract}(\text{strengthen}(\mathcal{A}, \psi)) \models \phi \Rightarrow \mathcal{A} \models \phi$$

On Refinement with Non-Interference Lemmas [CMP 2004]



Remark

If ψ is invariant in the concrete system $\mathcal{A}$ (i.e. $\mathrm{Reach}(\mathcal{A}) \subseteq \psi$), then

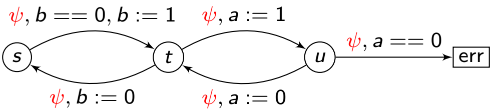
$$\mathrm{abstract}(\mathrm{strengthen}(\mathcal{A}, \psi)) \models \phi \Rightarrow \mathcal{A} \models \phi$$

Is ψ an invariant?

For **safety** properties: $\mathrm{strengthen}(\mathcal{A}, \psi) \models \psi \Leftrightarrow \mathcal{A} \models \psi$

So one can check this on the strengthened abstraction!

On Refinement with Non-Interference Lemmas [CMP 2004]



Remark

If ψ is invariant in the concrete system $\mathcal{A}$ (i.e. Reach($\mathcal{A}$) $\subseteq \psi$), then

$$\text{abstract}(\text{strengthen}(\mathcal{A}, \psi)) \models \phi \Rightarrow \mathcal{A} \models \phi$$

Is ψ an invariant?

For **safety** properties: strengthen($\mathcal{A}, \psi$) $\models \psi \Leftrightarrow \mathcal{A} \models \psi$

So one can check this on the strengthened abstraction!

Verification task: abstract(strengthen($\mathcal{A}, \psi$)) $\models \mathsf{G}\neg\mathrm{err} \wedge \psi$.

If there is again a spurious cex, then find ψ_2, and check abstract(strengthen($\mathcal{A}, \psi \wedge \psi_2$)) $\models \mathsf{G}\neg\mathrm{err} \wedge \psi \wedge \psi_2$.