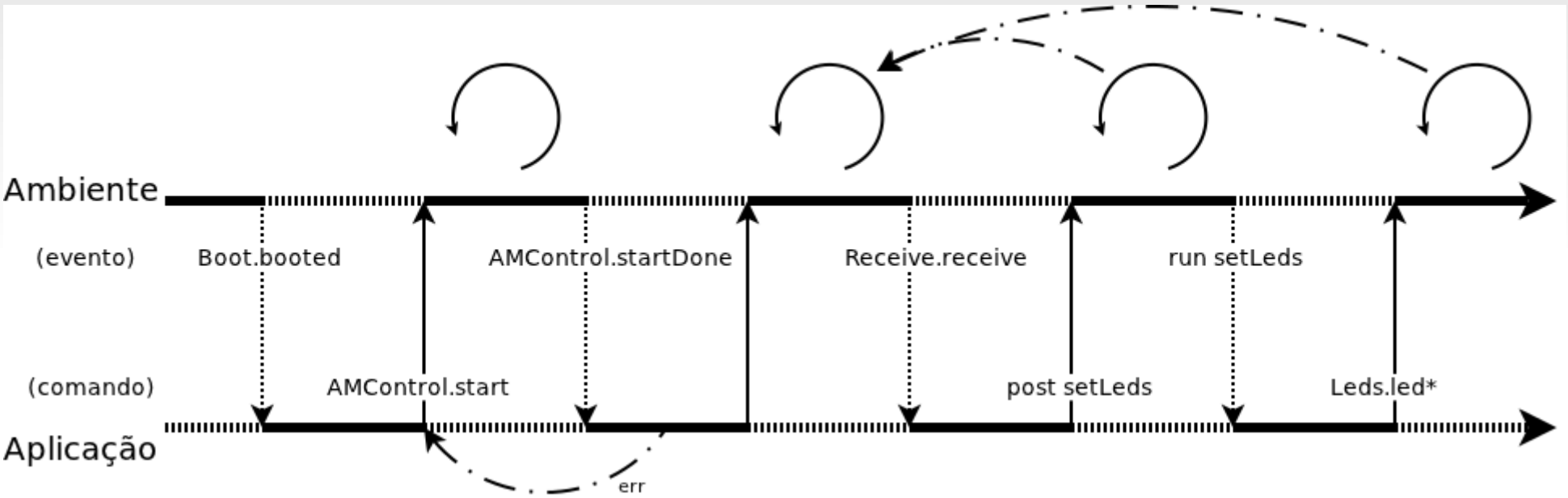
A Safe and Reactive language for Embedded Systems

www.ceu-lang.org

```
par do
    loop do
        await 250ms;
        _Leds_led0Toggle();
    end
with
    loop do
        await 500ms;
        _Leds_led1Toggle();
    end
with
    loop do
        await 1000ms;
        _Leds_led2Toggle();
    end
end
```


Ambiente
(evento)
Boot.booted
AMControl.startDone
Receive.receive
run setLeds
(comando)
AMControl.start
post setLeds
Leds.led*
Aplicação
err

Overview of Céu

- Reactive
  - environment in control: *events*
- Imperative
  - sequences, loops, assignments
- Concurrent
  - multiple lines of execution: *trails*
- Synchronous
  - trails synchronize at each external event
- Deterministic
  - always yields the same outcome for a given timeline

```
input int Photo_readDone;

loop do
    par/or do
        await 250ms;
    with
        _Photo_read();
        int data = await Photo_readDone;

        if data > 700 then
            _Leds_led2On();
        else
            _Leds_led2Off();
        end
        ...   // other leds

        await forever;
    end
end
```

Céu under TinyOS

- Boot
  - 1st line of code
- Timers
  - await 500ms
- Leds
  - _Leds_led0Toggle(), _Leds_led1On(), _Leds_set(), …
- Sensor
  - _Photo_read() → await Photo_readDone
- Radio
  - _Radio_start() → await Radio_startDone
  - _Radio_send() → await Radio_sendDone
  - await Radio_receive

Radio - Init

```
loop do
    int err = _Radio_start();
    if err == _SUCCESS then
        err = await Radio_startDone;
        if err == _SUCCESS then
            break;
        end
    end
    await 1s;
end
```

Synchronous Execution

- Time: discrete sequence of external input events
  - sequence: only one event reacts at a time
  - discrete: a reaction executes in bounded time

1) Await next event and awake awaiting trails.
2) Active trails execute without interruption. (*Reaction Chain)*
3) Goto 1.

Two (possible) problems

1) Unbounded execution

- Breaks the synchronous model

```
int a = 0;
loop do
    a = a + 1;
end
```

```
int sum = 0;
int i = 1;
loop do                  // a tight loop
    sum = sum + i;
    if i == 100 then
        break;
    else
        i = i + 1;  // no await
    end
end
return sum;
```

Two (possible) problems

2) Non-determinism

- Usually undesired

```
input void A;
int v;
par/and do
    await A;
    v = 1;
with
    await A;
    v = 2;
end
return v;
```

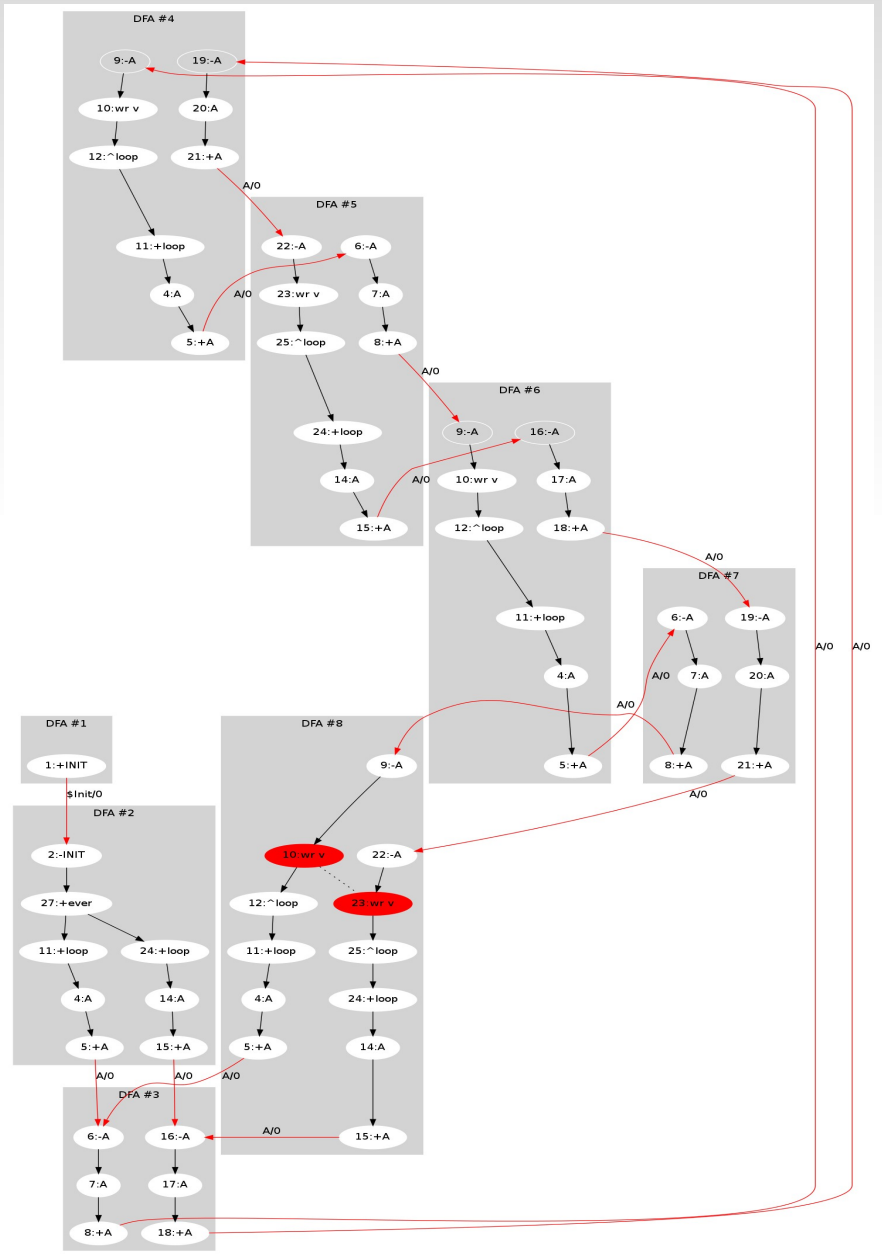
```
input void A;
int v;
par/and do
    await A;
    v = 1;
with
    await A;
    await A;
    v = 2;
end
return v;
```

Céu -> DFA

- Céu programs are converted to DFAs
- Céu is static
- Detects:
  - concurrent access to variables or C functions
  - concurrent escapes
  - unreachable expressions

DFA example

```
input void A;
int v;
par do
    loop do
        await A;
        await A;
        v = 1;
    end
with
    loop do
        await A;
        await A;
        await A;
        v = 2;
    end
end
```



1st class Timers

- ”Wall-clock” time is very common in reactive apps
  - samplings, watchdogs, etc

```
loop do
   await 1h2m3s4ms5us;
   ... // do something
end
```

```
loop do
    par/or do
        // do something
    with
        await 500ms;
    end
end
```

1st class Timers

- 1s + 1s = 2s (!!!) (model and implementation)

```
int v;
par/and do
    await 51us;
    v = v + 1;
    await 49us;
    return 1;
with
    await 49us;
    v = v * 1;
    await 51us;
    return 2;
end
```

Asynchronous blocks

- Execute time consuming operations
  - tight loops

```
int ret = 0;
par/or do
    ret = async do
        int i=1, sum=0;
        loop do
            sum = sum + i;
            if sum == 100 then
                break;
            else
                i = i + 1;
            end
        end
        return sum;
    end;
with
    await 1s;
end
```

Simulation

- Important in cross-compiling platforms
- Simulators
  - Usually inaccurrate
  - Require additional knowledge
  - Vary among platforms
- Céu uses the own language for simulation
  - input events: only source
  - asynchronous blocks can emit input events

Simulation

```
par do
     input int A;
     int v = await A;
     loop do
         await 100ms;
         _printf(”v = %d\n”, v);
         v = v + 1;
     end
with
    async do
        emit A=1;
        emit 1s350ms;
    end
    return 0;
end
```

- Wiki do site:
  - Tutorial
  - Manual
- Try online!