Céu: A low-level imperative reactive language

Objective

- Support both *Esterel* and *FRP* functionality
  - both models are *synchronous* and *reactive*

- Esterel (imperative/control)

```
do
  every STEP do
    emit JUMP
  end
watching LAP
```

- FRP (declarative/data)

```
x = integral v dt
v = integral a dt
a = <ui_control>
```

Overview

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

Examples

```
( ~Key ~> Print )*
```

```
(~Step; ~>Jump)* || ~Lap
```

Execution Model

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

1) Program starts in one trail from the 1st expression.
2) Active trails execute without interruption. (*Reaction Chain*)
3) Check termination.
4) Await next event and repeat Step 2.

Example

```
(~Tick => v)* || (~End ; v)
```

```
Timeline:
2 ~> Tick
3 ~> Tick
  ~> End
```

Temporal Analysis

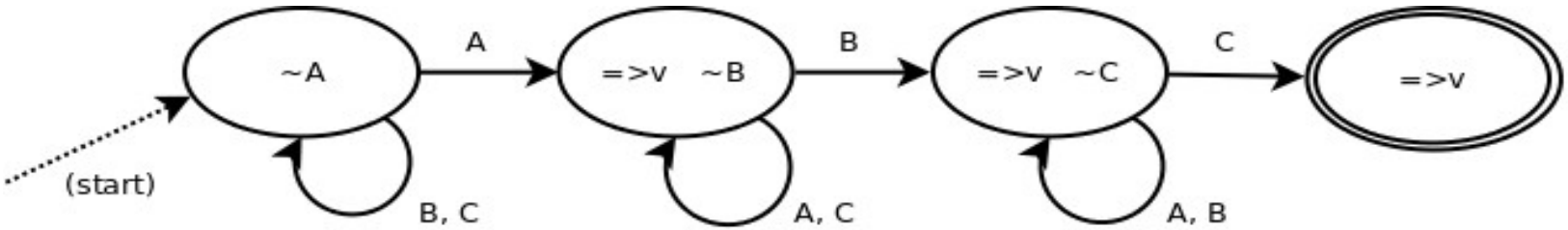
- Bounded execution
  - *Loops:* `~A ; (1)*`
  - *Operators:* 1 -> op
- Determinism
  - `1=>v || 2=>v`
  - `1 || 2`
  - `(1^ && 2^)*`

Céu -> DFA

- DFA
  - State: what to execute and what to await
  - Transition: events
- Detects:
  - concurrent access to variables or events
  - concurrent *split/or* termination
  - concurrent loop escape
  - unreachable expressions
  - whether a program can terminate or not

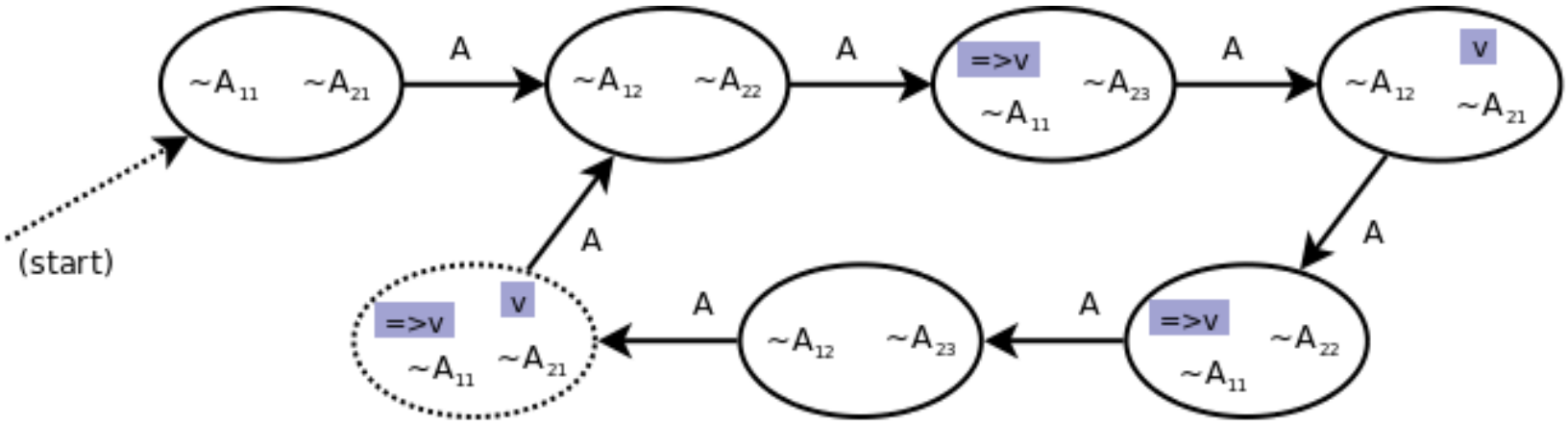
DFA example 1

~A=>v ; ~B=>v ; ~C=>v



DFA example 2

```
(~A ; ~A ; 1=>v)* && (~A ; ~A ; ~A ; v)*
```



- *Basic* Céu:
  - similar to Esterel (equivalent?)
  - better support for variables (still deterministic)
- *Extended* Céu:
  - internal events -> FRP
  - ”physical” time
  - asynchronous blocks -> unbounded & simulation

Internal events

- Communication mechanism among trails

```
(~a ~> Print)* || (~Key ~> a)*
```

- internal events <=> variables : *reactive variables*
- Stack based execution policy:

```
(   0=>v ; ~Start ;
    1~>a ; v->inc=>v    -- expr.1
&&
  ~a~>b ; v->inc=>v     -- expr.2
&&
  ~b~>c ; v->inc=>v ) -- expr.3
```

FRP

- Behaviors: `(pos2 = pos1+20)`

```
( (~pos1,20)->add ~> pos2 )*
```

- Integral: `(pos = S(v,dt))`

```
((pos, (v,~DT)->mul)->add ~> pos)*
```

- Cyclic dependency:

```
(((~fahr,32)->sub, 5)->mul, 9)->div ~> celc)*
||
(((~celc,32)->add, 9)->mul, 5)->div ~> fahr)*
```

Physical Time

- Time from real world, in hours, milliseconds, etc.
- Most used input event (sampling, watchdogs, animations)
- *Timers* in conventional languages:
  - cannot ensure zero delay
  - *residual delta time* (dt)
- Time is a physical quantity:
  - comparable, addable

Physical Time in Céu

- Syntax similar to events `~1h30m10s500ms`
- *dt* is accessible: `~1s500ms => v`
- *dt* is taken into account: `~50ms ; ~50ms ; …`
- Time is comparable:
  - `(~50ms ; … ; ~49ms) || ~100ms`
  - `(~50ms ; … ; ~50ms ; v) || (~100ms ; 1=>v)`
- Avoids ”Collision Tunneling”:
  - `((pos, (100,~DT)->mul)->add ~> pos)*` (wrong!)
  - `(~10ms ; pos->inc~>pos)*` (right!)

Asynchronous Blocks

- Bounded execution -> no unbounded loops
- @{...} can have unbounded loops, but:
  - assigned variables are local
  - split blocks disallowed
  - awaiting disallowed
  - executes while no input events
- Equivalent to unique I/O events:
  - `~>Async_XXX_ini ; ~Async_XXX_end`
- Subjected to *split/or* termination:
  - `~timeout || @{...}`

Simulation in Céu

- *Asyncs* are allowed to trigger
  - external *input* events
  - passage of time

```
  (~10ms ; pos->inc~>pos)*
||
  @{ ~>1s35ms ; (pos,103)->eq->assert) }
```

Conclusion

- Céu ~ Esterel + FRP
  - kernel as complex as Esterel
  - FRP: state changes + internal events
- Internal events
  - handles cyclic dependencies
- Physical Time
  - convenient syntax, *dt* awareness, avoids tunneling
- Asynchronous blocks
  - unbounded execution + simulation
- Temporal Analysis
  - determinism / physical time

Next Steps

- Formal Semantics
  - compare to Esterel and FRP
- Proofs
  - temporal analysis
- Expressiveness
  - type system
  - dynamic code
- Implementation
  - parser
  - parallelism

The *basic* Céu

```
seq:        e1 ; e2 ; ...
and:        e1 && e2 && ...
or:         e1 || e2 || ...

cond:       (e1 ? e2 : e3)
loop:       (e)*
break:      e^

load:       ID
await:      ~ID
assign:     e => ID
trigger:    e ~> ID
op:         e -> ID

exp:        <all above> | CONST
```