Static Termination Analysis for Event-driven Distributed Algorithms

Felix Wiemuth [1,2]
Peter Amthor [1]
Winfried E. Kühnhauser [1]

[1] Technische Universität Ilmenau
Ilmenau, Germany

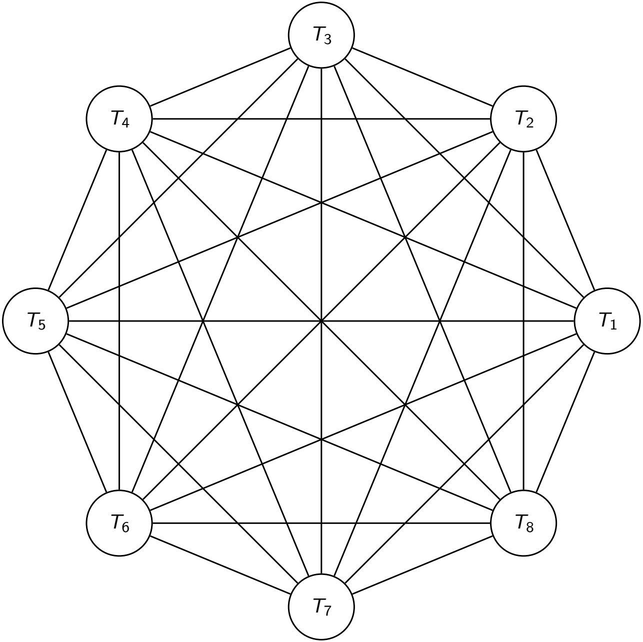
[2] **Concordium Research ApS**
Aarhus, Denmark

June 28, 2019

Overview

- **Termination**
  - Important non-functional property
  - Undecidable
- **Approach**
  - Event-based model
  - Criterion implying termination

Model

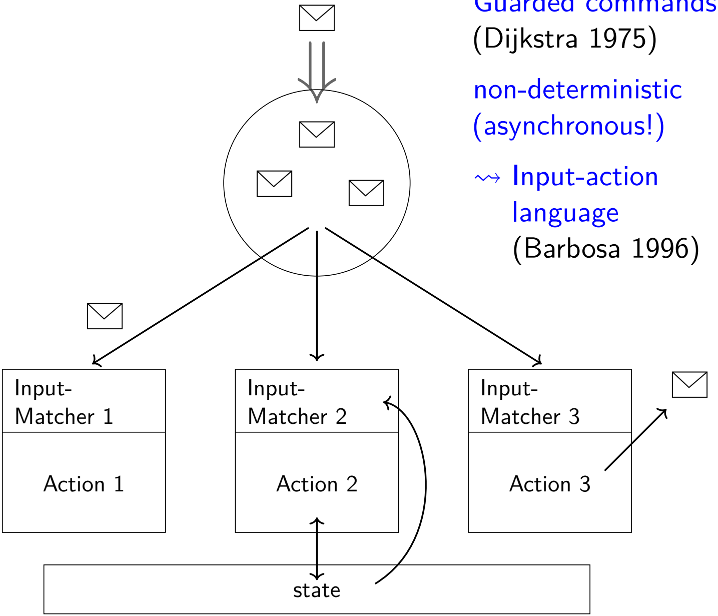


message-based

point-to-point

asynchronous

Model: Tasks



Guarded commands
(Dijkstra 1975)

non-deterministic
(asynchronous!)

$\rightsquigarrow$ Input-action language
(Barbosa 1996)

Two-phase commit protocol

Coordinator: { C }

Variables

```
t // local part of transaction
v // own vote (commit/abort)
count := 0 // commit count
```

```
input init() {
    if v = abort then
        send abort() to P
        t.abort()
    else
        send vote_request() to P
    end
}
```

```
input vote(x) {
    if x = abort then
        send abort() to P
        t.abort()
    else
        count++
        if count = n then
            send commit() to P
            t.commit()
        end
    end
}
```

Participant: $P = \{ P_1, \ldots, P_n \}$

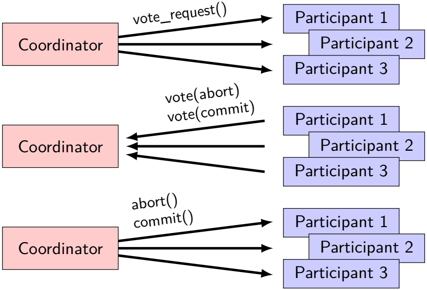
Variables

```
t // local part of transaction
v // own vote (commit/abort)
```

```
input vote_request() {
    reply vote(v)
}
```

```
input abort() {
    t.abort()
}
```

```
input commit() {
    t.commit()
}
```



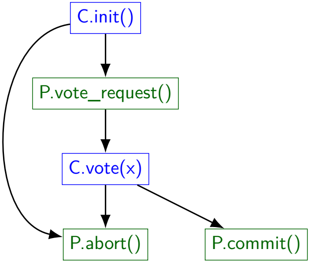
Message flow in the two-phase commit protocol

Coordinator: { C }

```
Variables
    t // local part of transaction
    v // own vote (commit/abort)
    count := 0 // commit count

input init() {
    if v = abort then
        send abort() to P
        t.abort()
    else
        send vote_request() to P
    end
}

input vote(x) {
    if x = abort then
        send abort() to P
        t.abort()
    else
        count++
        if count = n then
            send commit() to P
            t.commit()
        end
    end
}
```

Participant: $P = \{ P_1, \ldots, P_n \}$

```
Variables
    t // local part of transaction
    v // own vote (commit/abort)

input vote_request() {
    reply vote(v)
}

input abort() {
    t.abort()
}

input commit() {
    t.commit()
}
```



Termination criterion

Theorem

If an algorithm's message flow graph is acyclic, then the algorithm always terminates.

Basic assumptions

- All actions terminate
- No spontaneous actions
    - Each action consumes a message

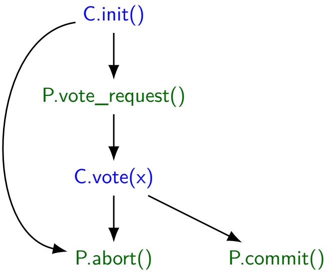
Termination criterion

Theorem

If an algorithm's message flow graph is acyclic, then the algorithm always terminates.

Advantages

- Practical language
- Syntactic criterion
  - Static analysis
  - Efficient: $\mathcal{O}(L + \#IAP^2)$
- Visualization as a tool



Termination criterion

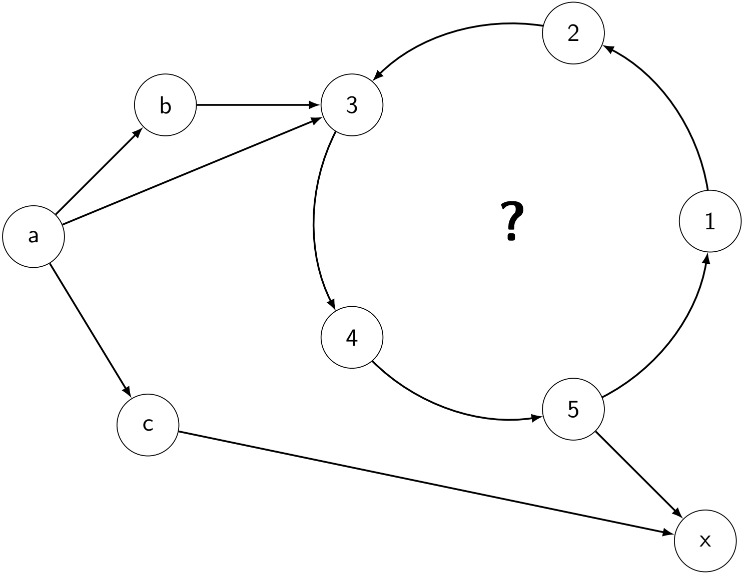
Theorem

If an algorithm's message flow graph is acyclic, then the algorithm always terminates.

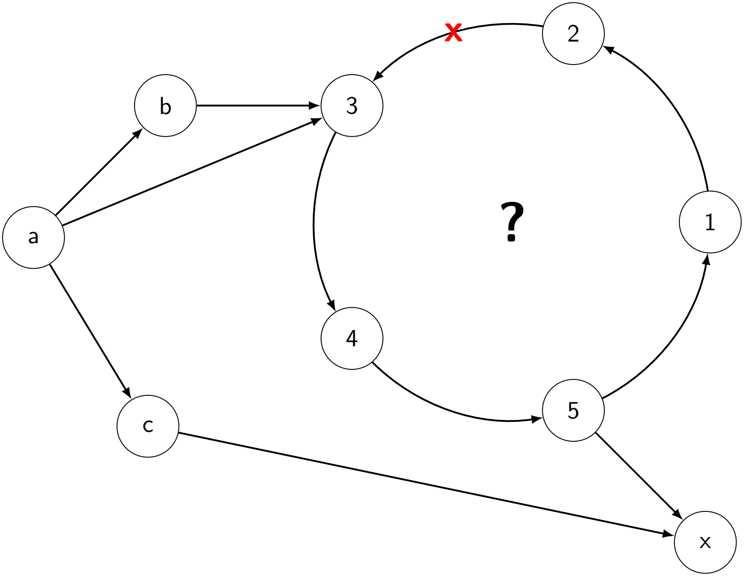
Disadvantages

- No spontaneous actions – timers?
- Precision?
  - Sequential protocols ✓
  - More complicated protocols?

Improving precision



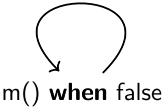
Non-traversable cycles



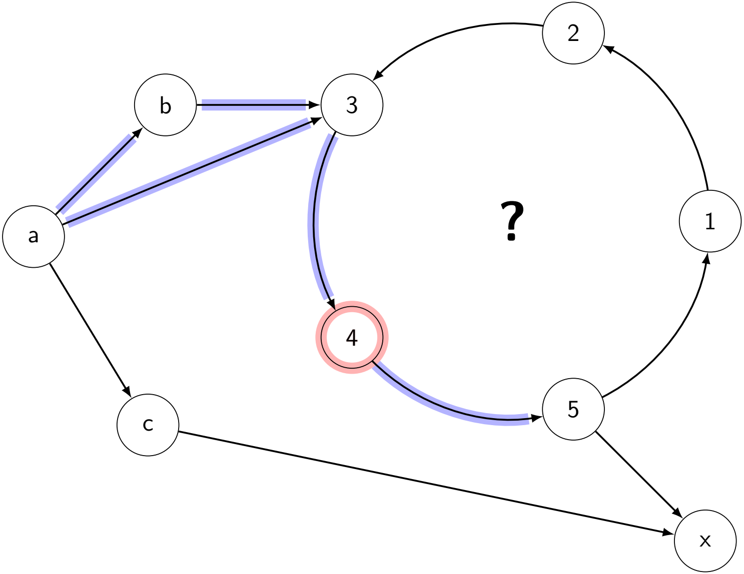
Non-traversable cycles: Impossible message flow

T

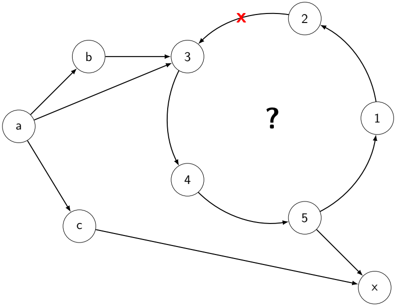
input m() **when** false {
    send m() **to** T
}

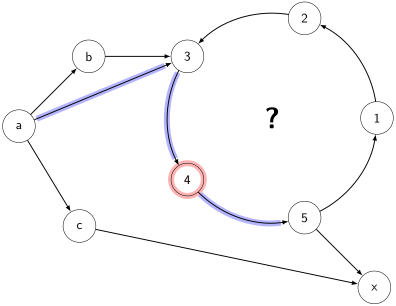


Non-traversable cycles (2)

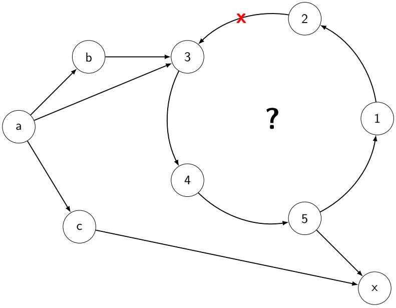


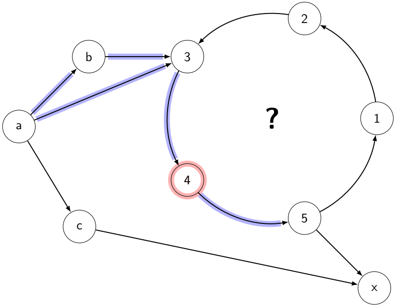
Non-traversable cycles

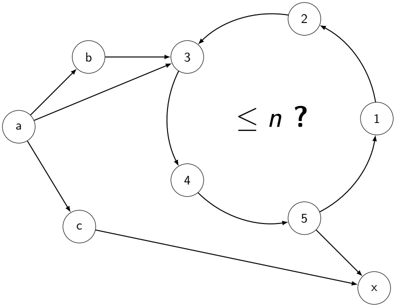




Limited cycles

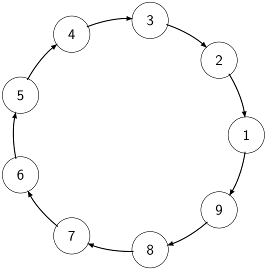


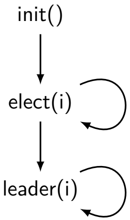




Leader election on a ring (Chang/Roberts 1979)

```
Ring node
Variables
    leader // the current leader
    self, next // own/next ID on ring

input init() {
    send elect(self) to next
}

input elect(i) limit 2 {
    if i = self then
        send leader(i) to next
    else
        send elect(max(i, self)) to next
    end
}

input leader(i) limit 1 {
    leader := i
    if i ≠ self then
        send leader(i) to next
    end
}
```

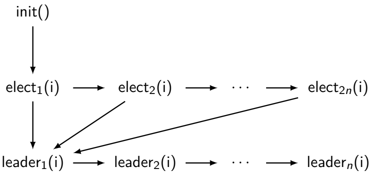




Chang and Roberts ring algorithm – unrolled

Ring node

```
Variables
    leader, self, next

input init() {
    send elect_1(self) to next
}

input elect_1(i) {
    if i = self then
        send leader_1(i) to next
    else
        send elect_2(max(i, self)) to next
    end
}

input elect_2(i) {
    if i = self then
        send leader_1(i) to next
    else
        send elect_3(max(i, self)) to next
    end
}

...

input elect_2n(i) {
    if i = self then
        send leader_1(i) to next
    else
        error "Limit exceeded"
    end
}

input leader_1(i) {
    leader := i
    if i ≠ self then
        send leader_2(i) to next
    end
}

input leader_2(i) {
    leader := i
    if i ≠ self then
        send leader_3(i) to next
    end
}

...

input leader_n(i) {
    leader := i
    if i ≠ self then
        error "Limit exceeded"
    end
}
```



Summary

- **Goal:** Static termination analysis for distributed algorithms
- **Approach:**
  - Event-driven model
  - Analyze possible communication between input-action pairs
    $\rightsquigarrow$ Message flow graph
- **Result:** Syntactic termination criterion
  - Acyclicity implies termination
- Improving precision
- **Conclusion:** Framework for static termination analysis
- Implementation: https://github.com/felixwiemuth/JIAL