Synchronous and deterministic distributed communication computing model by message passing

Janna Burman
janna.burman@lisn.fr

Reference book: Distributed Algorithms N. Lynch

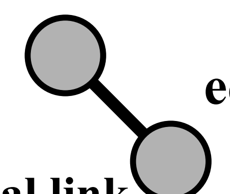
Communication network graph (1)

Represented by a directed graph ***G(V,E)***:

– the set ***V*** of nodes in the network represents the computing and communicating entities, called ***processes***

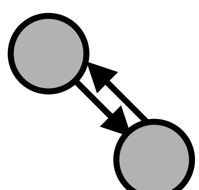
– the set ***E*** = {(i,j) : i,j ∈ V} of directed edges represents the unidirectional **communication links/channels**

– |V| = ***n***

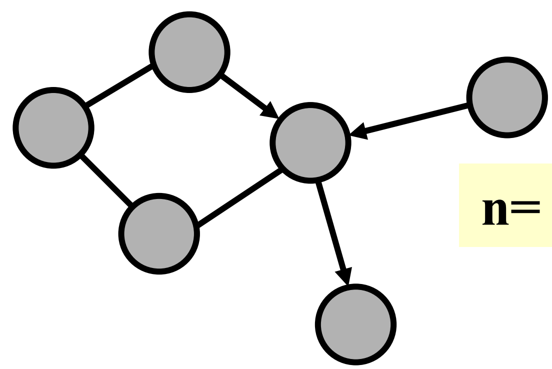


bidirectional link

eq.



2 unidirectional links



n= 6

Communication network graph (2)

- Each directed edge in E (representing a communication link) may contain:
  - either a single message of the alphabet (set) ***M*** (at a time)
  - or an empty message named ***null***

- For each process $p_{i,}$ we denote by:
  - ***out_nbrs**$_i$* the set of successor nodes of p_i $\{\mathbf{j}: (i,\mathbf{j}) \in E\}$
  - ***in_nbrs**$_i$* the set of predecessor nodes of p_i $\{\mathbf{j}: (\mathbf{j},i) \in E\}$
  - In a non-directed graph G: $out\_nbrs_i = in\_nbrs_i =$ ***nbrs**$_i$*

A p_i process consists of:

$states_i$: set of all possible **states of p_i**

- can be infinite
- defined by the program (algorithm) variable types of p_i

$start_i$ $\subseteq$ $states_i$: set of **initial states**

$msgs_i$: **message generation function**

$msgs_i : states_i \times out\_nbrs_i \rightarrow M \cup \{null\}$

$trans_i$: **transition function**

$trans_i : states_i \times [M \cup \{null\}]^{in\_nbrs_i} \rightarrow states_i$

Note: functions **msgs** and **trans** are single-valued in output →

- computation model (transitions) is **deterministic**
- from a given initial configuration (for a given G) the **execution is unique**

Synchronous execution

Starts with:

- processes in the **initial states**
- they **wake up simultaneously**
- the **communication** links are **empty**

Afterwards, the processes operate together in synchronous rounds (phases)

In each round, each process p_i:

1. **Apply $msgs_i$** to generate new messages on each link to out_nbrs_i and put these messages on the corresponding links.
2. **Apply $trans_i$** on the current state and the messages in the in_nbrs_i links to switch to the new state. Cleans up the links (to out_nbrs_i) of any messages.

A **round** (the 2 steps) is assumed to execute **without interruption, i.e. instantaneously.**

An ***execution*** is an infinite repetition of rounds.

Why a **perfectly synchronous model**?

1. Ease of design and analysis
    - single execution for a given initial configuration and graph
2. There are synchronisers
    - algorithms that simulate a synchronous network on an asynchronous one, thus
    - allowing synchronous algorithms to be executed on asynchronous networks
3. Useful for impossibility proofs
    - the set of all asynchronous executions includes all synchronous executions →
    - impossibility results for synchronous networks apply for asynchronous ones

Formal definitions

***Configuration*:** is a vector of states of all processes $[s_1, s_2, s_3, \ldots, s_n]$ (or a assignment of states to processes C: V → states)

Execution: is an **infinite sequence $C_0, M_1, C_1, M_2, C_2, M_3, C_3, \ldots$** in which:

- $\mathbf{C_r}$ is a configuration after round r
- $\mathbf{M_r}$ is a vector of messages on all communication links (directed edges), or an assignment of messages to links ($M_r : E \rightarrow M \cup \{null\}$)

***Terminal configuration*:** is a configuration such that for any process p_i :

1. application of $trans_i$ does not change the state of p_i (for every p_i), and
2. $msgs_i$ generates only null messages

Termination is said to be reached when a terminal configuration is reached

Problem is a predicate P on the executions (which defines the set of admissible executions)

Algorithm/System A solves a problem P if any execution of A satisfies the predicate P (is in the set P)

Complexity measures

Worst case (default)

- **Time complexity**: in (the maximum) number of rounds to termination
- **Message complexity**: in (the maximum) number of non-null messages sent (generated) during an execution
- **Bit complexity**: in number of generated bits

Algorithm complexities are generally expressed:

- **in terms (or as a function) of the parameters that describe the size of the problem**:
  n, |E|, etc.
- **in order of magnitude**

  Let f and g be two functions from integers to integers.

  - **f = O(g),** if there is a constant c>0 s.t. for all n from n>n', f(n) <= c g(n).
  - **f = Ω(g),** if there is a constant c>0 s.t. for all n from n>n', f(n) >= c g(n).
  - **f = Θ (g),** if f is simultaneously O(g) and Ω(g).