SSlang: A Sparse Synchronous Language for Hard Real-Time Tasks

Stephen A. Edwards et al.

IFIP WG2.8 Functional Programming
May 2022, Cornell Tech, Roosevelt Island, NYC
\[ \lambda f . (\lambda x . f (x x)) \lambda x . (f (x x)) \]
gcd a b =
  if a == b
    a
  else
    if a < b
      gcd a (b - a)
    else
      gcd (a - b) b

// Inferred types
// Indentation for grouping
// Everything is an expression
// User-defined binary infix operators
// Juxtaposition for function calls
gcd a b c =   // gcd : &Int → &Int → &Int → ()
  while deref a != deref b    // While loops
    if deref a < deref b      // OCaml-like references
      b ← deref b − deref a   // Assignment to references
    else
      a ← deref a − deref b
  c ← deref a                // Sequencing
add2 a = a ← deref a + 2
mult4 a = a ← deref a * 4

main =
    let a = new 1 // Allocate and name a new variable
    par add2 a // Parallel function calls
    mult4 a // execute in prescribed order
    // a is (1 + 2) * 4 = 12 here
    a ← 1
    par mult4 a
    add2 a
    // a is (1 * 4) + 2 = 6 here
blink led =
loop  // Infinite loop
    after ms 50, led ← 1 // Schedule future variable update
    wait led  // Block on variable update
    after ms 50, led ← 0
    wait led
blink led =
while 1

  after ms 50, led ← 1
  wait led
  after ms 50, led ← 0
  wait led
blink led =
while 1
  fib 19
  after ms 50, led ← 1
  wait led
  after ms 50, led ← 0
  wait led
blink led =

while 1
    fib 23
    after ms 50, led ← 1
    wait led
    after ms 50, led ← 0
    wait led
blink led = loop
after ms 50, led <- 1
wait led
after ms 50, led <- 0
wait led
blink led = loop after ms 50, led <- 1
wait led after ms 50, led <- 0
wait led
blink led =

    loop
        after ms 50, led <- 1
        wait led
    after ms 50, led <- 0
    wait led
blink led =

loop

    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led

led 0
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
blink led =
    loop
        after ms 50, led <- 1
        wait led
        after ms 50, led <- 0
        wait led

led 0
blink led =
  loop
  after ms 50, led <- 1
  wait led
  after ms 50, led <- 0
  wait led

led 0
blink led =

    loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led

led ← 1

0ms  50ms  100ms  150ms

led 0
blink led =
  loop
  after ms 50, led <- 1
  wait led
  after ms 50, led <- 0
  wait led
blink led =

loop

after ms 50, led <- 1
wait led
after ms 50, led <- 0
wait led
blink led =

loop

  after ms 50, led <- 1

wait led

after ms 50, led <- 0

wait led

---

led <- 1

0ms

50ms

100ms

150ms
blink led =

loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led

led ← 1  

0ms    50ms    100ms    150ms
blink led =

```plaintext
loop
  after ms 50, led <- 1
  wait led
  after ms 50, led <- 0
  wait led
```

0ms       led ← 1       50ms     100ms     150ms
blink led =
  loop
    after ms 50, led ← 1
    wait led
    after ms 50, led ← 0
    wait led
blink led =

loop
  after ms 50, led <- 1
  wait led
  after ms 50, led <- 0
  wait led
```plaintext
blink led =

loop

  after ms 50, led <- 1
  wait led

  after ms 50, led <- 0
  wait led
```
blink led =

loop

  after ms 50, led <- 1

  wait led

  after ms 50, led <- 0

  wait led

0ms  50ms 100ms 150ms

led ← 0

led
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
blink led =

loop
after ms 50, led <- 1
wait led
after ms 50, led <- 0
wait led
blink led =
    loop
        after ms 50, led ← 1
        wait led
        after ms 50, led ← 0
        wait led
blink led =

```
loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```
blink led =

  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
blink led =

```
loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
```
blink led =

loop

after ms 50, led <- 1
wait led

after ms 50, led <- 0
wait led
blink led =

loop
  after ms 50, led <- 1
  wait led
  after ms 50, led <- 0
  wait led

led
blink led =
  loop
    after ms 50, led <- 1
    wait led
    after ms 50, led <- 0
    wait led
blink led period =
let event = new ()  // Unit-valued variables pure events
loop
  led ← 1 – deref led
  after period, event ← ()  // Schedule pure event
  wait event  // Wait on write, not change

main led =
  par blink led (ms 50)
  blink led (ms 30)
  blink led (ms 20)  // LED may toggle three times
Deterministic concurrency

Immutable and mutable values

Algebraic data types, pattern matching

Compiles to C for portability across microcontrollers

Heap-resident function activation records

Reference-counted heap, inspired by Perceus [PLDI 2021]

No true parallelism (for now)

No gradual typing (sorry)
Priority queue of events (time, variable, value), ordered by time

Priority queue of threads, ordered by priority

tick()
While there are queued events now,
   Dequeue event $e = (\text{now}, \nu, n)$
   Update variable $\nu$ with new value $n$
   Schedule each thread blocked on variable $\nu$
While there are ready threads,
   Dequeue the lowest-priority thread $t$
   Run thread $t$ from where it last blocked,
      which may write variables immediately to trigger threads now,
      or may schedule future variable update events

One event per variable: scheduling an update deletes any outstanding

Only “later”-priority threads are scheduled when a thread writes to a variable.
### SSlang vs. Esterel

[Berry and Gonthier, SCP 1992]

<table>
<thead>
<tr>
<th>Feature</th>
<th>SSlang</th>
<th>Esterel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deterministic</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Time</td>
<td>Sparse</td>
<td>Dense</td>
</tr>
<tr>
<td>Within instants</td>
<td>Totally-ordered</td>
<td>Constructive</td>
</tr>
<tr>
<td>Runtime</td>
<td>Dynamic Event Queues</td>
<td>Statically Scheduled</td>
</tr>
<tr>
<td>Topology</td>
<td>Dynamic, recursive</td>
<td>Static</td>
</tr>
</tbody>
</table>
## SSlang vs. Ptides

[Zhao, Liu, and Lee, RTAS 2007] [Zou Ph.D 2011]

<table>
<thead>
<tr>
<th></th>
<th>SSlang</th>
<th>Ptides</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between instants</td>
<td>Discrete-event</td>
<td>Discrete-Event</td>
</tr>
<tr>
<td>Within instants</td>
<td>Totally-ordered</td>
<td>Discrete-Event</td>
</tr>
<tr>
<td>Topology</td>
<td>Dynamic, recursive</td>
<td>Static</td>
</tr>
<tr>
<td>Implementation</td>
<td>Single-threaded</td>
<td>Distributed</td>
</tr>
</tbody>
</table>
https://github.com/ssm-lang/sslang