Massachusetts Institute of Technology | Lecture 16
6.895: Advanced Distributed Algorithms | October 31, 2002
Professor Nancy Lynch

Self-stabilization

Papers

- Shmuel Katz, Kenneth J. Perry: Self-Stabilizing Extensions for Message-Passing Systems. Distributed Computing 7(1): 17-26 (1993). Conference version, PODC, 1990
- Baruch Awerbuch, Boaz Patt-Shamir, George Varghese, and Shlomi Dolev. Self-stabilization by local checking and network reset. Proceedings of the 8th International Workshop on Distributed Algorithms, WDAG'94, 1994.
- Baruch Awerbuch, Boaz Patt-Shamir, and George Varghese. Self-stabilization by local checking and correction.Based on FOCS 91 paper and Ph.D. thesis. .ps

Recap from last time: 1. Never underestimate the importance of a good introduction (and the intellectual effort required to write it). 2. Dijkstra won the "most influential paper award" last PODC; the award is now renamed after him.

1 Message Passing: Bellman-Ford tree construction

Bellman-Ford is easily self-stabilizing if we have a fixed leader).

Source: $d \leftarrow 0$ Other nodes: $d_v \leftarrow 1 + \min\{d_u \mid u \text{ is a neighbor of } v\}$.

Proof sketch for the synchronous model:

Lemma 1: After i time units, for any node v, $d_v \geq \min(i, distance(s,v))$ Proof: by induction on i. Base: $i = 0$. Step: Assume that $d_u \geq \min(i, distance(s,u))$. Consider v: If it assigns $d > i+1$, we are done. Otherwise, $d < i+1$. In this case v has a neighbor u with $d_u = d-1$ in the previous step, and by induction $distance(s,u) \leq d-1$, hence $distance(s,v) \leq d$. QED

Lemma 2: At time t, if $distance(s,v) \leq t$ then $d_v \leq distance(s,v)$. Proof: By induction on the distances. Base: $t = 0$, node is the source, trivial. Step: Suppose lemma holds for all nodes at distance d and all times after t. Consider a node v at distance $d+1$: it has a neighbor at distance d, which by induction has its variable to be at most d by time d, hence d_v is set to at most d+1 at any time from $t+1$ and on. QED

2 General stabilization transformers: Katz & Perry

This paper is the first to undertake the task of making an *arbitrary* message-passing system self-stabilizing. They give first answers to the following questions:

1. What does this mean, really? 2. When can this be done? 3. How to do it?

The answers to 1 and 3 present an approach which was later improved (see today). The answer to 2 is quite simple. Nevertheless, this paper is a major breakthrough in the theory of self-stabilization.

2.1 What is self-stabilization, revisited

Model: a network of processors connected by perfect channels (standard asynchronous model).

They distinguish between variables and location counter in processors, but this is equivalent to assuming that the processors are state machines.

The state of a channel is just the queue of messages it stores.

A global state is a vector of all local states of processors and channels.

An execution is a sequence of global states, where each state is obtained from its predecessor by applying one local transition.

The set of all executions for a protocol P: $sem(P)$. No initial state!

A predicate over $sem(P)$ defines the set of legal executions, called $legsem(P)$. A state that occurs in any execution is $legsem(P)$ is called *legal state*.

Convenient special case: $legsem(P)$ can be defined using *initial states*. (Get closure immediately).

DEF: P is self-stabilizing if for every execution $e \in sem(P)$ there exists an execution in $e' \in legsem(P)$ such that e and e' have a common suffix.

DEF: Q extends P if each legal state of Q is an extension of a legal state of P, and the projected legal executions of Q are exactly $legsem(P)$.

Intention: additional variables. (they say no location counter correspondence) Cleaner: a mapping. Cleanest: Implementation relation...

DEF: Q is a **self-stabilizing extension** of P if Q extends P and Q is self-stabilizing.

Nancy's definition: Problem is a set of *behaviors*. A system is self-stabilizing solution for a given problem if every execution has a suffix whose behavior is a suffix of the legal behaviors.

2.2 When can we make a system self-stabilizing?

First, some general observations:

* (Barring trivial cases) a local state can't indicate that the global state is legal. * any non-trivial self-stabilizing program can never halt. (non-trivial halting means that there are good and bad pasts.) Model a terminating state by repeating it indefinitely.

THM: Let Q be a s.s.e. of P, and let A be a predicate over executions suffixes of P. A holds for every suffix of executions of Q iff A is true i.e. Pf: every suffix of an execution of Q is an execution of Q. Use projection.

Moral: this is the way self-stabilizing specification should be written. Example: mutual exclusion. Can we require the program *never* to break ME? Can be broken in the initial state!

2.3 Algorithm

Idea: run a control program in parallel, that verifies that the given program is correct, and re-initializes it if necessary. Specifically: * From time to time a leader initiates a snapshot protocol. * When collected, state is checked again the legality predicate. * To reset the system, use the snapshot mechanism to *write* instead of read!

technical challenge: a self-stabilizing snapshot algorithm. Basis: Chandy-Lamport. Problem: distinguish between different incarnations. Solution: unbounded counters attached to tokens. Bounded solution: Assume that the total number of messages in the system cannot be more than some bound N.