

An Overview of Céu

A synchronous language inspired by Esterel

Francisco Sant'Anna

francisco@ime.uerj.br



“Hello world!” in Céu

■ Blinking a LED

1. *on* ↔ *off* every 500ms
2. *stop* after “*press*”
3. *restart* after 2s

■ Compositions

- seq, loop, par (*trails*)
 - At any level of depth
- ~~state variables / communication~~

```
loop do
  par/or do
    loop do
      await 500ms;
      _led_toggle();
    end
    with
      await PRESS;
    end
  end
  await 2s;
end
```

Lines of execution
=
Trails (in Céu)

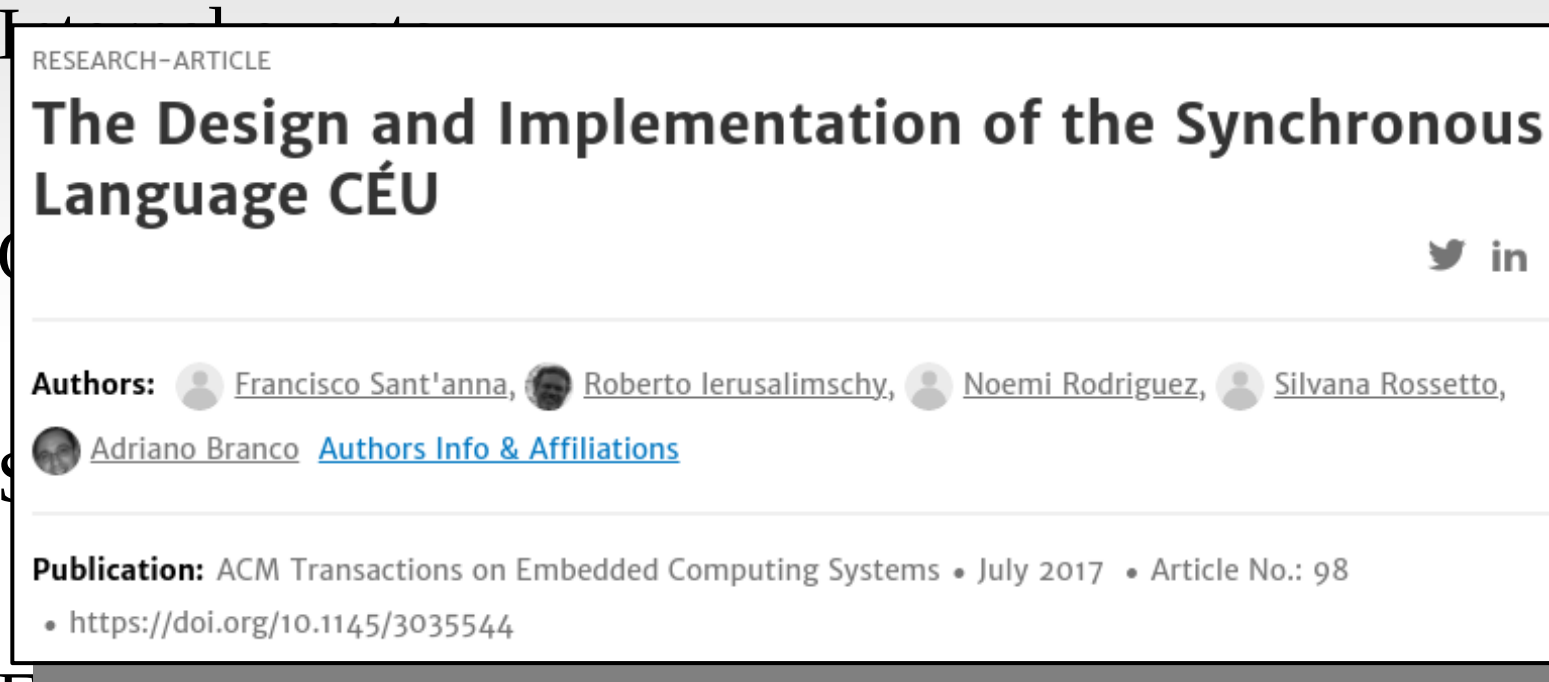
**Céu is heavily inspired
by Esterel**

Céu Peculiarities

1. External events

- notion of time ~ queue of unique events, mutual exclusion

2.



3.

4.

5. First-class timers

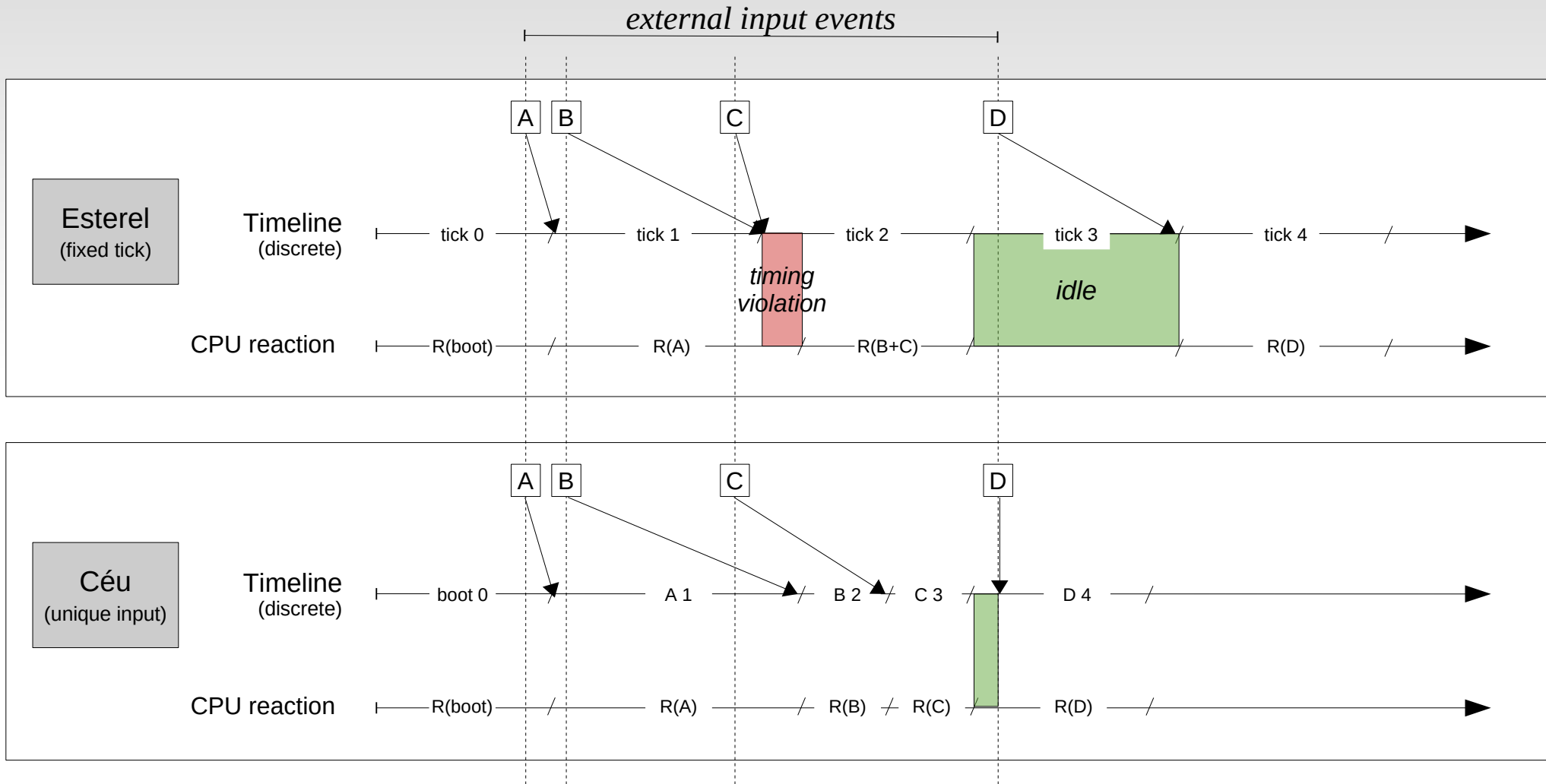
- dedicated syntax, automatic readjustment

6. Dynamic execution

15 min video at ceu-lang.org

- pool allocation, static/lexical memory management

1. External Events



2. Internal Events

- Stack-based execution
 - an **emit** stacks next statement → awakes awaiting trails in an *intra reaction*
 - emits can nest (hence a stack)
- Like function calls
 - but richer: coroutines, resumable exceptions, reactive variables

```
event int* inc;
par/or do
  loop do
    var int* p = await inc; ① ④
    *p = *p + 1; ③
  end
with
  var int v = 1;
  <...>
  emit inc => &v; ②
  assert(v==2); ⑤
end
```

stacked

3. Internal Determinism

- Esterel:

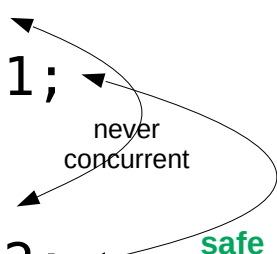
- “if there is no control dependency, as in (call f1()^① || call f2()^②), the order is unspecified and it would be an error to rely on it”
- “if a variable is written by some thread, then it can neither be read nor be written by concurrent threads”

- Céu:

- “when multiple trails are active during the same reaction, they are scheduled in lexical order”
- pragmatic (e.g., `printf`, `redraw`), but fragile

3. Simple Static Checks

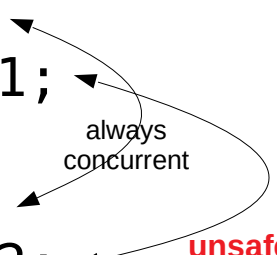
```
input void A, B;  
var int x = 1;  
par/and do  
  await A;  
  x = x + 1;  
with  
  await B;  
  x = x * 2;  
end
```



never concurrent

safe

```
input void A;  
var int y = 1;  
par/and do  
  await A;  
  y = y + 1;  
with  
  await A;  
  y = y * 2;  
end
```



always concurrent

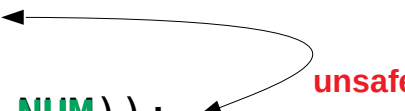
unsafe

- Static checks
 - Level 0: both are refused
 - Level 1: unsafe is refused
 - Level 2: both are accepted
- Possible because of uniqueness of inputs
- Do not affect the semantics

4. Safe Integration with C

```
native do
  #define NUM 10
  void f (void) { <...> }
  void g (int v) { <...> }
  int id (int v) { <...> }
end

par/and do
  _f();
with
  _g(_id(_NUM));
end
```



```
native @const _NUM;
native @pure _id();
native @safe _f() with _g();
```

- Trackable identifiers ``_`` (*C hat*)
- Assumes all identifiers are conflicting
- Annotations to eliminate conflicts

4. Safe Integration with C

- Abortion of trails dealing with resources is unsafe
- Finalization mechanism
 - Pointer assignment must be finalized
 - External resource: pointer from C to Céu (*memory leak*)
 - Local resource: pointer from Céu to C (*dangling pointer*)

```
par/or do
  var _FILE* f;
  f = _fopen(...);
  fwrite(..., f);
  line 2 : requires 'finalize'
  _fwrite(..., f);
  _fclose(f);
with
  <...>
end
```

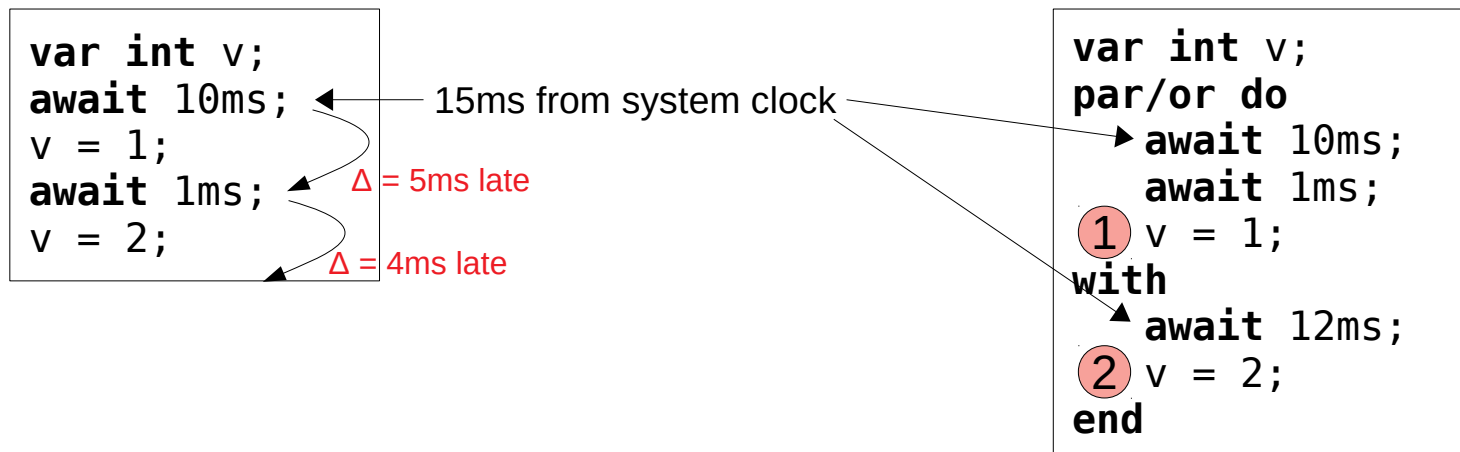
aborts here

```
par/or do
  var _FILE* f;
  finalize
    f = _fopen(...);
  with
    _fclose(f);
  end
  _fwrite(..., f);
  await A;
  _fwrite(..., f);
with
  <...>
end
```

```
par/or do
  var _buffer_t msg;
  <...> // prepare msg
  finalize
    _send_request(&msg);
  with
    _send_cancel(&msg);
  end
  await SEND_ACK;
with
  <...>
end
```

5. First-Class Timers

- Timers, watchdogs, sampling all very common
 - Dedicated syntax
 - Delta compensation (system vs program mismatch)



Applications / Other Work

- ~10-year effort (first commit in 2011)
- [games] *Structured Synchronous Reactive Programming for Game Development Case Study: On Rewriting Pingus from C++ to Céu*, SBGames, 2018
 - **10k/40k reactive code rewritten**
- [embed] *Transparent Standby for Low-Power, Resource-Constrained Embedded Systems: A Programming Language-Based Approach*, LCTES, 2018
 - **interrupt-service routines, automatic standby**
- [media] *Céu-Media: Local Inter-Media Synchronization Using Céu*, WebMedia, 2016
 - **multimedia applications (videos, slideshows)**
- [wsns] *Terra: Flexibility and Safety in Wireless Sensor Networks*, TOSN, 2015
 - **remote reprogramming**



Céu Peculiarities

1. External events

- time is a queue of unique external events

2. Internal events

- intra reactions, stack based

3. Concurrency: internal determinism + static checks

- simple, concurrent assignments/system calls

4. Safe integration with C

- finalization for local/external resources

5. First-class timers

- dedicated syntax, automatic synchronization

An Overview of Céu

A synchronous language inspired by Esterel

Francisco Sant'Anna

francisco@ime.uerj.br



1. External Events

