Algorithms for Distributed Systems under Wireless Sensor Networks

www.ceu-lang.org

Overview

- Library of DS algorithms
  - topology, broadcast, leader, etc
- Wireless Sensor Networks
  - unreliable communication
  - hundreds of nodes

- *Abstractions in Céu?*
- *Simulation in Céu?*

Abstractions

Overview of Céu

- Sequences
  - `… ; S1 ; S2 ; S3 ; …`
- Await
  - `… ; S1 ; await E1 ; S2 ; await E2 ; …`
- Parallelism
  -
```
par do
    … ; S11 ; await E11 ; S12 ; await E12 ; …
with
    … ; S21 ; await E21 ; S22 ; await E22 ; …
end
```

Abstractions

```
par do
    … ; S11 ; await E11 ; S12 ; await E12 ; …
with
    … ; S21 ; await E21 ; S22 ; await E22 ; …
end
```

```
function f1 = { S ; await E; }
par do
    … ; S11 ; await E11 ; f1(); …
with
    … ; f1(); S22 ; await E22 ; …
end
```

Macros

```
macro f1 = { S ; await E; }
par do
    … ; S11 ; await E11 ; f1(); …
with
    … ; f1(); S22 ; await E22 ; …
end
/************/
par do
    … ; S11 ; await E11 ; S ; await E ; …
with
    … ; S ; await E ; S22 ; await E22 ; …
end
```

Macros in Céu

- Text substitution prior to Céu compilation
  - ”macro expansion”
  - **GNU m4**
- Advantages
  - minor changes to the language
  - Céu remains static (analysis + faster)
- Disadvantages
  - duplicates code (more memory)
  - error messages

```
// app.ceu
include(lib.m4)
par do
    ALIVE(1s);
with
    ...
end
```

```
// lib.m4
define(ALIVE, `/*{-{*/
    loop do
        await $1;
        _printf("I'm alive\n");
    end
/*}-}*/´)
```

// app.ceu_m4

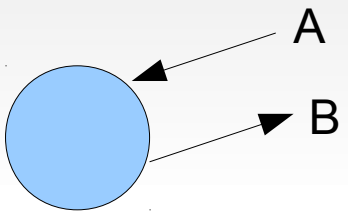
```
par do
    /*{-{*/
loop do
    await 1s;
    _printf(”I'm alive\n”);
end
/*}-}*/;
with
    ...
end
```

Simulation

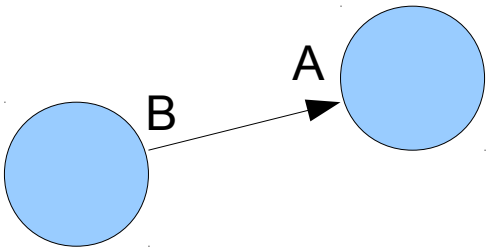
I/O in Céu

- I/O events:
  - interface with the environment

```
input  int A;
output int B;
```



- Multiple programs:
  - each run in an OS process
  - link I/O: app1 B → app2 A
  - communicate through OS queues



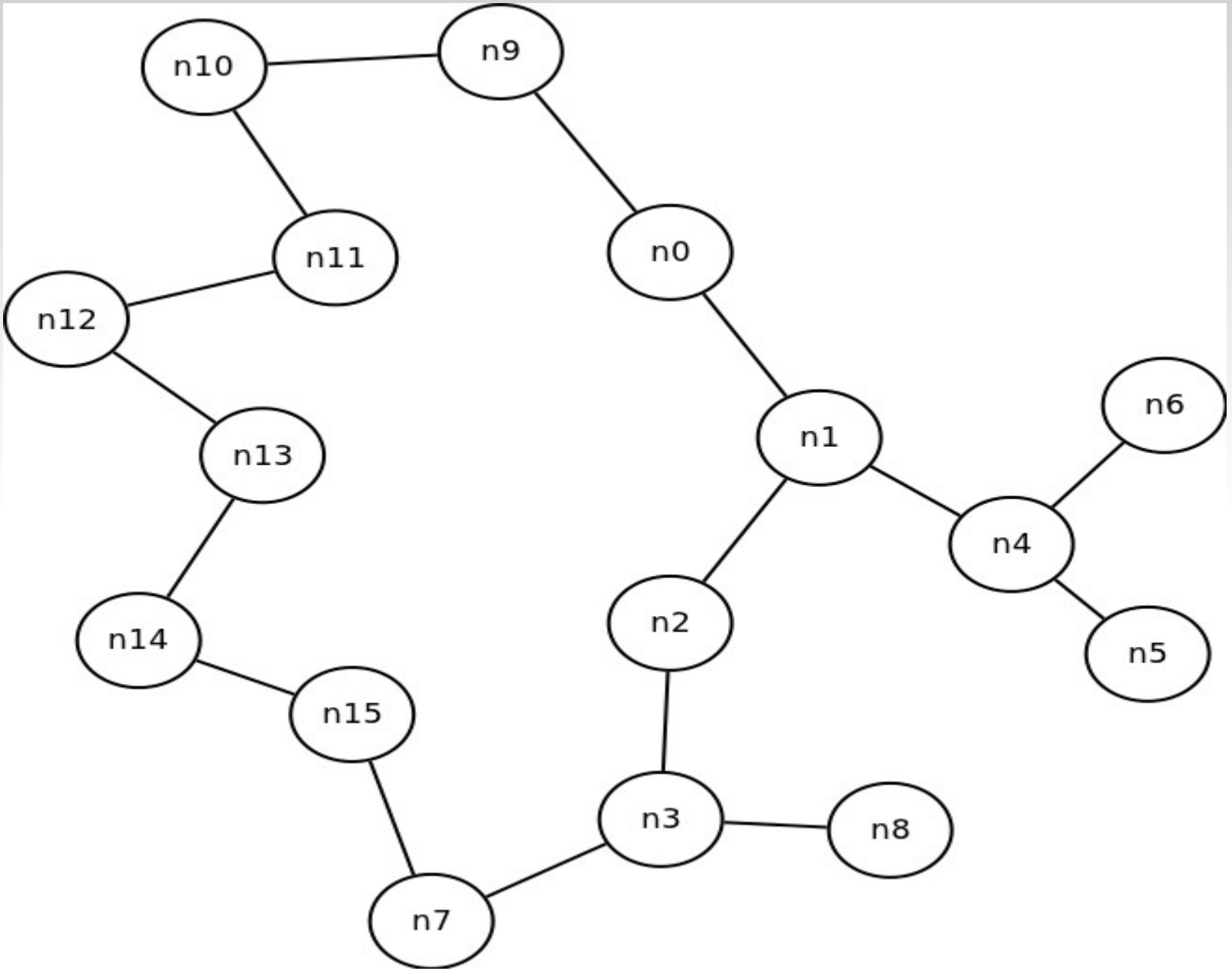
Example

send_recv.lua

DS algorithms

Fault Tolerance

- Sends are unreliable
  - **never** trust it
  - Example: *collisions.lua*
- Buffers may overflow
  - avoid use of buffers


n10
n9
n11
n0
n12
n6
n1
n13
n4
n14
n2
n5
n15
n3
n8
n7

#1 – Neighbours

- Discovers which nodes are within range

```
u8[2] nodes = b0000000000000000;
DS_neighbours(nodes, n_nodes, am_type, retry)
```

- Sends an `am_type` broadcast message every `retry` event
- Receives `am_type` messages / saves src in `nodes`
- It never terminates!

#2 – Topology

- Discovers the topology of the network

```
u8[2x16] nodes = b0000000000000000  // n0
                 b0000000000000000  // n1
                 ...
DS_topology_hb(nodes, n_nodes, am_type, heartbeat)
```

- Heartbeat algorithm
- Sends an `am_type` broadcast message with local knowledge
- Receives `am_type` messages with topology / ORs with local knowledge

#2 – Topology

- First variation:

```
DS_topology_hb_diam(nodes, n_nodes, am_type, hb, diam)
```

  - terminates after `diam` heartbeats

- Second variation:

```
DS_topology_hb_ack(nodes, n_nodes, am_type, hb)
```

  - terminates after
    - all nodes have a neighbour
    - all neighbours acknowledge

#3 – Broadcast

- Broadcast of messages in sequence to all network
  - single sender
  - multiple receivers
- Node broadcasts its frontier periodically to neighbours
- Node re-sends requests from neighbours
- Buffer to store received messages
  - tolerance: sender_period x buffer_size