Reconfiguration and message losses in Parameterized Broadcast Networks

Nathalie Bertrand[1], Patricia Bouyer[2] & Anirban Majumdar[1,2]

[1]Inria Rennes, France

[2]LSV, CNRS & ENS Paris-Saclay, France

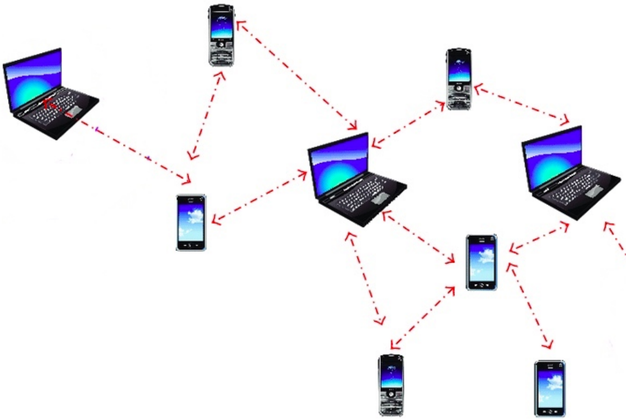
August 30, 2019

Mini-map

Mini-Map

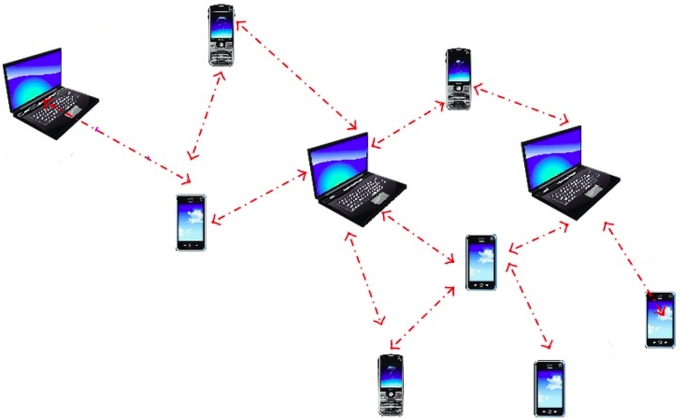
Ad-hoc Networks

- Devices (nodes) communicate wirelessly without a central access point.
- Any device can send message to its neighbours.



Ad-hoc Networks

- Devices (nodes) communicate wirelessly without a central access point.
- Any device can send message to its neighbours.



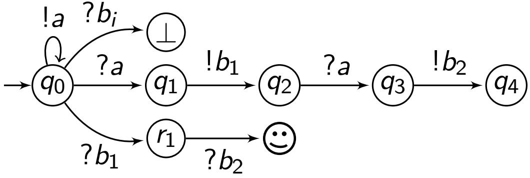
- Parameterized framework: the network should satisfy a given property for any number of devices.

Mini-Map

Broadcast Protocol

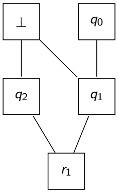
Finite state system whose transitions are labelled with:

- broadcast of messages - $!a$
- reception of messages - $?a$ [DSZ'10]



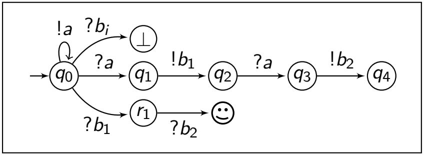
Semantics

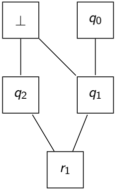
- Configuration: nodes (devices), labellings of nodes and edges.



Semantics

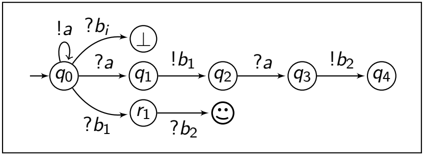
- Configuration: nodes (devices), labellings of nodes and edges.
- Every node follows the (same) protocol.

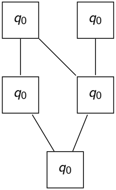




Semantics

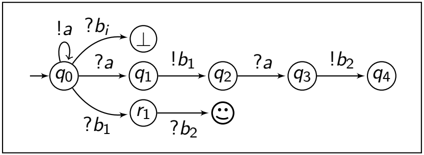
- Configuration: nodes (devices), labellings of nodes and edges.
- Every node follows the (same) protocol.
- Initial configuration: every node is in initial state.

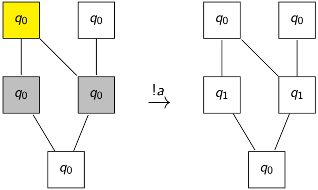




Semantics

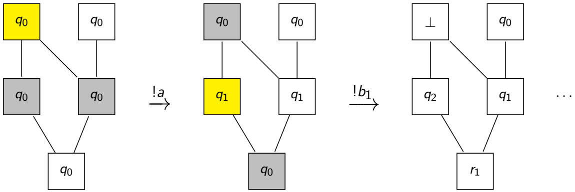
- Configuration: nodes (devices), labellings of nodes and edges.
- Every node follows the (same) protocol.
- Initial configuration: every node is in initial state.
- Nodes can send (yellow) messages to their neighbours (gray).





Two possible semantics

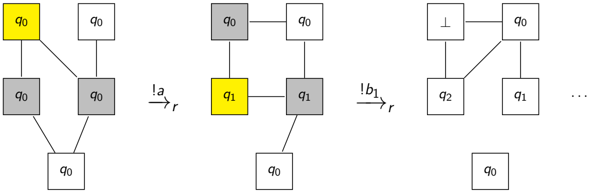
- Static: edges in the configuration graph is unchanged.



Semantics

Two possible semantics

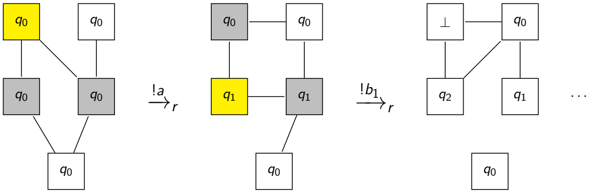
- Static: edges in the configuration graph is unchanged.
- Reconfigurable: edges may change arbitrarily at every step.



Semantics

Two possible semantics

- Static: edges in the configuration graph is unchanged.
- Reconfigurable: edges may change arbitrarily at every step.



Alert: Number of nodes does not change along an execution.

Reachability

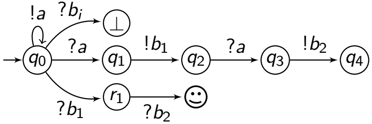
Reachability problem

Given a broadcast protocol $\mathcal{P}$ and target state ☺, does there exist $\gamma_0 \rightarrow^* \gamma$ such that γ_0 is initial and γ contains ☺.

Reachability

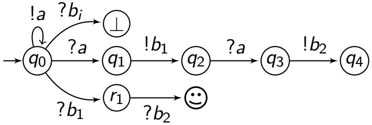
Reachability problem

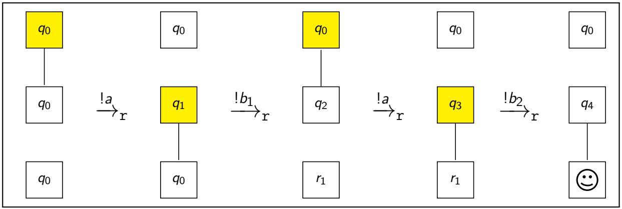
Given a broadcast protocol $\mathcal{P}$ and target state ☺, does there exist $\gamma_0 \rightarrow^* \gamma$ such that γ_0 is initial and γ contains ☺.



Reachability problem

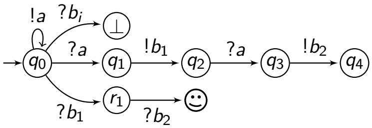
Given a broadcast protocol $\mathcal{P}$ and target state ☺, does there exist $\gamma_0 \rightarrow^* \gamma$ such that γ_0 is initial and γ contains ☺.





Reachability problem

Given a broadcast protocol $\mathcal{P}$ and target state ☺, does there exist $\gamma_0 \rightarrow^* \gamma$ such that γ_0 is initial and γ contains ☺.

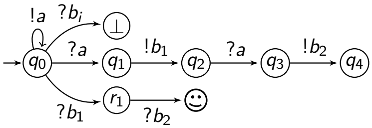


REACH = set of all reachable states.

Reachability

Reachability problem

Given a broadcast protocol $\mathcal{P}$ and target state ☺, does there exist $\gamma_0 \rightarrow^* \gamma$ such that γ_0 is initial and γ contains ☺.



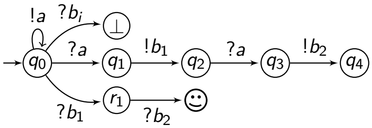
REACH = set of all reachable states.

Goal: Decide Reachability.

Reachability

Reachability problem

Given a broadcast protocol $\mathcal{P}$ and target state ☺, does there exist $\gamma_0 \rightarrow^* \gamma$ such that γ_0 is initial and γ contains ☺.



REACH = set of all reachable states.

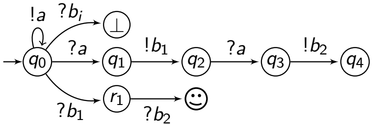
Goal: Decide Reachability.

- Undecidable in static semantics. [DSZ'10]

Reachability

Reachability problem

Given a broadcast protocol $\mathcal{P}$ and target state ☺, does there exist $\gamma_0 \rightarrow^* \gamma$ such that γ_0 is initial and γ contains ☺.



REACH = set of all reachable states.

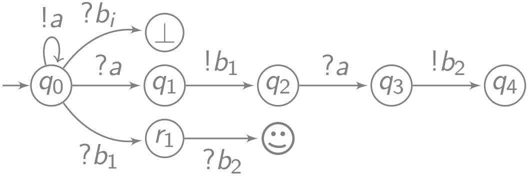
Goal: Decide Reachability.

- Undecidable in static semantics. [DSZ'10]
- PTIME algorithm for computing REACH in reconfigurable semantics. [DSTZ'12] (next)

Mini-Map

Saturation Algorithm

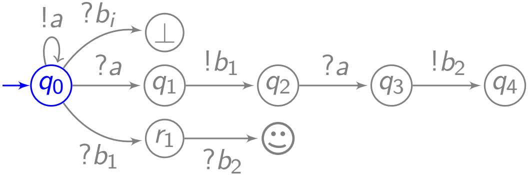
Recall: $q \in REACH \iff \exists \gamma_0 \rightarrow^* \gamma$ s.t. q is in γ.
Computing REACH: example



REACH $= \{\}$. [DSTZ'12]

Saturation Algorithm

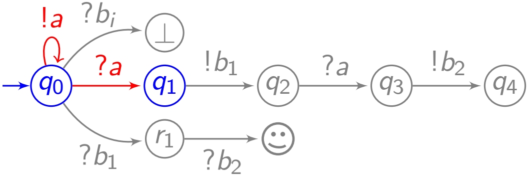
Recall: $q \in REACH \iff \exists \gamma_0 \rightarrow^* \gamma$ s.t. q is in γ.
Computing REACH: example



REACH $= \{q_0\}$. [DSTZ'12]

Saturation Algorithm

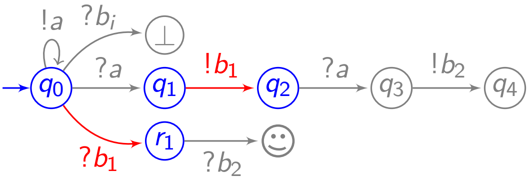
Recall: $q \in \mathit{REACH} \iff \exists \gamma_0 \rightarrow^* \gamma$ s.t. q is in γ.
Computing REACH: example



REACH $= \{q_0, q_1\}$. [DSTZ'12]

Saturation Algorithm

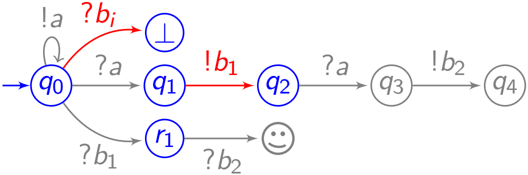
Recall: $q \in REACH \iff \exists \gamma_0 \rightarrow^* \gamma$ s.t. q is in γ.
Computing REACH: example



REACH $= \{q_0, q_1, q_2, r_1\}$. [DSTZ'12]

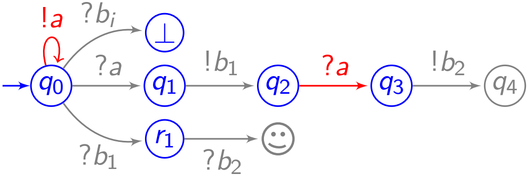
Saturation Algorithm

Recall: $q \in REACH \iff \exists \gamma_0 \to^* \gamma$ s.t. q is in γ.
Computing REACH: example



REACH $= \{q_0, q_1, q_2, r_1, \perp\}$. [DSTZ'12]

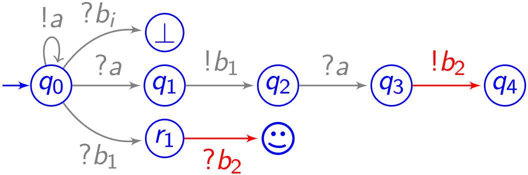
Recall: $q \in \mathit{REACH} \iff \exists \gamma_0 \to^* \gamma$ s.t. q is in γ.
Computing REACH: example



$\text{REACH} = \{q_0, q_1, q_2, r_1, \perp, q_3\}$. [DSTZ'12]

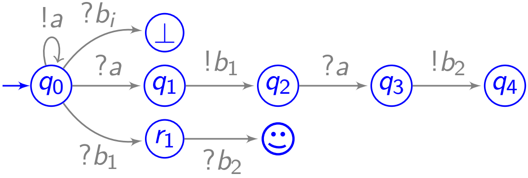
Saturation Algorithm

Recall: $q \in REACH \iff \exists \gamma_0 \rightarrow^* \gamma$ s.t. q is in γ.
Computing REACH: example



$\text{REACH} = \{q_0, q_1, q_2, r_1, \bot, q_3, q_4, ☺\}$. [DSTZ'12]

Recall: $q \in REACH \iff \exists \gamma_0 \to^* \gamma$ s.t. q is in γ.
Computing REACH: example

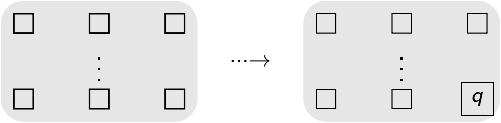


REACH $= \{q_0, q_1, q_2, r_1, \bot, q_3, q_4, ☺\}$. [DSTZ'12]

Correctness proof idea: duplicate the witness of iteration i.

Correctness - idea

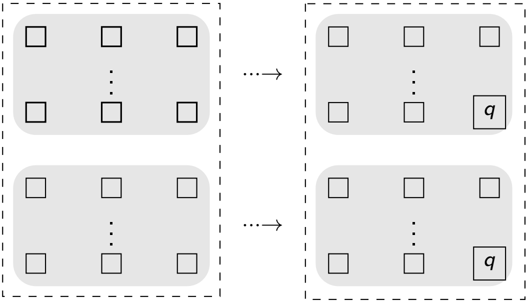
iteration $i \to i + 1 : \; q \xrightarrow{!a} q'$



[DSTZ'12]

Correctness - idea

iteration $i \to i+1$: $q \xrightarrow{!a} q'$

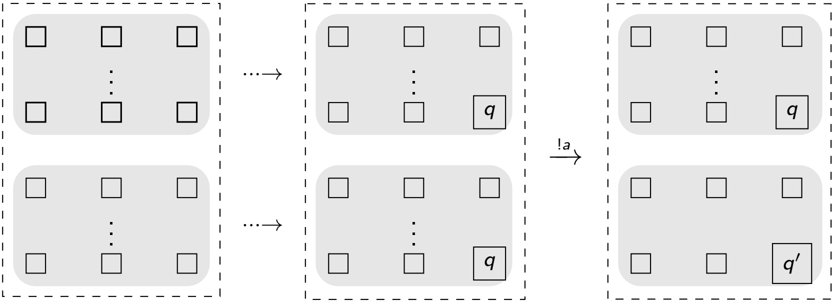


Duplicate the execution

[DSTZ'12]

Correctness - idea

iteration $i \to i+1 : \; q \xrightarrow{!a} q'$

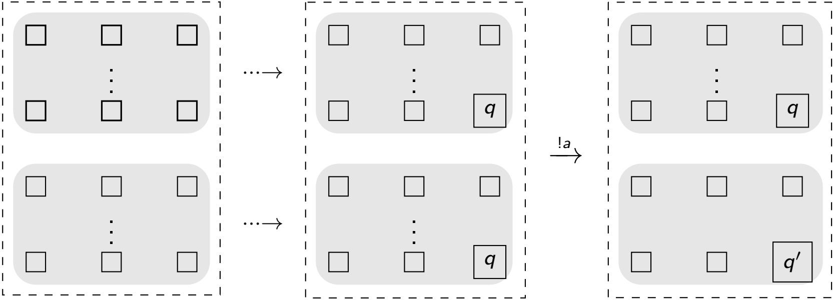


Duplicate the execution and new node takes the transition.

[DSTZ'12]

Correctness - idea

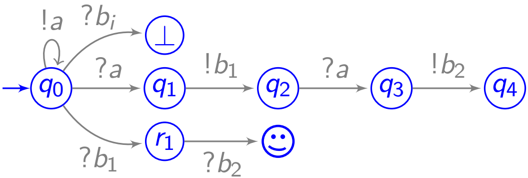
iteration $i \to i+1: \; q \xrightarrow{?a} q',\, q'' \xrightarrow{!a} q'''$.



Duplicate the execution and new node takes the transition.
Similar for **reception**. [DSTZ'12]

Saturation Algorithm

Recall: $q \in REACH \iff \boxed{\exists \gamma_0 \rightarrow^* \gamma}$ s.t. q is in γ.
Computing REACH: example



REACH $= \{q_0, q_1, q_2, r_1, \perp, q_3, q_4, ☺\}$. [DSTZ'12]

Correctness proof idea: duplicate the witness of iteration i.
Final witness has size **exponential**.
What about a minimum size witness?

Some measures

Cutoff

Minimal number of nodes to **reach ☺.**

Covering length

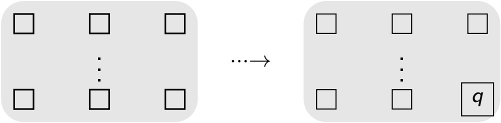
Length of a minimal execution to **reach ☺.**

Our goal: given a protocol, find the cutoff and covering length.

Mini-Map

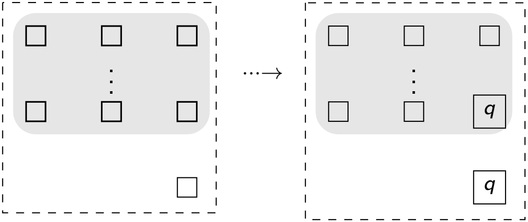
Copycat property

Duplication not needed - one extra node is enough to follow a certain node.



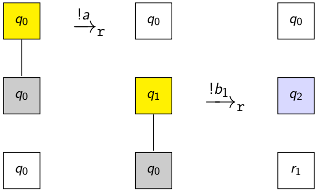
Copycat property

Duplication not needed - one extra node is enough to follow a certain node.

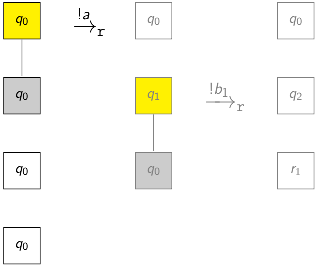


The new node exactly follows the **old** node, thanks to reconfiguration.

Copycat property - Example

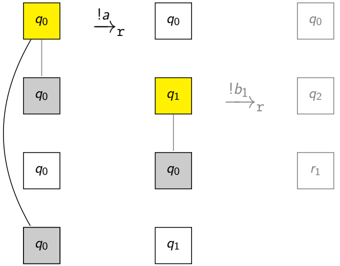


Copycat property - Example



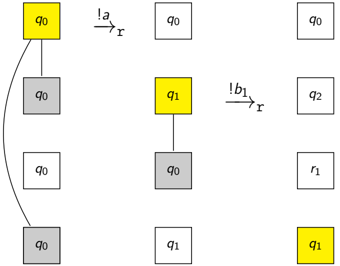
Introduce the new node (which will follow the second node).

Copycat property - Example



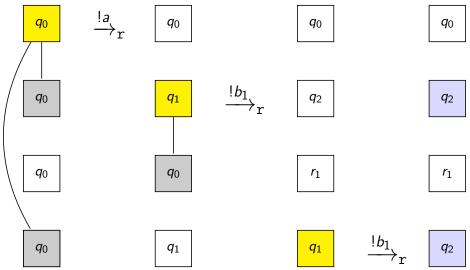
Introduce the new node (which will follow the second node).
New node **receives** message as old node.

Copycat property - Example



Introduce the new node (which will follow the second node).
New node **receives** message as old node.
Sending is simulated in two steps - old node sends as before

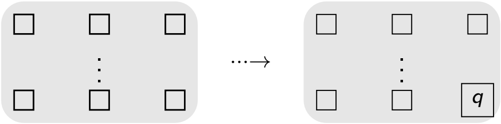
Copycat property - Example



Introduce the new node (which will follow the second node).
New node **receives** message as old node.
Sending is simulated in two steps - old node sends as before, then disconnect the new node and new node sends the same message.

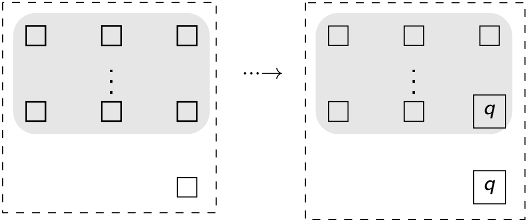
Saturation Algorithm - revisited

iteration $i \to i+1$: $q \xrightarrow{!a} q'$



Saturation Algorithm - revisited

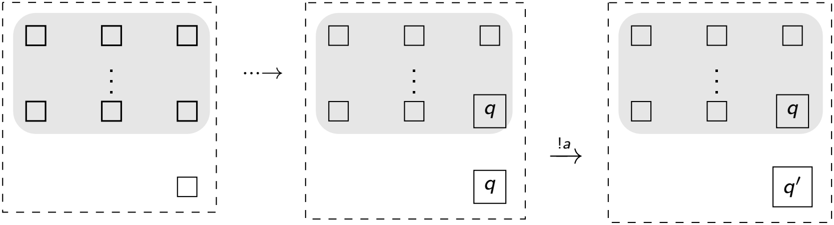
iteration $i \to i+1: \; q \xrightarrow{!a} q'$



Duplicate the “required” node

Saturation Algorithm - revisited

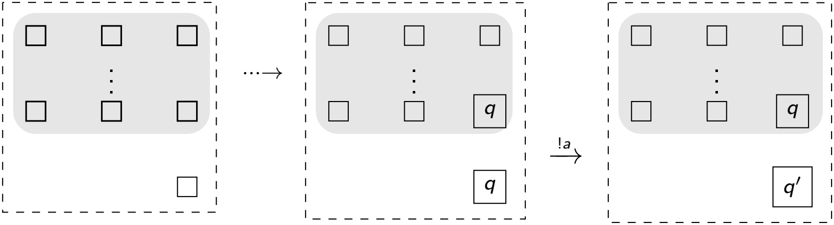
iteration $i \to i+1$: $q \xrightarrow{!a} q'$



Duplicate the “required” node and new node takes the transition.

Saturation Algorithm - revisited

iteration $i \to i+1:\ q \xrightarrow{?a} q',\ q'' \xrightarrow{!a} q'''$.



Duplicate the "required" node and new node takes the transition.
Similar for **reception**; with two new nodes.

Consequence on cutoff and covering length

Results

- A minimal witness has size at most $O(n)$ [more precisely, $2n$].
- A minimal witness has length at most $O(n^2)$.

Consequence on cutoff and covering length

Results

- A minimal witness has size at most $O(n)$ [more precisely, $2n$].
- A minimal witness has length at most $O(n^2)$.
- Matching lower bounds (example : a family of protocols).

Mini-Map

Lossy Networks

- Message can be lost in the network.

Lossy Networks

- Message can be lost in the network.
- Reconfigurable semantics: lossy semantics equivalent to non-lossy semantics.

Lossy Networks

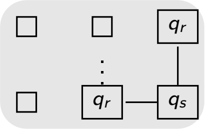
- Message can be lost in the network.
- Reconfigurable semantics: lossy semantics equivalent to non-lossy semantics.
- Focus on static topology (edges remain the same throughout).

Lossy Networks

- Message can be lost in the network.
- Reconfigurable semantics: lossy semantics equivalent to non-lossy semantics.
- Focus on static topology (edges remain the same throughout).
- Two possible semantics:
  - Loss on reception. [DSZ'12]

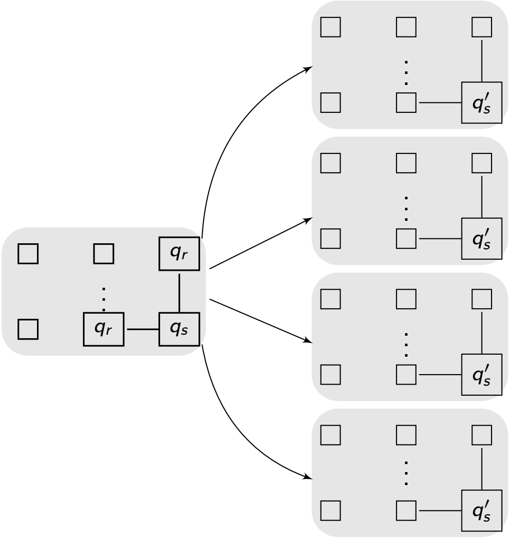
Loss on reception

Consider $q_s \xrightarrow{!a} q'_s$; $q_r \xrightarrow{?a} q'_r$.



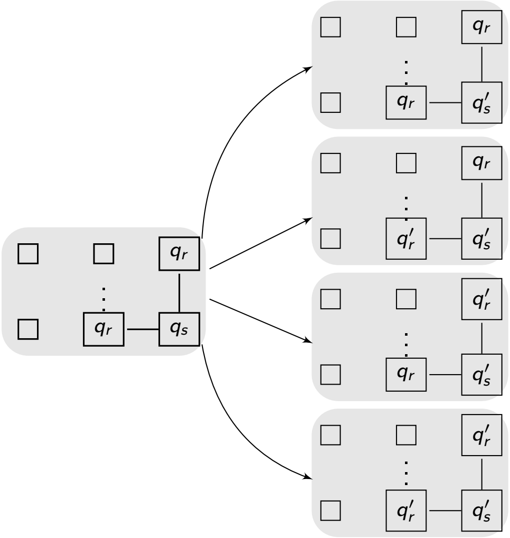
Loss on reception

Consider $q_s \xrightarrow{!a} q'_s$; $q_r \xrightarrow{?a} q'_r$.



Loss on reception

Consider $q_s \xrightarrow{!a} q_s'$; $q_r \xrightarrow{?a} q_r'$.



Lossy Networks

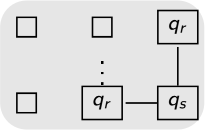
- Message can be lost in the network.
- Reconfigurable semantics: lossy semantics equivalent to non-lossy semantics.
- Focus on static topology (edges remain the same throughout).
- Two possible semantics:
    - Loss on reception. [DSZ'12] (equiv. to reconfigurable non-lossy semantics)

Lossy Networks

- Message can be lost in the network.
- Reconfigurable semantics: lossy semantics equivalent to non-lossy semantics.
- Focus on static topology (edges remain the same throughout).
- Two possible semantics:
  - Loss on reception. [DSZ'12] (equiv. to reconfigurable non-lossy semantics)
  - Loss on sending. [This work]

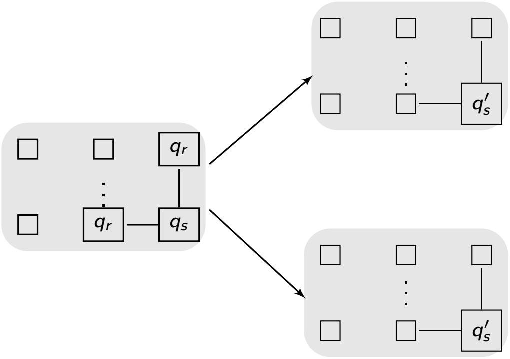
Loss on sending

Consider $q_s \xrightarrow{!a} q_s'$; $q_r \xrightarrow{?a} q_r'$.



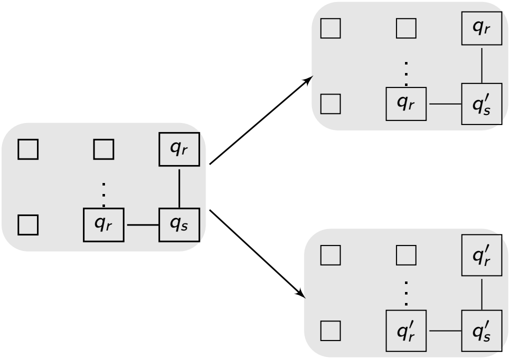
Loss on sending

Consider $q_s \xrightarrow{!a} q_s'$; $q_r \xrightarrow{?a} q_r'$.



Loss on sending

Consider $q_s \xrightarrow{!a} q_s'$; $q_r \xrightarrow{?a} q_r'$.

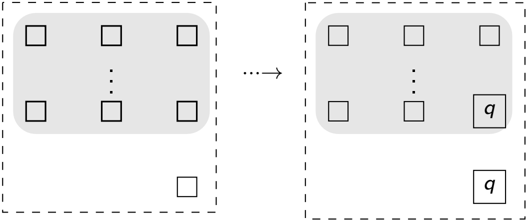


Lossy Networks

- Message can be lost in the network.
- Reconfigurable semantics: lossy semantics equivalent to non-lossy semantics.
- Focus on static topology (edges remain the same throughout).
- Two possible semantics:
  - Loss on reception. [DSZ'12] (equiv. to reconfigurable non-lossy semantics)
  - Loss on sending. [This work]
    - Goal: compute REACH.
    - What about **cutoff, covering length?**

Copycat property - Lossy net

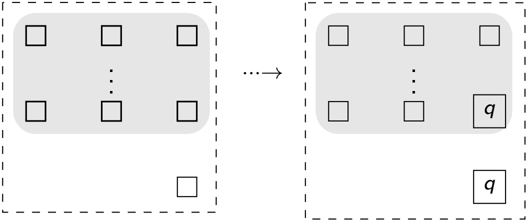
One can copy any node by an extra node.



The new node exactly follows the **old** node.

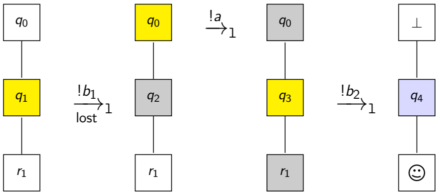
Copycat property - Lossy net

One can copy any node by an extra node.



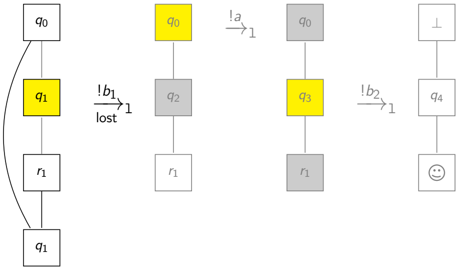
The new node exactly follows the **old** node.
Alert: static topology.

Copycat property - Lossy net - Example



Note - Static topology.

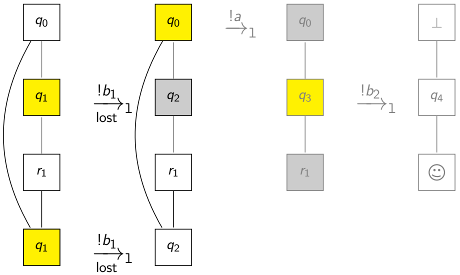
Copycat property - Lossy net - Example



Note - Static topology.

Introduce new node; it has the same connections as old node.

Copycat property - Lossy net - Example

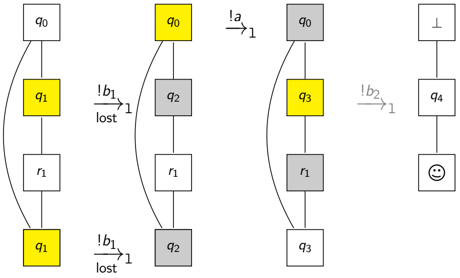


Note - Static topology.

Introduce new node; it has the same connections as old node.

Lossy transition is taken by both nodes.

Copycat property - Lossy net - Example



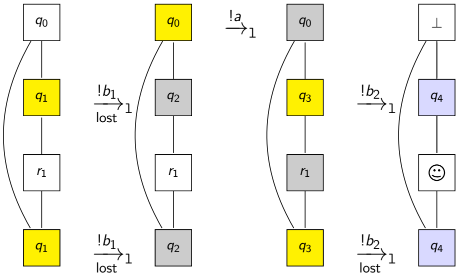
Note - Static topology.

Introduce new node; it has the same connections as old node.

Lossy transition is taken by both nodes.

New node **receives** message as old node.

Copycat property - Lossy net - Example



Note - Static topology.

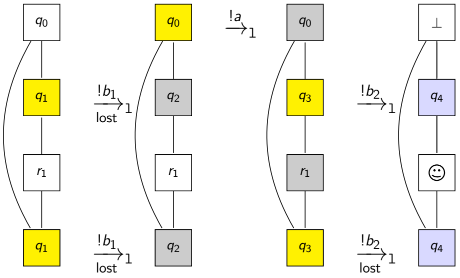
Introduce new node; it has the same connections as old node.

Lossy transition is taken by both nodes.

New node **receives** message as old node.

Sending is simulated in two steps - old node sends as before, and new node sends a lossy message.

Copycat property - Lossy net - Example



Note - Static topology.

Introduce new node; it has the same connections as old node.

Lossy transition is taken by both nodes.

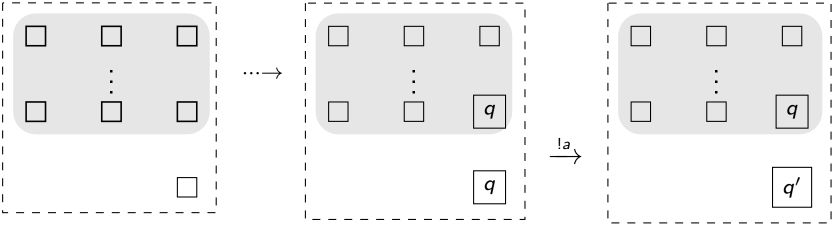
New node **receives** message as old node.

Sending is simulated in two steps - old node sends as before, and new node sends a lossy message.

Note - all messages by new node are **lossy**.

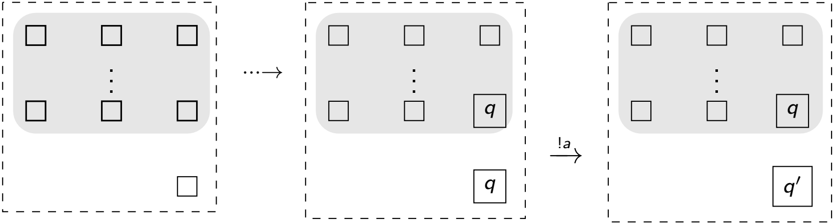
Saturation Algorithm - Lossy net

iteration $i \to i+1$: $q \xrightarrow{!a} q'$



Saturation Algorithm - Lossy net

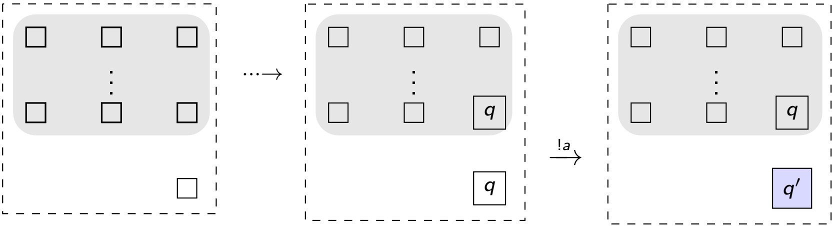
iteration $i \to i+1$: $q \xrightarrow{!a} q'$



Alert: static topology

Saturation Algorithm - Lossy net

iteration $i \to i+1$: $q \xrightarrow{!a} q'$

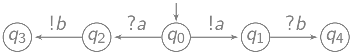


Alert: static topology

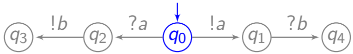
Idea: keep a **main** copy for each Reachable state.

The **main** copy is **untouched** in future iterations.

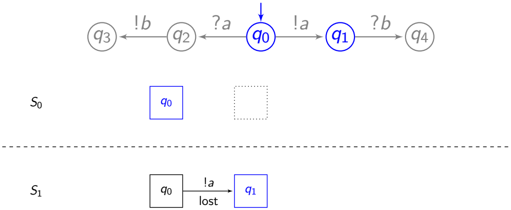
Saturation Algorithm - Lossy net - Example



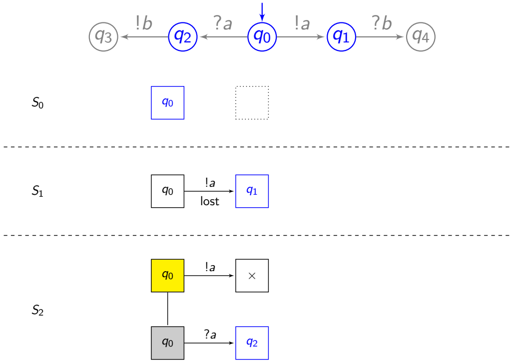
Saturation Algorithm - Lossy net - Example



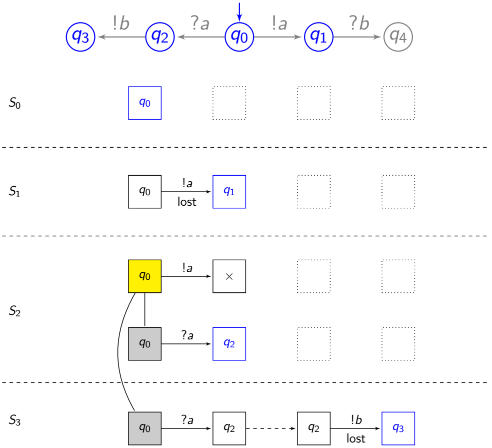
Saturation Algorithm - Lossy net - Example



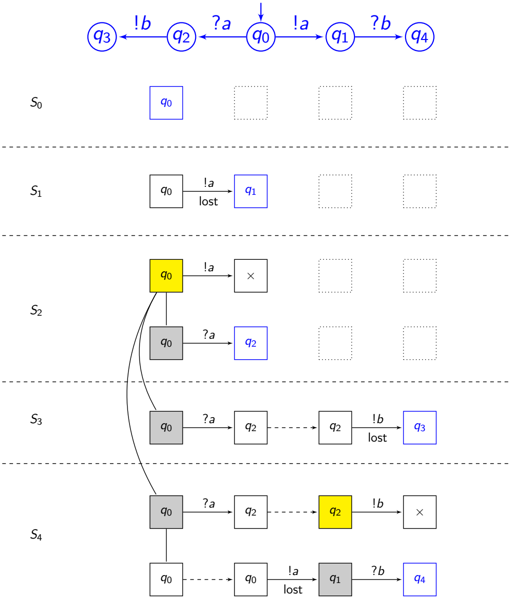
Saturation Algorithm - Lossy net - Example



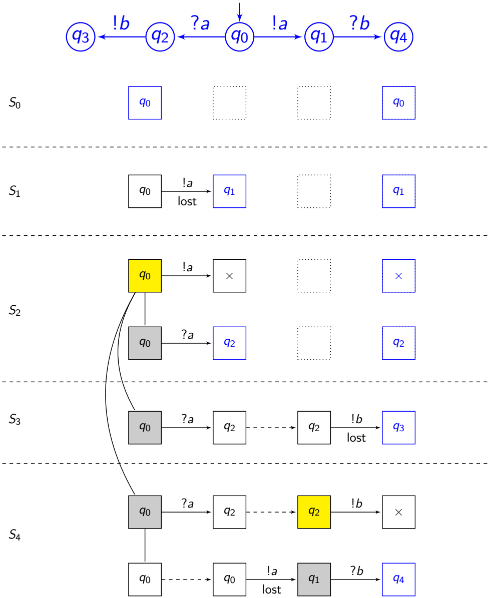
Saturation Algorithm - Lossy net - Example



Saturation Algorithm - Lossy net - Example



Saturation Algorithm - Lossy net - Example



Complexity measures in lossy semantics

Results

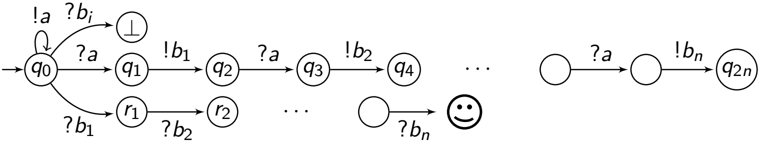
- REACH is same as in reconfigurable non-lossy semantics.
- Similar bounds for cutoff $[O(n)]$ and covering length $[O(n^2)]$.

Mini-Map

Reconfigurable semantics needs less nodes than lossy semantics:

Reconfiguration is more succinct

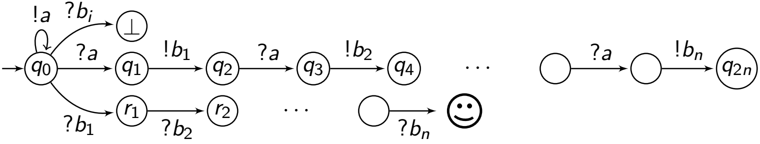
Reconfigurable semantics needs less nodes than lossy semantics:



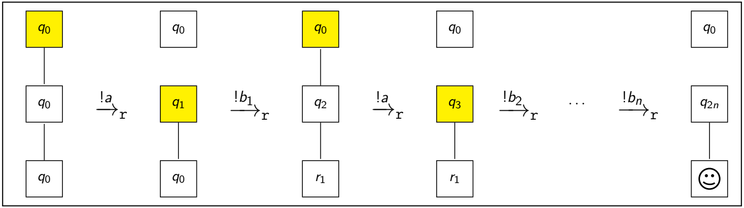
One needs 3 nodes in reconfigurable semantics to reach ☺.

Reconfiguration is more succinct

Reconfigurable semantics needs less nodes than lossy semantics:

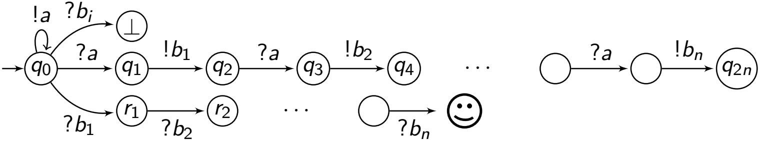


One needs 3 nodes in reconfigurable semantics to reach ☺.

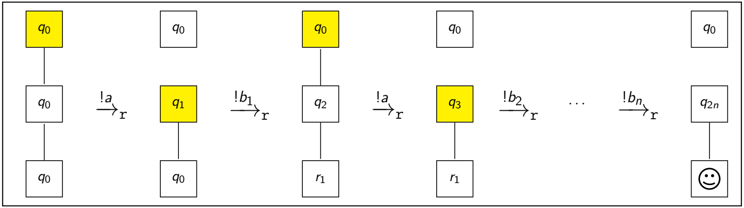


Reconfiguration is more succinct

Reconfigurable semantics needs less nodes than lossy semantics:



One needs 3 nodes in reconfigurable semantics to reach ☺.



But, one needs at least $O(n)$ many nodes in lossy sematics.

Finding minimal covering execution

Cutoff is at most linear.

What about exact size of a minimal witness?

Finding minimal covering execution

Cutoff is at most linear.

What about exact size of a minimal witness?

MinCover

Input: A broadcast protocol $\mathcal{P}$, target state ☺, and $k \in \mathbb{N}$.

Output: Yes iff there exists a covering execution with k many nodes.

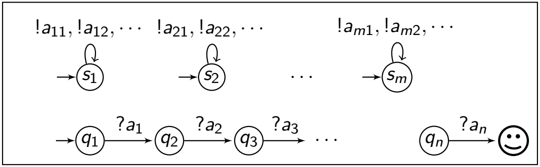
Finding minimal covering execution

Cutoff is at most linear.

What about exact size of a minimal witness?

MinCover
Input: A broadcast protocol $\mathcal{P}$, target state ☺, and $k \in \mathbb{N}$.
Output: Yes iff there exists a covering execution with k many nodes.

- NP-complete.
- Proof of hardness: reduction from **set-cover**.



The instance has a cover of size k if and only if there is a witness execution of size $k+1$.

Mini-Map

Conclusion/Future

- REACH is same for reconfigurable non-lossy semantics and static lossy semantics.
- Cutoff is at most $O(n)$ (both semantics).
- And covering length at most $O(n^2)$ (both semantics).
- Similar lower bounds as well (both semantics).
- But reconfigurable non-lossy semantics is more succinct.
- Finding minimal covering execution NP-complete (both semantics).

- What is the tradeoff between the size and length of an execution?
- What about probabilistic message losses?

Thank You