Computer Networks

EDA387/DIT663

Fault-tolerant Algorithms for Computer Networks

Maximal Matching (Ch. 2)

Today

- Proof Techniques

Proof Techniques: Variant Function

$$c \xrightarrow{\text{step}} c_1 \xrightarrow{\text{step}} c_2 \xrightarrow{\text{step}} c_3 \xrightarrow{\text{steps}} c_{safe}$$

$$|VF(c)| \geq |VF(c_1)| \geq |VF(c_2)| \geq |VF(c_3)| \geq \ldots \geq |VF(c_{safe})| \geq \ldots \geq \text{bound}$$

- Used for proving convergence
- Can be used to estimate the number of steps required to reach a safe configuration

Variant Function - Example: self stabilizing Maximal Matching

- *Matching* is a set of edges without common nodes

- The algorithm should reach a configuration in which $pointer_i = j$ implies that $pointer_j = i$

- We will assume the existence of a central daemon

- The set of legal executions MM for the maximal matching task includes every execution in which the values of the pointers of all the processors are fixed and form a maximal matching

Variant Function - Example: self stabilizing Maximal Matching- definitions

Program for p_i :

do forever

  if $pointer_i = null$ **and** $(\exists\ p_j \in N(i) \mid pointer_j = i)$ **then**

    $\boxed{pointer_i = j}$ matched

free **if** $pointer_i = null$ **and** $(\forall\ p_j \in N(i) \mid pointer_j \neq i)$ **and**

      $(\exists\ p_j \in N(i) \mid pointer_j = null)$ **then**

    $\boxed{pointer_i = j}$ waiting

  if $pointer_i = j$ **and** $pointer_j = k$ **and** $k \neq i$ **then** chaining

    $pointer_i = null$

od

single

if pointer p_i =null and pointer p_j =k and k != i then pointer p_i = null
singel

matched, waiting, free and chaining

- **matched** in c_l, if p_i has a neighbor p_j such that $pointer_i = j$ and $pointer_j = i$
- **waiting** in c_l, if p_i, has a neighbor p_j such that $pointer_i = j$ and $pointer_j = null$
- **free** in c_l, if $pointer_i = null$ and there exists a neighbor p_j such that $pointer_j = null$
- **single** in c_l, if $pointer_i = null$ and there is no neighbor p_j such that $pointer_j = null$
- **chaining** in c_l, if there exists a neighbor p_j for which $pointers_i = j$ and $pointer_j = k$, $k \neq j$

Variant Function - Example: self stabilizing Maximal Matching- proving correctness

- The variant function VF(c) returns a vector (m+s,w,f,c)

  m - matched, s – single, w – waiting,
  f – free, c - chaining

- Values of VF are compared lexicographically

- VF(c) = (n,0,0,0) $\Leftrightarrow$ c is a safe configuration with relation to MM and to our algorithm

- Can you show that: Once a system reaches a safe configuration, no processor changes the value of its pointer?

Variant Function - Example: self stabilizing Maximal Matching- proving correctness

- In every non-safe configuration, there exists at least one processor that can change the value of its pointer

- Can you show that: Every change of a pointer-value increases the value of VF?

- The number of such pointer-value changes is bounded by the number of all possible vector values.

- The first three elements of the vector (m+s,w,f,c)

  imply the value of c, thus there at most $O(n^3)$ changes.

Proof

- An assignment in line 3 of the code reduces the number of free processors and waiting processors by 1 and increments the number of matched processors by 2.
- An assignment in line 6 of the code reduces the number of free processors by 1 and increments the number of waiting processors by 1.
- The assignment in line 8 is executed when p_i is chaining.

Proof

- Two cases are considered: first, if no neighboring processor points to p_i.
- In this case, p_i changes status to free if there exists an unmatched neighbor, or to single if all neighbors are matched.
- Therefore, the number of chaining processors is reduced by 1 and the number of free or single processors is incremented by 1.

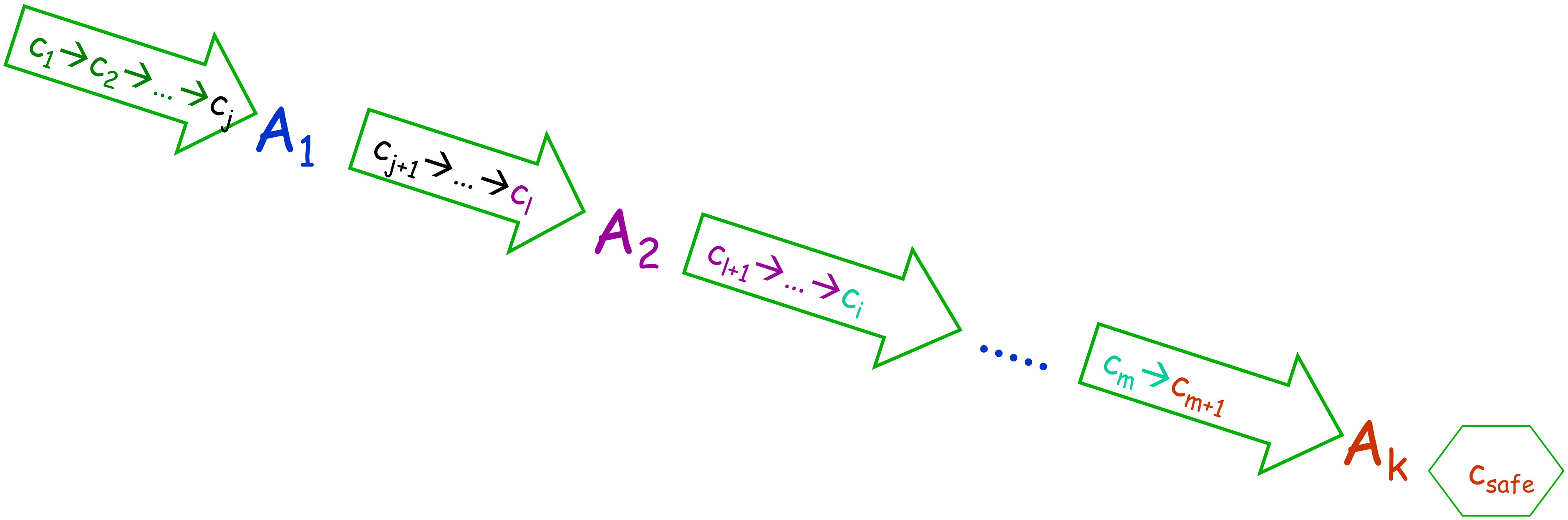
Proof

- In the 2nd case, when at least one neighbor p_ℓ points toward p_i, the status of p_i is changed to free and the status of p_ℓ is changed from chaining to waiting.
- Hence the number of chaining processors is reduced by 2, while the number of both free and waiting processors is incremented by 1.
  - Thus each assignment increments the value of VF.
- The system stabilizes once it reaches a configuration in which no increment is possible, which is a safe configuration.

Proof

- The number of such pointer-value changes is bounded by the number of all possible vector values.
  - The fact that $m + s + w + f + c = n$ implies that the number of possible vector values is $O(n^3)$.
  - A rough analysis uses the following argument.
  - One can choose $n + 1$ possible values for $m + s$ and then $n + 1$ values for w and f.
  - The value of n and the first three elements of the vector $(m + s, w, f, c)$ imply the value of c.
- Therefore the system reaches a safe configuration within $O(n^3)$ pointer-value changes.

Convergence Stairs



- A_i – predicate
- for every $1 \leq i < k$, A_{i+1} is a refinement of A_i

Convergence Stairs - Example: Leader election in a General Communication Network

Program for p_i, each processor reads it's neighbors leader and chooses the candidate with the lowest value :

```
do forever
    ⟨candidate,distance⟩ := ⟨ID(i), 0⟩
    forall P_j ∈ N(i) do
        begin
            ⟨leader_i[j],dis_i[j]⟩ := read⟨ leader_j, dis_j ⟩
            if (dis_i[j] < N) and ((leader_i[j] < candidate) or
                ((leader_i[j] = candidate) and (dis_i[j] < distance))) then
                    ⟨candidate,distance⟩ := ⟨ leader_i[j],dis_i[j] + 1⟩
        end
    write ⟨leader_i ,dis_i⟩ := ⟨candidate,distance⟩
od
```

Convergence Stairs - Example:
Leader election in a General Communication Networks

- We assume that every processor has a unique identifier in the range 1 to N

- The leader election task is to inform every processor of the identifier of a single processor in the system, this single processor is the leader

- Floating identifier - an identifier that appears in the initial configuration, when no processor in the system with this identifier appears in the system

Convergence Stairs - Example: Leader election, proving correctness

- We will use 2 convergence stairs :
  - A_1 - no floating identifiers exists
  - A_2 (for a safe configuration) every processor chooses the minimal identifier of a processor in the system as the identifier of the leader
- To show that A_1 holds, we argue that, if a floating identifier exists, then during $O(\Delta)$ rounds, the minimal distance of a floating identifier increases

Convergence Stairs - Example: Leader election, proving correctness ...

- After the first stair only the correct ids exist, so the minimal can be chosen

- From that point, every fair execution that starts from any arbitrary configuration reaches the safe configuration

- Notice : if A_1 wasn't true, we couldn't prove the correctness

Proof

- The proof of correctness uses two convergence stairs.
- The first convergence stair is a predicate A_1 on system configurations verifying that no floating identifier exists.
- The second convergence stair is a predicate A_2 for a safe configuration — a predicate that verifies that every processor chooses the minimal identifier of a processor in the system as the identifier of the leader.

Proof

- The value of a floating identifier can appear in the local arrays of every processor p_i, $l_i[1, \ldots, \delta]$, in the candidate local variable, and in the field $leader_i$ of the communication register.
  - The distance of a floating identifier appearing $l_i[j]$, candidate, or $leader_i$ is $d_i[j]$, distance, or dis_i, respectively.
- To show that the first attractor holds, we argue that, if a floating identifier exists, then during any $O(\Delta)$ rounds, the minimal distance of a floating identifier increases.

Proof

- **Lemma 2.5**: Every fair execution that starts from any arbitrary configuration has a suffix in which no floating identifier exists.
- We first show that, as long as a floating identifier exists, the minimal distance of a floating identifier increases during any $O(\Delta)$ rounds.
- Let p_i be a processor that holds in its local variables or in its communication registers a floating identifier with the minimal distance.

Proof

- Once p_i starts executing the do forever loop, it must choose (either its own identifier for $leader_i$ or) a distance that is at least one greater than the distance read from a neighbor (line 8 of the code).
  – Thus, if p_i chooses to assign a floating identifier to leader u it must choose a distance that is greater by one than the distance it read.
  – Once the minimal distance of a floating identifier reaches N, all processors do not choose a floating identifier.
- Therefore, all the floating identifiers are eventually eliminated. ■

Proof

- The fact that the first predicate A_l holds from some configuration of the system lets us prove the next theorem using arguments similar to those used for the non-stabilizing algorithm.
- **THEOREM 2.3**: Every fair execution that starts from any arbitrary configuration reaches a safe configuration.

Summary

We have looked into some common algorithmic and proof techniques in self-stabilization

Review Questions

1. Design a self-stabilizing mutual exclusion algorithm for a system with processors p_1, p_2,... , p_n that are connected in a line. The leftmost processor p_1 is the special processor. Every processor p_i , 2 ≤ i ≤ n − 1, communicates with its left neighbor p_i−1 and its right neighbor p_i+1. Similarly, p_1 communicates with p_2 and p_n with p_{n-1}.
2. Define a safe configuration for the self-stabilizing synchronous consensus algorithm of figure 2.7.