PSM:
Policy-Synchronised Shared Memory in Haskell

M. Mendler & M. Pouzet

Synchron 2025

https://www.arxiv.org/abs/2506.15424 (June 2025)

Introduction

Objective

The objective of the work is to implement ...

- ... Esterel-style synchronous model of computation for main-stream programmers (sequentially constructive + reaction to absence)
- ... based on the work

  J. Aguado, M. Mendler, M. Pouzet, P. Roop, R. v. Hanxleden: Deterministic Concurrency: A Clock-Synchronised Shared Memory Approach, ESOP 2018.

  which abstracts Esterel's signals to general abstract data structures (ADS)
- ... shallowly, as a domain-specific library (DSL) in Haskell.

Synchronous MoC in Mainstream Languages

- F. Boussinot: *SugarCubes Implementation of Causality*, 1998 [Java]
- R. Budde, A. Poigné, K.-H. Sylla: *SynERJY – An Object-oriented Synchronous Language*, SLAP 2004. [Java]
- F. Boussinot. FairThreads: *Mixing Cooperative and Preemptive Threads in C*. In Concurrency and Computation: Practice and Experience, 18:445–469, 2005. C
- L. Mandel, M. Pouzet: ReactiveML: a reactive extension to ML. PPDP'05. [ML]
- F. Boussinot, F. Dabrowski: *Safe reactive programming: The FunLoft proposal*, 2007. [Java]
- I. Perez, M. Bärenz, and H. Nilsson: *Functional Reactive Programming, Refactored*. SIGPLAN Notices, 2016. [Haskell]
- M. Serrano, R. Findler: *The Functional, the Imperative, and the Sudoku: Getting Good, Bad, and Ugly to Get Along*. ICFP 2024. [Javascript]
- ...

Haskell Memory Abstractions

IO Monad (IORef) – Input/Output

```
newIORef :: a → IO (IORef a)
readIORef :: IORef a → IO a
writeIORef :: IORef a → a → IO ()
```

concurrent & multi-threaded
non-deterministic
simple mutable references

STM Monad (TVar, MVar, ...) – Software Transactional Memory

```
newTVar :: a → STM (TVar a)
writeTVar :: TVar a → a → STM ()
readTVar :: TVar a → STM a
atomically :: STM a → IO a
```

sequential & single-threaded
deterministic
fixed selection of memory objects

But what about multi-threaded, deterministic, general ADS?

PSM – Policy Synchronised Memory

PSM is a type constructor (generalised monad) wraps a safe evaluation context for concurrent functional transactions (Church-Rosser):

- provides user-defined, race-free, shared data structures and synchronised IO, as instances of class PSMStruct.
- data structures instance PSMStruct p are accessed through policy synchronised references ref :: PSMVar p.

Expressions e::PSM a are multi-threaded IO-actions:

- compute a determinate value of type a
- access shared PSMVars
- synchronise on a (single) clock à la SCEsterel
-

Hidden under the Carpet: PSM a $\cong$ (Prediction, IO a) publish a prediction and update their potential as part of the IO action.

The Class of PSM Structures

The PSMStruct Class

class PSMStruct p where

```
type PSMStatic p
data PSMState p

init   :: PSMStatic p → IO (PSMState p)
tick   :: PSMState p → IO ()
term   :: PSMState p → IO ()

data family Method p b c
method   :: Method p b c → PSMState p → b → STM c
methodId :: Method p b c → MethodId
prec     :: PSMState p → MethodId → STM (Maybe [MethodId])
```

... subject to Coherence Conditions (omitted here, see ESOP 2018).

PSM Actions

The PSMVar Wrapper

Assuming p is an instance of the class PSMStruct:

data PSMVar p

- Policy-synchronised memory references
- Every ref :: PSMVar p is a sharable data structure of type p

Hidden under the Carpet: PSMVar p $\cong$ (Id, PSMState p, Potential)

Instantiating a PSMVar

```
newPSMVar :: PSMStruct p ⇒
                PSMStatic p → (PSMVar p → PSM b) → PSM b
```

```
myAct :: PSM b = newPSMVar init $ λ ref →
  <use 'ref::PSMVar p' to access the data structure of type 'p'>
```

The PSMVal Monad

data PSMVal a

- monad of thread-local pure values of type a
- Can only be used inside PSM actions but cannot influence prediction

```
return  :: a → PSMVal a
(>>=)   :: PSMVal a → (a → PSMVal b) → PSMVal b
```

PSMVal and PSMVar lift PSM Methods for Application

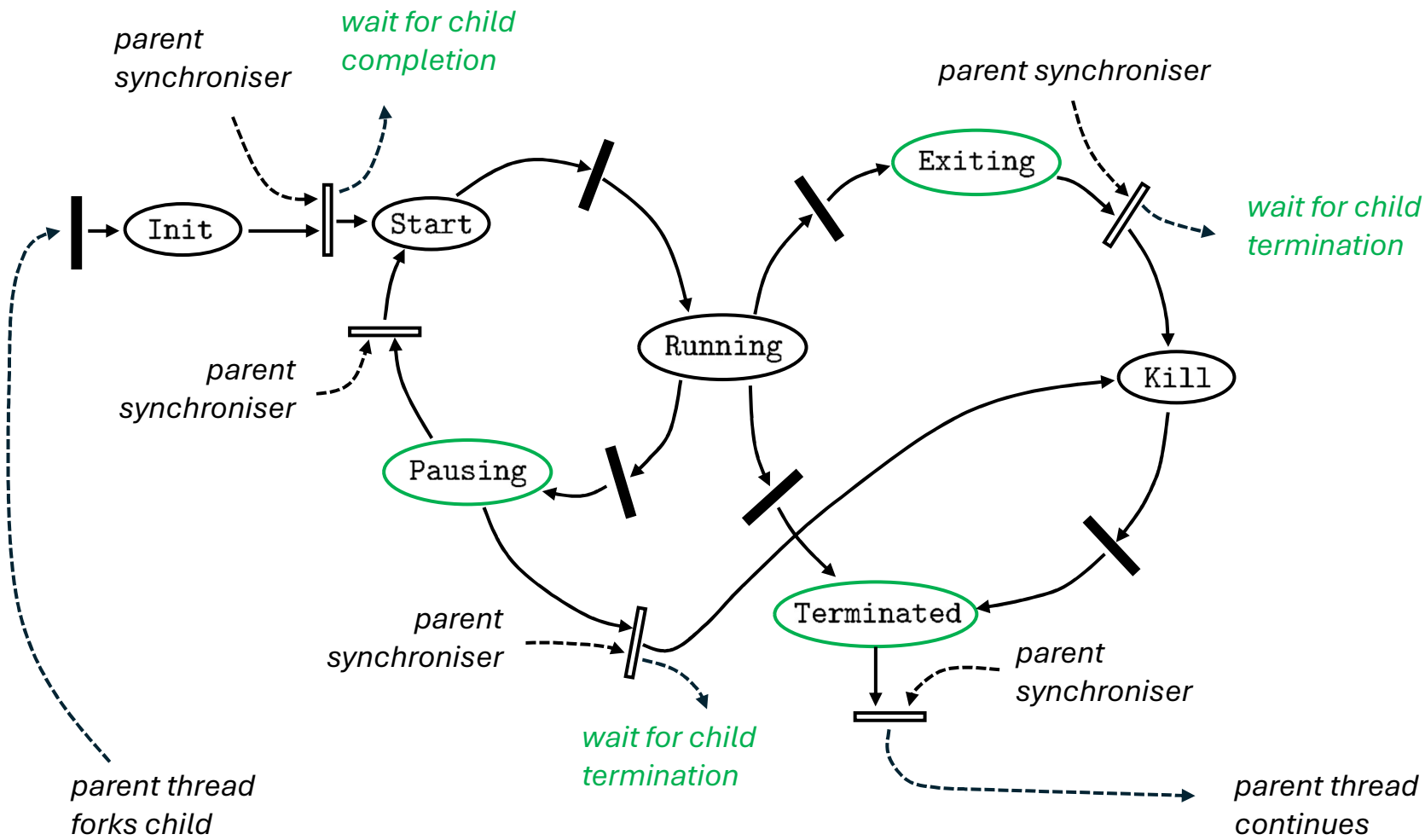
```
runMtd :: PSMStruct p ⇒
          Method p a b → PSMVar p → PSMVal a → PSM b
```

PSM Actions

Core PSM Action Combinators

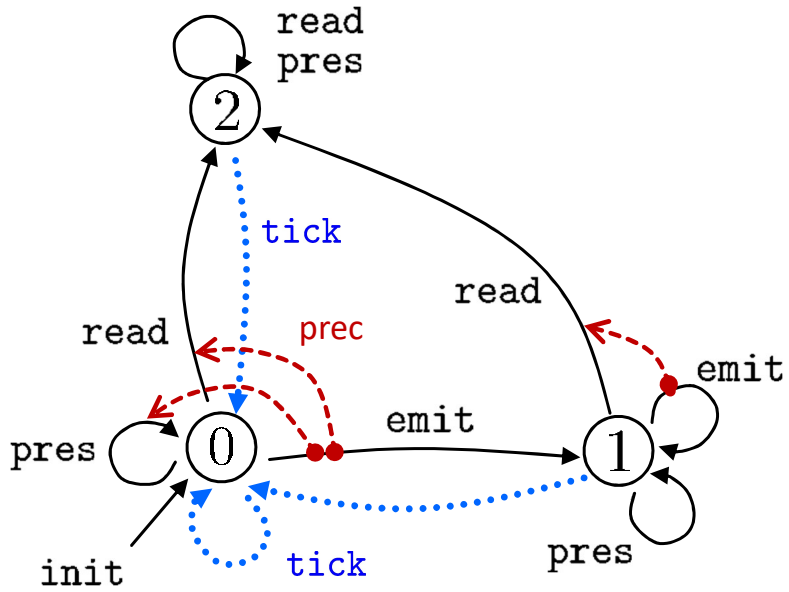
```
delay :: Int → PSM () -- terminating with delay
pause :: PSM ()  -- pausing
kill  :: PSM ()  -- exiting all

val      :: PSMVal a → PSM a
(>>=)    :: PSM a → (PSMVal a → PSM b) → PSM b

(|||)    :: PSM a → PSM b → PSM (a,b)
ifte     :: PSMVal Bool → PSM a → PSM a → PSM a
repUntil :: PSM Bool → PSM ()

runPSM   :: PSM a → IO ThreadId
```

Thread Scheduling



PSMStruct for Esterel Signals

TSig: Temporary Valued Esterel Signal (V7)



Method Interface

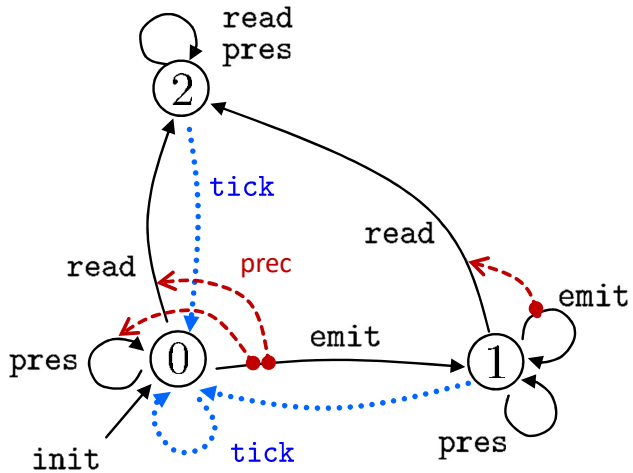
```
data TSig a = TSig a

emitPTSig :: PSMVar(TSig a) → PSMVal a → PSM ()   -- emit
presPTSig :: PSMVar(TSig a) → PSM Bool            -- pres
readPTSig :: PSMVar(TSig a) → PSM a               -- read
```

TSig: State

```
data StatSig a = StatSig {default :: a, cmb :: a → a → a}

instance PSMStruct (TSig a) where

  type PSMStatic (TSig a) = StatSig a
  data PSMState (TSig a)  = TSigState a (a → a → a) (TVar (a, Int))


  init (StatSig v0 cmb) = do
    sRef ← atomically $ newTVar (v0, 0)
    return $ TSigState v0 cmb sRef

  tick (TSigState def _ sRef) =
    atomically $ writeTVar sRef (def, 0)
```

TSig: Methods & Policy

```
data instance Method (TSig a) b c where
  DEmit :: Method (TSig a) a ()
  DPres :: Method (TSig a) () Bool
  DRead :: Method (TSig a) () a

methodId DEmit = MethodId 2 "emit"
methodId DPres = MethodId 3 "pres"
methodId DRead = MethodId 4 "read"
```



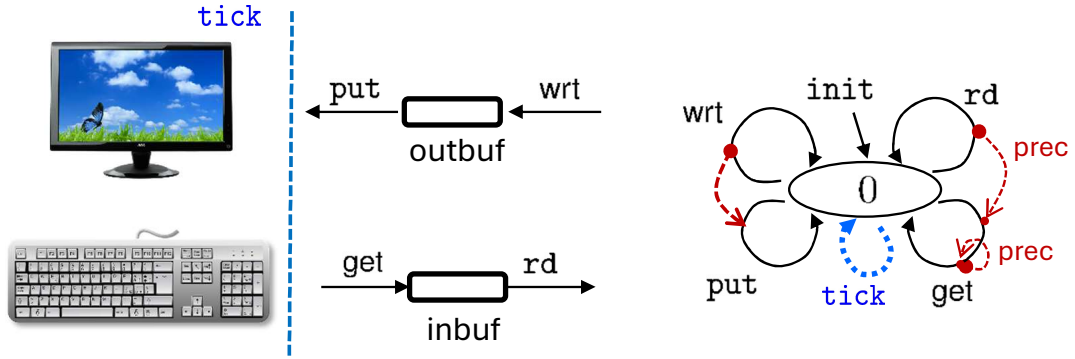

```
prec (TSigState _ _ sRef) (MethodId 2 "emit") = do
  (_, s) ← readTVar sRef
  return $ if s == 2 then Nothing else Just []
prec (TSigState _ _ sRef) (MethodId 3 "pres") = do
  (_, s) ← readTVar sRef
  return $ if s == 0 then Just [MethodId 2 "emit"] else Just []
...
```

TSig: Method Implementation

```
method DEmit (TSigState _ cmb sRef) v = do
  (val, _) ← readTVar sRef; writeTVar sRef (cmb val v, 1)

method DPres (TSigState _ cmb sRef) v = do
  (_, s) ← readTVar sRef; return $ s != 0

method DRead (TSigState _ _ sRef) _ = do
  (val, _) ← readTVar sRef; return val

emitPTSig :: PSMVar(TSig a) → PSMVal a → PSM ()
emitPTSig = runMtd DEmit

presPTSig :: PSMVar(TSig a) → PSM Bool
presPTSig sig = runMtd DPres sig (PSMVal())

readPTSig :: PSMVar(TSig a) → PSM a
readPTSig sig = runMtd DRead sig (PSMVal ())
```

PSMStruct for String IO

StrIO: String IO Object



Method Interface

```
data StrIO = StrIO

wrStrIO  :: PSMVar StrIO → PSMVal String → PSM ()
  -- writing to the out-buffer (wrt)
rdStrIO  :: PSMVar StrIO → PSM String
  -- reading the in-buffer (rd)
```

Example

Examples

Termination Controller

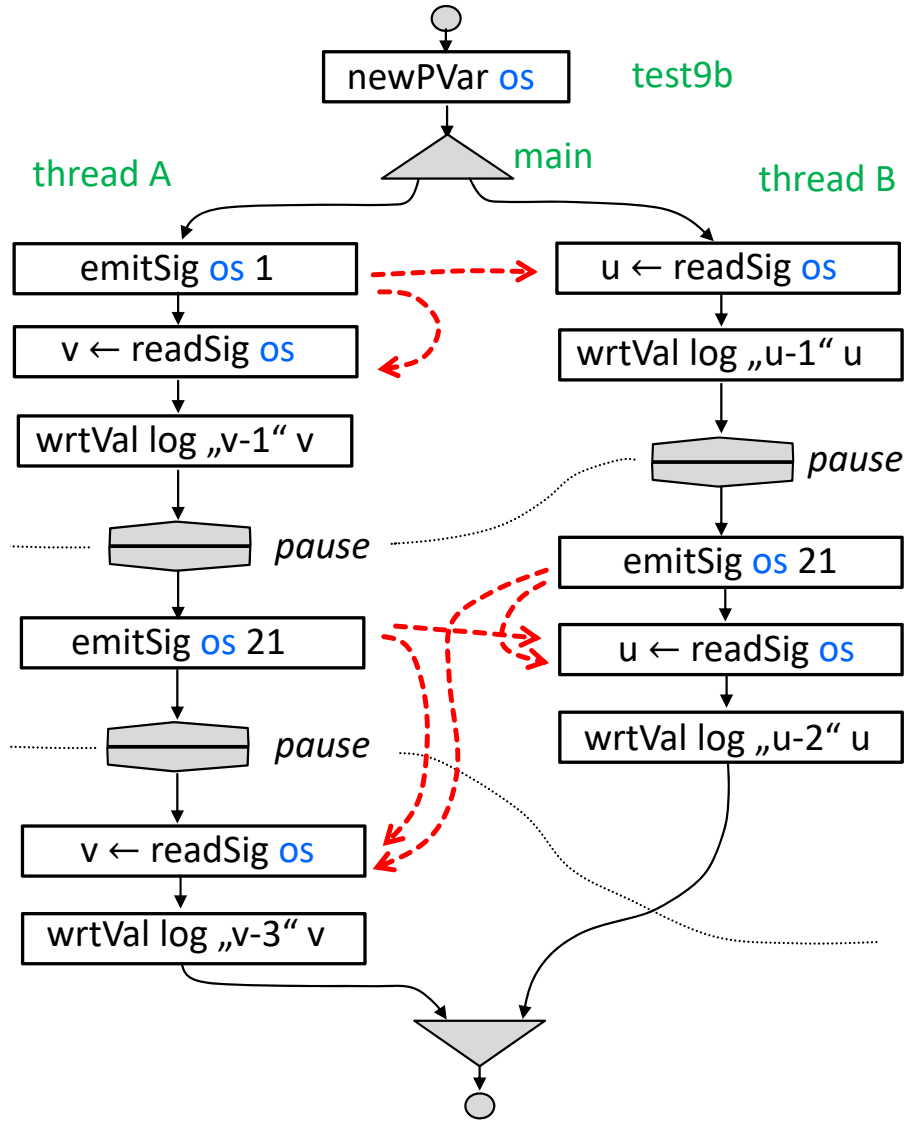
```
wait4Quit :: PSMVar StrIO → PSM ()
wait4Quit io = repUntil cycle where
  cycle =
    rdStrIO io >>>= λl →
    ifte (do s ← l; return (s == ":q"))
      (kill >>> val (pure False))
      (pause >>> val (pure False))
```

Logging to StdOut

```
wrtVal :: Show a ⇒ PSMVar StrIO → String → a → PSM ()
wrtVal log s n =
  let line = s ++ " = " ++ show n ++ " " in
    wrStrIO log (pure line)
```

Example

```
testPTSig = runPSM $
 newPSMVar zpSigInit $ λ os →
 newPSMVar "io" $ λ io →
        wait4Quit io
   ||| ( emitPTSig os (pure 1) >>>
         readPTSig os >>= λ v →
         wrtVal io "v-1" v >>>
         pause >>>
         emitPTSig os (pure 21) >>>
         pause >>>
         readPTSig os >>= λ v →
         wrtVal io "v-3" v)
   ||| ( readPTSig os >>= λ u →
         wrtVal io "u-1" u >>>
         pause >>>
         emitPTSig os (pure 21) >>>
         readPTSig os >>= λ u →
         wrtVal io "u-2" u >>>
         pause)
```



Conclusion

Conclusion

PSM provides ...

- ... deterministic shared abstract data types in Haskell
- ... shallow encoding of policy-based scheduling of threads, generalising Esterel's constructive semantics
- ... language-independent workbench for synchronous programming with threads and destructive memory, exploiting the power of full Haskell.

ToDos:

- Optimise Scheduling (e.g., object-local prediction management, e.g., using ideas from FunLoft [F. Boussinot, F. Dabrowski])
- Extend to multiple clocks (e.g., as in Synpatick, with C. Stolze & L. Liquori) and general trap mechanism
- Application case study (e.g., GALS for Railway Signaling in context of DFG-funded Transfer Project PRETSY, with R. von Hanxleden).