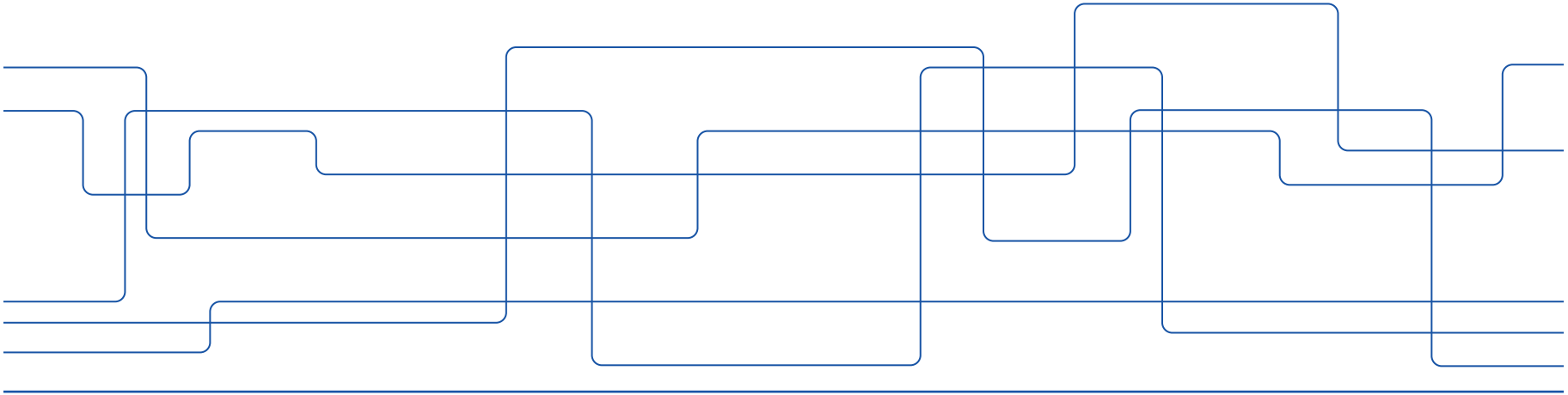
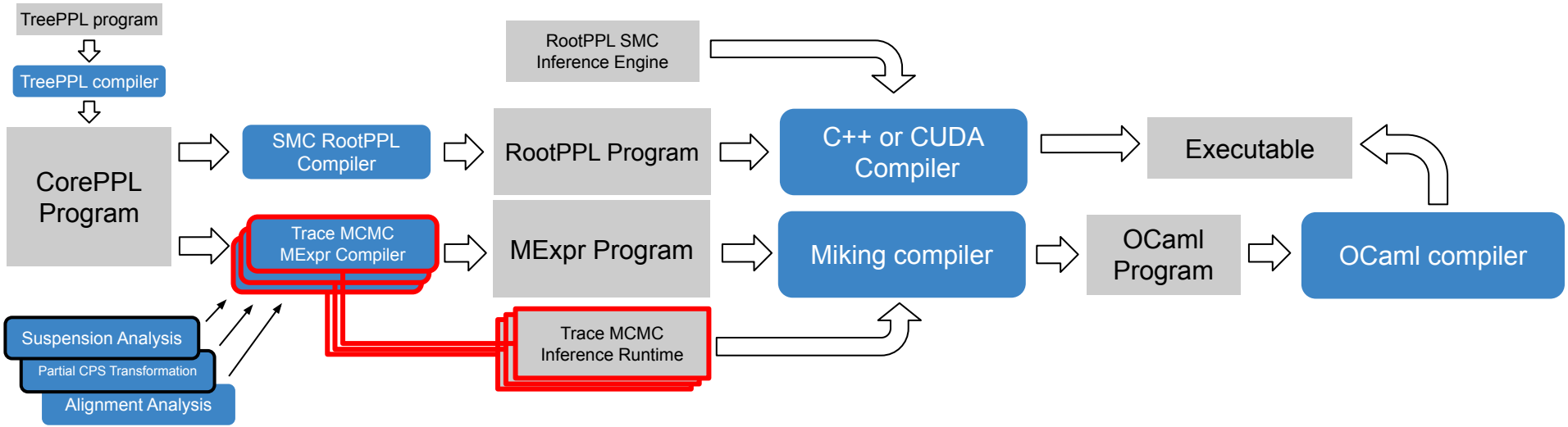
KTH ROYAL INSTITUTE
OF TECHNOLOGY

Implementing Inference Algorithms in Miking DPPL

Daniel Lundén



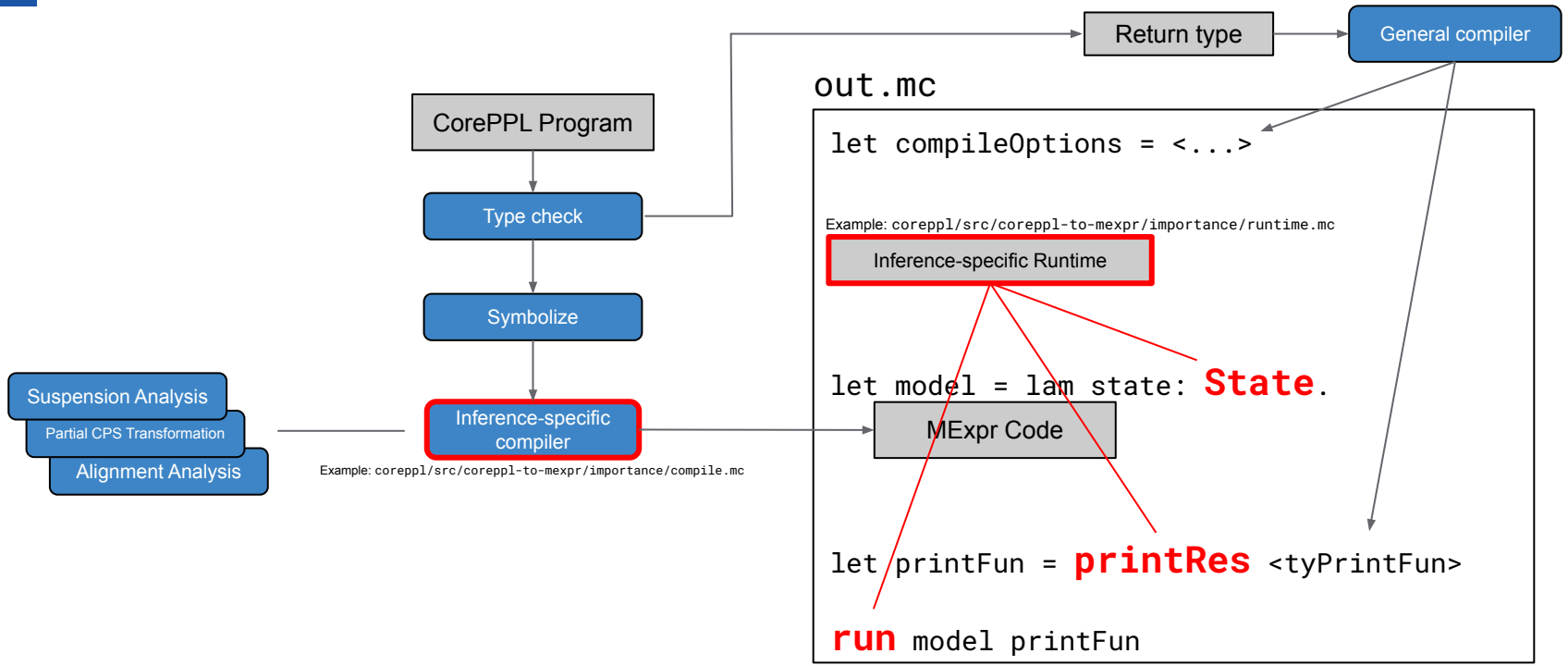
Overall Toolchain



Directory Structure

File(s)	Function
`coreppl/src/*`	Files directly related to CorePPL (and no specific compile target).
`coreppl/src/cppl.mc`	The entrypoint of the `cppl` command.
`coreppl/src/coreppl-to-mexpr/*`	Files related to compiling CorePPL to MExpr.
`coreppl/src/coreppl-to-mexpr/compile.mc`	The entrypoint of the CorePPL to MExpr compiler.
`coreppl/src/coreppl-to-mexpr/*/*`	Files implementing a specific inference algorithm.

coreppl/src/coreppl-to-mexpr/compile.mc



coreppl/src/coreppl-to-mexpr/importance/runtime.mc (Simplified)

```
-- In importance sampling, the state is simply the accumulated weight.
type State = Ref Float

let updateWeight = lam v. lam state. modref state (addf (deref state) v)

-- General inference algorithm for importance sampling
let run : all a. (State -> a) -> (Res a -> ()) -> () = lam model. lam printResFun.

  -- Read number of runs and sweeps
  match monteCarloArgs () with (particles, sweeps) in

  -- Repeat once for each sweep
  repeat (lam.
      let weightInit: Float = 0. in
      let states = createList particles (lam. ref weightInit) in
      let res = mapReverse model states in
      let res = (mapReverse deref states, res) in
      printResFun res
    ) sweeps

let printRes : all a. (a -> String) -> Res a -> () = lam printFun. lam res.
  printLn (float2string (normConstant res.0));
  printSamples printFun res.0 res.1
```

coreppl/src/coreppl-to-mexpr/importance/compile.mc (Simplified)

```
lang MExprPPLImportance =

  sem compile : Options -> Expr -> Expr
  sem compile options =
  | t ->
    -- Transform distributions to MExpr distributions
    let t = mapPre_Expr_Expr transformTmDist t in
    -- Transform samples, observes, and weights to MExpr
    let t = mapPre_Expr_Expr transformProb t in
    t

  sem transformProb =
  | TmAssume t -> app_ (recordproj_ "sample" t.dist) unit_
  | TmObserve t ->
    let weight = app_ (recordproj_ "logObserve" t.dist) t.value in
    appf2_ (var_ "updateWeight") weight (var_ "state")
  | TmWeight t -> appf2_ (var_ "updateWeight") t.weight (var_ "state")
  | TmResample t -> unit_
  | t -> t

end

let compilerImportance = lam options. use MExprPPLImportance in
  ("importance/runtime.mc", compile options)
```