Topological Self-Stabilization with Name-Passing Process Calculi

Christina Rickmann
Christoph Wagner, Uwe Nestmann, Stefan Schmid

24. August 2016

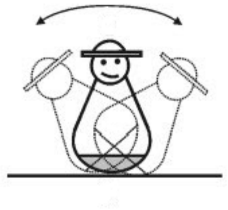
Technische Universität Berlin

Modelle und Theorie Verteilter Systeme

Idea

"'We call the system "self-stabilizing" if and only if, regardless of the initial state [...], the system is guaranteed to find itself in a legitimate state after a finite number of moves."'

- Dijkstra 1974

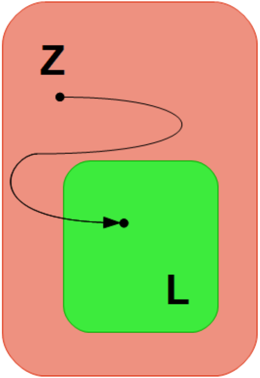


Definition

Self-stabilization

A system is self-stabilizing if and only if (provided no fault occurs)

Convergence: started in any arbitrary state, the system reaches a desired state after a finite number of steps and



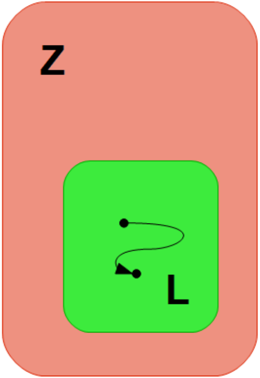
Definition

Self-stabilization

A system is self-stabilizing if and only if (provided no fault occurs)

Convergence: started in any arbitrary state, the system reaches a desired state after a finite number of steps and

Closure: if the system is in a desired state, it remains in a desired state.

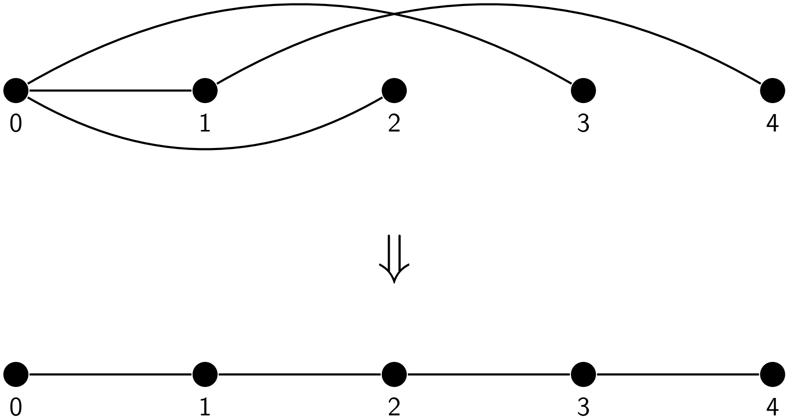


Characteristics

Self-stabilization:

- specialization of nonmasking fault tolerance
- tolerate arbitrary transient faults
- no initialization
- no fault detection
- must not terminate
- no local knowledge whether stabilized
- adapt to dynamic changes

Linearization



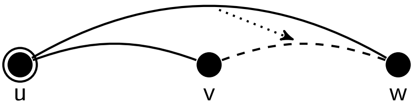
Original Shared-Memory-Algorithm

Gall et al. (2014):
for every node u, there are the following rules for every pair of neighbors v and w

linearize right(v,w) :

$(v, w \in u.R \ \wedge \ u < v < w)$

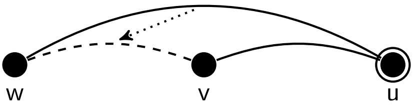
$\rightarrow \ e(u,w) := 0, \ e(v,w) := 1$



linearize left(v,w) :

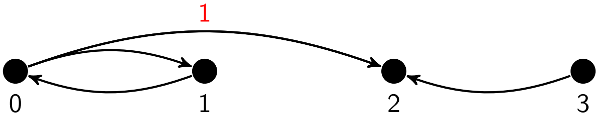
$(v, w \in u.L \ \wedge \ w < v < u)$

$\rightarrow \ e(u,w) := 0, \ e(v,w) := 1$



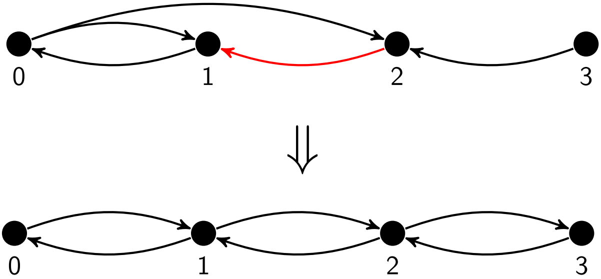
Asynchronous Message-Passing-Algorithm

asynchronous messages: non blocking



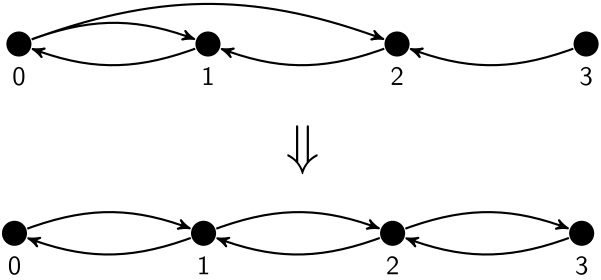
Asynchronous Message-Passing-Algorithm

asynchronous messages: non blocking



Asynchronous Message-Passing-Algorithm

asynchronous messages: non blocking



asynchronous message passing + changing communication structure
+ distributed
$\Rightarrow$
localized π-calculus

Asynchronous Message-Passing-Algorithm

$$Alg\,(p, nb) = (\nu\, nb_p) \left(\overline{nb_p}\langle nb\rangle \mid Alg_{rec}\,(p) \mid Alg_{match}\,(p) \right)$$

$$Alg'\,(p, nb, x) = (\nu\, nb_p) \left(\overline{nb_p}\langle nb\rangle \mid Alg_{add}\,(p, x) \mid Alg_{match}\,(p) \right)$$

$$Alg_{rec}\,(p) = p(x)\,.Alg_{add}\,(p, x)$$

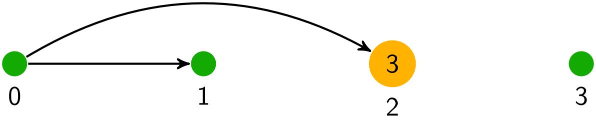
$$Alg_{add}\,(p, x) = nb_p(y)\,.\left(\overline{nb_p}\langle y \cup \{x\}\rangle \mid Alg_{rec}\,(p)\right)$$

$$\begin{aligned}
Alg_{match}\,(p) = nb_p(y)\,.(\,&\textbf{let } x = select\,(findLin\,(p, y))\ \textbf{in}\\
&\textbf{if } x = \bot \textbf{ then } \prod_{j \in y} \bar{j}\langle p\rangle \mid \overline{nb_p}\langle y\rangle\\
&\textbf{else if } x = (j, k) \textbf{ then}\\
&\quad \textbf{if } j < k \wedge k < p \textbf{ then } \bar{j}\langle k\rangle \mid \overline{nb_p}\langle y \setminus \{j\}\rangle\\
&\quad \textbf{else if } j < k \wedge p < j \textbf{ then } \bar{k}\langle j\rangle \mid \overline{nb_p}\langle y \setminus \{k\}\rangle\\
&\ldots\\
&\mid Alg_{match}\,(p)\,)
\end{aligned}$$

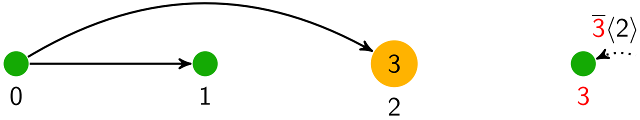
Asynchronous Message-Passing-Algorithm

Asynchronous Message-Passing-Algorithm

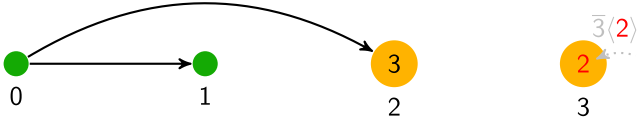
Asynchronous Message-Passing-Algorithm



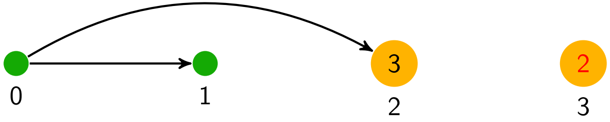
Asynchronous Message-Passing-Algorithm



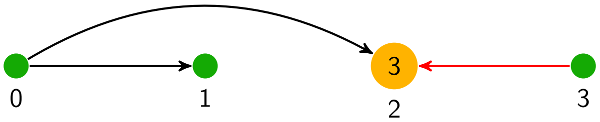
Asynchronous Message-Passing-Algorithm



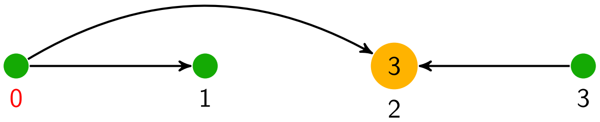
Asynchronous Message-Passing-Algorithm



Asynchronous Message-Passing-Algorithm

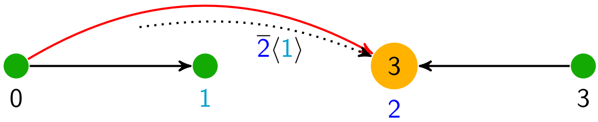


Asynchronous Message-Passing-Algorithm

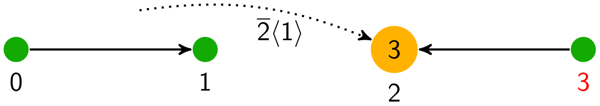


Asynchronous Message-Passing-Algorithm

Asynchronous Message-Passing-Algorithm

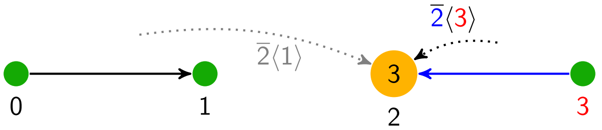


Asynchronous Message-Passing-Algorithm

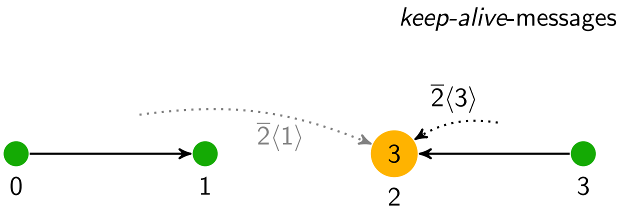


Asynchronous Message-Passing-Algorithm

keep-alive-messages



Asynchronous Message-Passing-Algorithm

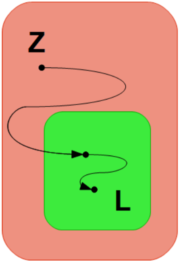


Standard Proof Techniques

Let Z be the set of all states and $L \subseteq Z$ the set of legal/desired states

Convergence

Starting from a state in $Z \setminus L$, after a limited number of steps a state in L is reached.
$\Rightarrow$ construct a function $t : Z \rightarrow \mathbb{N}$ (potential function) that decreases with every step and for every state $x \in L$ holds $t(x) = 0$

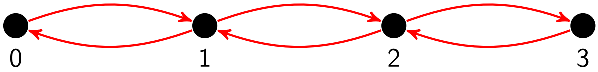


Closure

Starting in a state in L, each following state again is in L.
$\Rightarrow$ usually through an invariant

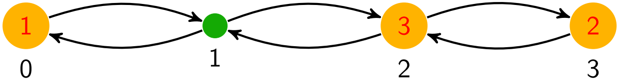
Closure

correct configuration is unique



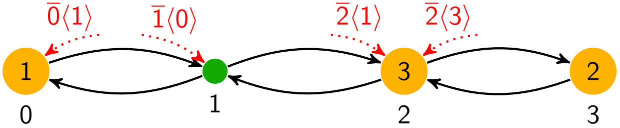
Closure

correct configuration is unique (up to)



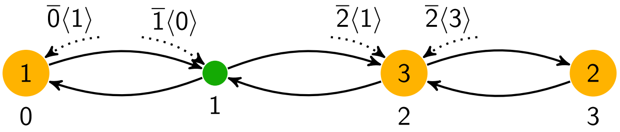
Closure

correct configuration is unique (up to)



Closure

correct configuration is unique (up to)



Idea closure proof:

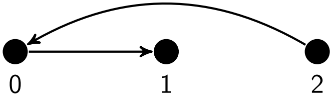
after every step again a correct configuration

- no linearization steps
- actions involve only desired neighbors

$\Rightarrow$ topology remains unchanged

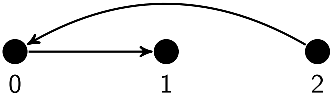
Convergence

Problem: *keep-alive*-messages



Convergence

Problem: *keep-alive*-messages



Weak Convergence

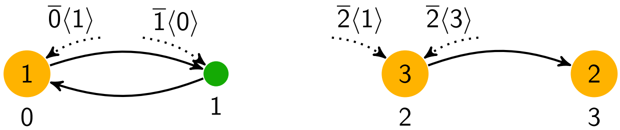
Starting from any arbitrary state, there is always a way to reach a desired state after a limited number of steps.

Strong convergence for restricted cases
Weak convergence in general

Convergence

(Strong) convergence in case:

- no undesired connections anymore



Convergence

(Strong) convergence in case:

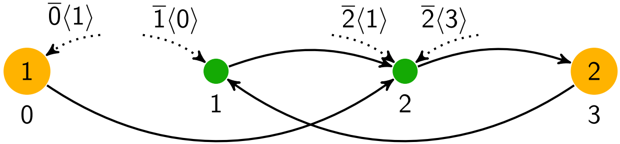
- no undesired connections anymore

everyone sending *keep-alive*-messages,
all received and processed eventually

Convergence

(Strong) convergence in case:

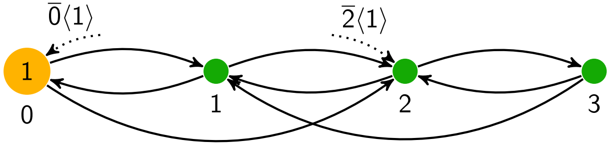
- no undesired connections anymore
- all desired connections established



Convergence

(Strong) convergence in case:

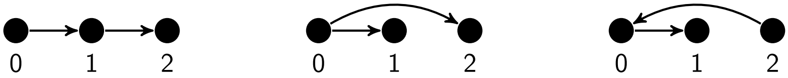
- no undesired connections anymore
- all desired connections established



eventually all desired edges
no unnecessary *keep-alive*-messages
convergence by potential function

Weak convergence in general:

- subset of executions without unnecessary *keep-alive*-messages
  non-empty for each initial configuration
  converges strongly



Conclusion

Our contributions:

- redesigning algorithm shared memory $\rightarrow$ asynchronous message-passing
- proving:
  - closure,
  - weak convergence in general,
  - strong convergence for restricted cases
- discussing strong convergence

Future work:

- proving strong convergence

Thanks! Questions?

Fault Tolerance

- Faults occur, we have to deal with them!
- masking fault tolerance:
  - aim: avoid system failure if possible
  - fault model
  - describes all faults that can be tolerated
  - never takes all possible faults into account
  - other faults may lead to system failure
  - needs redundancy in space or time
- nonmasking fault tolerance:
  - system may fail partly or temporarily
  - better than complete and/or permanent failure

Extended Localized Pi

DATA VALUES $\mathbf{V}$	$v ::= \bot \mid 0 \mid 1 \mid c \mid (v, v) \mid$ $\{v, \dots, v\} \mid \{\!\mid v, \dots, v \mid\!\}$, with $c \in \mathbf{A}$
VARIABLE PATTERN	$X ::= x \mid (X, X)$, with $x \in \mathbf{A}$
EXPRESSIONS	$e ::= v \mid X \mid (e, e) \mid f(e)$, with $f \in \mathbf{A}$
PROCESSES $\mathbf{P}$	$P ::= 0 \mid P \mid P \mid c(X).P \mid \overline{c}\langle v \rangle \mid (\nu c)\, P \mid$ $\textbf{if } e \textbf{ then } P \textbf{ else } P \mid \textbf{let } X = e \textbf{ in } P \mid K(e)$
PROCESS EQUATIONS	$D = \{K_j(X) = P_j\}_{j \in J}$ a finite set of process definitions

where in $c(X).P$ variable x as part of X may not occur free in P in input position.

Structural Congruence

$$P \equiv Q \text{ if } P \equiv_\alpha Q \qquad P \mid 0 \equiv P \qquad P \mid Q \equiv Q \mid P$$

$$P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad (\nu n)\, 0 \equiv 0$$

$$P \mid (\nu n)\, Q \equiv (\nu n)\, (P \mid Q)\, , \text{if } n \notin \mathrm{fn}(P)$$

$$(\nu n)\, (\nu m)\, P \equiv (\nu m)\, (\nu n)\, P$$

$$\textbf{if } e \textbf{ then } P \textbf{ else } Q \equiv P, \ \text{ if } [\![e]\!] = 1$$

$$\textbf{if } e \textbf{ then } P \textbf{ else } Q \equiv Q, \ \text{ if } [\![e]\!] = 0$$

$$\textbf{let } X = e \textbf{ in } P \equiv \{[e]/x\}P$$

$$K(e) \equiv \{[e]/x\}P, \ \text{ if } (K(X) = P) \in D$$

Reduction Semantics

$$\texttt{comm:}\ \frac{}{c(X).P \mid \overline{c}\langle v\rangle \longmapsto \{{}^{v}\!/x\}P}$$

$$\texttt{res:}\ \frac{P \longmapsto P'}{(\nu c)\, P \longmapsto (\nu c)\, P'}$$

$$\texttt{par:}\ \frac{P \longmapsto P'}{P \mid Q \longmapsto P' \mid Q}$$

$$\texttt{struct:}\ \frac{P \equiv Q \qquad Q \longmapsto Q' \qquad Q \equiv Q'}{P \longmapsto P'}$$

System Assumptions

Process Ids

Every process has a unique constant id and every value in the system can be interpreted as the id of an existing process.

No Message Loss

Every message is received after a finite but arbitrary number of steps.

Fairness

Every continuously enabled subprocess will eventually (after an arbitrary but finite number of steps) execute a step.

Weakly Connected

The topology with messages is initially weakly connected.

Algorithm I

$$Alg\,(p, initNb) = (\nu\, nb_p) \left(\overline{nb_p}\langle initNb\rangle \mid Alg_{rec}\,(p) \mid Alg_{match}\,(p) \right)$$

$$Alg'\,(p, initNb, x) = (\nu\, nb_p) \left(\overline{nb_p}\langle initNb\rangle \mid Alg_{add}\,(p, x) \mid Alg_{match}\,(p) \right)$$

$$Alg_{rec}\,(p) = p(x)\,.Alg_{add}\,(p, x)$$

$$Alg_{add}\,(p, x) = nb_p(y)\,.\left(\overline{nb_p}\langle y \cup \{x\}\rangle \mid Alg_{rec}\,(p) \right)$$

Algorithm II

$$
\begin{aligned}
Alg_{match}(p) = nb_p(y) . \Big(& \textbf{let } x = select(findLin(p, y)) \textbf{ in} \\
& \textbf{if } x = \bot \textbf{ then} \\
& \quad \prod_{j \in y} \overline{j}\langle p \rangle \mid \overline{nb_p}\langle y \rangle \\
& \textbf{else if } x = (j, k) \textbf{ then} \\
& \qquad \textbf{if } j < k \wedge k < p \textbf{ then} \\
& \qquad\quad \overline{j}\langle k \rangle \mid \overline{nb_p}\langle y \setminus \{j\} \rangle \\
& \qquad \textbf{else if } j < k \wedge p < j \textbf{ then} \\
& \qquad\quad \overline{k}\langle j \rangle \mid \overline{nb_p}\langle y \setminus \{k\} \rangle \\
& \qquad \textbf{else } \overline{nb_p}\langle y \rangle \\
& \textbf{else } \overline{nb_p}\langle y \rangle \\
& \mid Alg_{match}(p) \Big)
\end{aligned}
$$

Algorithm III

LeftN : $\mathcal{P} \times 2^{\mathcal{P}} \rightarrow 2^{\mathcal{P}}$ calculates the *left neighborhood* of a process and corresponding *RightN* : $\mathcal{P} \times 2^{\mathcal{P}} \rightarrow 2^{\mathcal{P}}$ the *right neighborhood* of a process

$$\begin{aligned} LeftN(p, y) =& \{q \in \mathcal{P} | q \in y \wedge q < p\} \\ RightN(p, y) =& \{q \in \mathcal{P} | q \in y \wedge q > p\} \end{aligned}$$

Algorithm III

$LeftN : \mathcal{P} \times 2^{\mathcal{P}} \rightarrow 2^{\mathcal{P}}$ calculates the *left neighborhood* of a process and corresponding $RightN : \mathcal{P} \times 2^{\mathcal{P}} \rightarrow 2^{\mathcal{P}}$ the *right neighborhood* of a process

$$\begin{aligned} LeftN(p, y) =& \{q \in \mathcal{P} | q \in y \wedge q < p\} \\ RightN(p, y) =& \{q \in \mathcal{P} | q \in y \wedge q > p\} \end{aligned}$$

The function $findLin : \mathcal{P} \times 2^{\mathcal{P} \times \mathcal{P}} \rightarrow 2^{\mathcal{P} \times \mathcal{P}}$ calculates all possible linearization steps in the neighborhood of a process.

$$findLin(p, y) = \{(q, r) | q, r \in y \wedge q < r \wedge (q, r \in LeftN(p, y) \vee RightN(p, y))\}$$

Algorithm III

LeftN : $\mathcal{P} \times 2^{\mathcal{P}} \rightarrow 2^{\mathcal{P}}$ calculates the *left neighborhood* of a process and corresponding *RightN* : $\mathcal{P} \times 2^{\mathcal{P}} \rightarrow 2^{\mathcal{P}}$ the *right neighborhood* of a process

$$\begin{aligned} LeftN(p, y) &= \{q \in \mathcal{P} | q \in y \wedge q < p\} \\ RightN(p, y) &= \{q \in \mathcal{P} | q \in y \wedge q > p\} \end{aligned}$$

The function *findLin* : $\mathcal{P} \times 2^{\mathcal{P} \times \mathcal{P}} \rightarrow 2^{\mathcal{P} \times \mathcal{P}}$ calculates all possible linearization steps in the neighborhood of a process.

$$findLin(p, y) = \{(q, r) | q, r \in y \wedge q < r \wedge (q, r \in LeftN(p, y) \vee RightN(p, y))\}$$

The function *select* : $2^{\mathcal{P} \times \mathcal{P}} \rightarrow (\mathcal{P} \times \mathcal{P})$ returns one of these linearization steps

$$select(y) = \begin{cases} \perp & \text{if } y = \emptyset \\ \varepsilon x . x \in y & \text{if } y \neq \emptyset \end{cases}$$

Configuration (Standardform)

Configuration in Standardform

$$Alg_{all}(P, P', nb, Msgs, add) = \prod_{j \in P} Alg(j, nb(j)) \mid \prod_{j \in P'} Alg'(j, nb(j), add(j)) \mid \prod_{(j,k) \in Msgs} \bar{j}\langle k \rangle$$

$\mathcal{P}$ be the *set of unique identifiers*,

$P, P' \subseteq \mathcal{P}$ with $P \cup P' = \mathcal{P}$ and $P \cap P' = \emptyset$,

$nb : \mathcal{P} \to 2^{\mathcal{P}}$ a neighborhood-function

$Msgs \in \mathbb{N}^{\mathcal{P} \times \mathcal{P}}$ a multiset of the messages in transit and

$add : \mathcal{P} \rightharpoonup \mathcal{P}$ a partial function with $\forall p \in P'.\exists q \in \mathcal{P}.(p, q) \in add$ and $\forall p \in P.\forall q \in \mathcal{P}.(p, q) \notin add$ that describes the adding in progress

Topologies

Network Topology Graph

Let $A \equiv Alg_{all}(P, P', nb, Msgs, add)$ be an arbitrary configuration. Then the *(directed) network topology graph* $T(A) = (V, E)$ is defined as follows:

$$V = P \cup P' = \mathcal{P} \quad \text{and} \quad E = \{(p, q) | p, q \in V \wedge q \in nb(p)\}$$

Network Topology Graph with Messages

Let $A \equiv Alg_{all}(P, P', nb, Msgs, add)$ be an arbitrary configuration. Then the *(directed) network topology graph with messages* $T^M(A) = (V, E)$ is defined as follows:

$$V = P \cup P' = \mathcal{P} \quad \text{and}$$
$$E = \{(p, q) | p, q \in V \wedge (q \in nb(p) \vee (p, q) \in Msgs \vee add(p) = q)\}$$

Undirected Topologies

Undirected Topology Graph

Let $A \equiv Alg_{all}(P, P', nb, Msgs, add)$ be an arbitrary configuration. Then the *undirected network topology graph* $U(A) = (V, E)$ is defined as follows:

$$V = P \cup P' = \mathcal{P} \quad \text{and} \quad E = \{\{p, q\} | p, q \in V \land q \in nb(p)\}$$

Undirected Topology Graph with Messages

Let $A \equiv Alg_{all}(P, P', nb, Msgs, add)$ be an arbitrary configuration. Then the *undirected network topology graph with messages* $U^M(A) = (V, E)$ is defined as follows:

$$V = P \cup P' = \mathcal{P} \quad \text{and}$$
$$E = \{\{p, q\} | p, q \in V \land (q \in nb(p) \lor (p, q) \in Msgs \lor add(p) = q)\}$$

Potential Functions I

Potential Function Ψ_p for Processes

Let $p \in \mathcal{P}$ be an arbitrary process and A be a configuration. Let $Rec : (\mathcal{T} \times \mathcal{P}) \to \mathbb{N}^{\mathcal{P}}$ with

$$Rec(A, p) = \{\!| q \in \mathcal{P} | (p, q) \in Msgs_A \land q \notin \{succ(p), pred(p)\} |\!\}$$

multiset of all process ids to p still in transit and not a desired neighbor and $adding : (\mathcal{T} \times \mathcal{P}) \to \mathbb{N}$ with

$$adding(A, p) = \begin{cases} dist(p, q), & \text{if } add_A(p) = q \in \mathcal{P} \land q \notin \{succ(p), pred(p)\} \\ 0, & \text{otherwise} \end{cases}$$

The *potential function* $\Psi_p : (\mathcal{T} \times \mathcal{P}) \to \mathbb{N}$ sums up the distances of all outgoing connections of the process p while ignoring desired connections:

$$\Psi_p(A, p) = \sum_{q \in (nb_A(p) \setminus \{succ(p), pred(p)\})} dist(p, q) + \sum_{q \in Rec(A,p)} dist(p, q) + adding(A, p)$$

Potential Functions II

Potential Function Ψ for Configurations

Let A be a configuration. The *potential function for configurations* $\Psi : \mathcal{T} \to \mathbb{N}$ sums up the distances between all the connections of processes while ignoring desired connections:

$$\Psi(A) = \sum_{p \in \mathcal{P}} \Psi_p(A, p)$$