Synchronous Network Model

Paulo Sérgio Almeida

Distributed Systems Group
Departamento de Informática
Universidade do Minho

Synchronous network system

- Collection of processes at nodes of a directed graph;
- Start with some initial state;
- Can send message to neighbors along edges (channels);
- Can receive messages from neighbors;
- Proceed in lockstep doing rounds;

Notation

- Directed graph $G = (V, E)$;
- $n = |V|$: size of network;
- out_i: outgoing neighbors;
- in_i: incoming neighbors;
- $nbrs_i$: neighbors; under bidirectional edges (undirected graph);
- $distance(i, j)$: lenght of shortest directed path;
- $diam$: network diameter – maximum $distance(i, j)$ for all i, j;
- M: message alphabet; *null* = no message;

Processes

Components of a process $i \in V$:

- $states_i$: set of states (possibly infinite);
- $start_i$: set of possible starting states (non-empty);
- $msgs_i$: message-generating function

$$states_i \times out_i \rightarrow M \cup \{null\}$$

- $trans_i$: state-transition function

$$states_i \times (M \cup \{null\})^{|in_i|} \rightarrow states_i$$

Rounds and execution

- Execution starts with:
  - processes in some start state;
  - channels empty;
- Processes repeat rounds in lockstep, consisting of two steps:
  1. apply message-generating function to compute messages to all neighbors; put them in channels;
  2. apply state-transition function to state and incoming messages to compute new state; remove messages from channels;
- Model is deterministic; starting states determine all execution;

Halting

- A process halting can be modeled by having *halting states*;
- A process in a halting state:
  - does not send messages;
  - transits to the same state;
- Here we have node-specific halting states; not the system wide halting state of traditional finite-state automata;

Different start times

- It can be useful to have processes start at different times;
- Can be modeled by:
  - adding an extra *environment* node, with edges to normal nodes;
  - environment process sends *wakeup* messages when desired;
  - processes start in *quiescent* states; do not send messages;
  - they change state when receiving some wakeup or other message;

Failures

- Types of failure: process failure and channel failure;
- Process stopping failure:
  - a process can stop somewhere in its execution;
  - can stop after sending a *subset* of the messages it was supposed to;
- Process Byzantine failure:
  - can start sending next messages in arbitrary ways, not following its specification;
- Channel failures:
  - channels can fail by losing messages (some message placed in a channel in step 1 of a round are cleared before step 2);

Inputs and outputs

- Inputs are just possible values in designated *input variables*;
- Outputs are values in *output variables*:
  - these are write-once variables, recording the first write operation;
  - can be read multiple times;

Executions

- State assignment: assignment of a state to each process;
- Message assignment: assignment of a message (or *null*) to each channel;
- Execution: infinite sequence $C_0, M_1, N_1, C_1, M_2, N_2, C_2, \ldots$
  - C_i state assignment after round i;
  - M_i message assignment; messages sent in round i;
  - N_i message assignment; messages received in round i;
  - $M_i \neq N_i$ if there is message loss;
- Executions e and e' are *indistinguishable* to process i, denoted $e \stackrel{i}{\sim} e'$, if i has the same sequence of states, outgoing and incoming messages in e and e';
- Executions can also be said to be indistinguishable to process i up to r rounds.

Proof methods

- Invariant assertions:
  - property of the system state that is true in every execution, after every round;
  - can involve the number of completed rounds;
  - can be proven by induction on the number of completed rounds;
- Simulations:
  - correspondence between algorithm *A* and *B*;
  - *A* produces the same input/output behavior as *B*;
  - expressed by an assertion relating states of *A* and *B* (when both are started with same inputs and run with same failure pattern);

Complexity measures

- Time complexity:
  - number of rounds until output produced or processes halt;
- Communication complexity:
  - total number of (non *null*) messages sent;
  - eventually also number of bits in messages;
- Time is more important in practice;

Randomization

- It can be useful to allow random choices;
- Model is augmented with *random function*:
  - $rand_i$ is added for each node i;
  - $rand_i(s)$, for state s, is a probability distribution over a subset of $states_i$;
- Each round starts now by a random choice of new a state;
- Executions become $C_0, D_1, M_1, N_1, C_1, D_2, M_2, N_2, C_2, \ldots$
  - where D_r represents state assignment after random choices in round r;
- In randomized systems, claims become probabilistic;