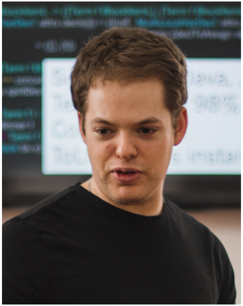
Abstract

In this talk I will show an implementation of angelic non-determinism (as with the ‘amb‘ operator) that uses only mutable state and exceptions – we have pure OCaml and SML implementations, without changingthe compiler or the runtime system.
This approach, which relies on a neat trick found by James Koppel, can be extended to an implementation of full delimited continuations!

Keep (re)playing until your get all the successes

James Koppel, **Gabriel Scherer**, Armando Solar-Lezama

June 22, 2018



Direct-style angelic non-determinism

val choose : 'a list → 'a
val with_choice : (unit → 'a) → 'a list

Direct-style angelic non-determinism

```
val choose : 'a list → 'a
val with_choice : (unit → 'a) → 'a list

let queens n =
  let range = List.init n (fun i → i) in (* [0; ...; n−1] *)
  let rec enum_nqueens i qs =
    if i = n then qs else
      let q = choose (List.filter (okay qs) range) in
      enum_nqueens (i+1) (q :: qs)
  in
  with_choice (fun () → enum_queens 0 [])
```

Direct-style angelic non-determinism

```
val choose : 'a list → 'a
val with_choice : (unit → 'a) → 'a list

let queens n =
  let range = List.init n (fun i → i) in (* [0; ...; n−1] *)
  let rec enum_nqueens i qs =
    if i = n then qs else
      let q = choose (List.filter (okay qs) range) in
      enum_nqueens (i+1) (q :: qs)
  in
  with_choice (fun () → enum_queens 0 [])
```

But: except for calls to choose, the argument of with_choice has to be (observably) *pure*.

Direct-style angelic non-determinism

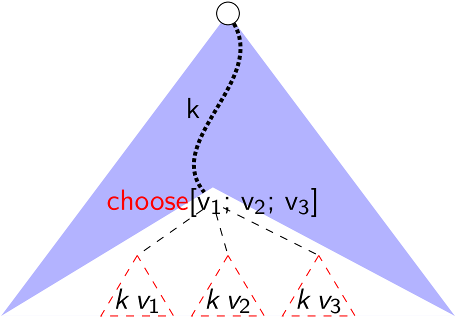
```
val choose : 'a list → 'a
val with_choice : (unit → 'a) → 'a list

let queens n =
  let range = List.init n (fun i → i) in (* [0; ...; n−1] *)
  let rec enum_nqueens i qs =
    if i = n then qs else
      let q = choose (List.filter (okay qs) range) in
      enum_nqueens (i+1) (q :: qs)
  in
  with_choice (fun () → enum_queens 0 [])
```

But: except for calls to choose, the argument of with_choice has to be (observably) *pure*.

But: proof of *feasability*, not a practical implementation.

Angelic non-determinism computation trees

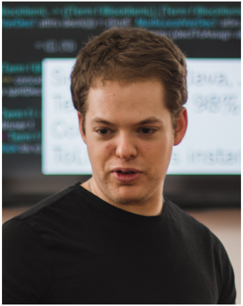
val choose : 'a list → 'a
val with_choice : (unit → 'a) → 'a list



(usually done by capturing and copying continuations)

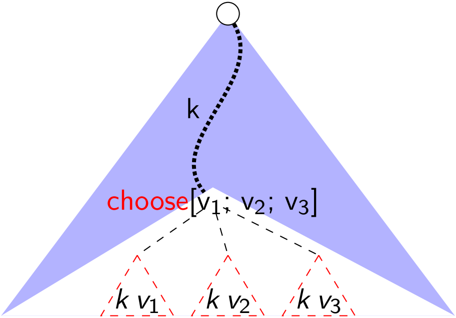
Section 1

Jimmy's neat trick



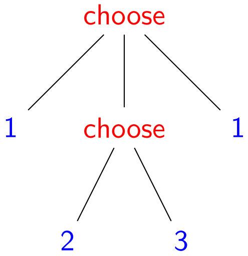
Jimmy's trick

val choose : 'a list → 'a
val with_choice : (unit → 'a) → 'a list



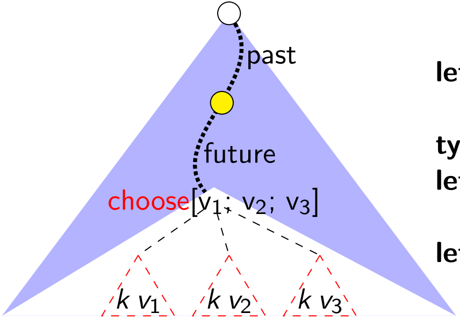
Jimmy's trick: if we can't *capture* k, just *replay* it.

```
with_choice (fun () →
  if choose [true; false; true] then 1
  else
    if choose [true; false] then 2 else 3
)
```



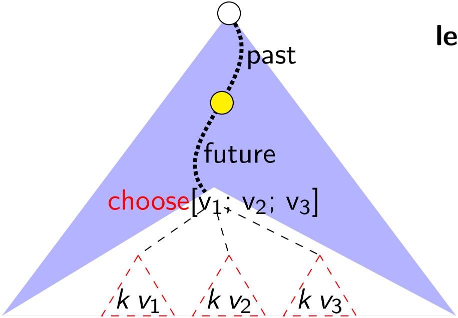
On replay, remember the value

Setup (1/3)



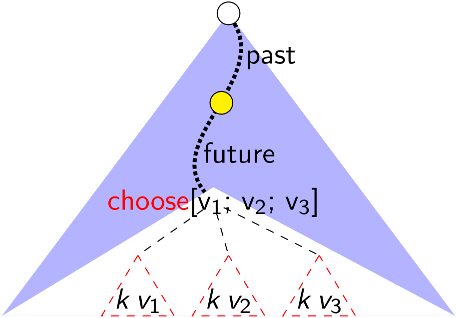
```
type idx = int * int
let start_idx xs = (0, List.length xs)
let next_idx (k, len) =
  if k + 1 = len then None
  else Some (k + 1, len)
let get xs (k, len) = List.nth xs k

type 'a stack = 'a list ref
let push stack x =
  stack := x :: !stack
let pop stack = match !stack with
  | [] → None
  | x::xs → stack := xs; Some x
```

choose (2/3)



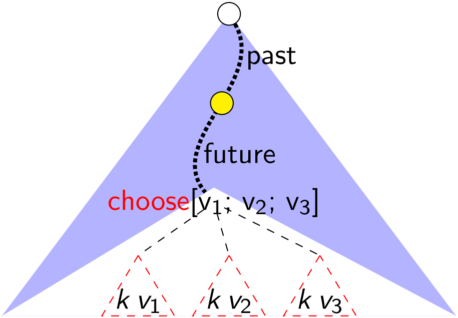
```
let past = ref []
let future = ref []
exception Empty

let choose = function
  | [] → raise Empty
  | xs →
     let i = match pop future with
     | None → start_idx xs
     | Some i → i
     in
     push past i;
     get xs i
```

with_choice (3/3)



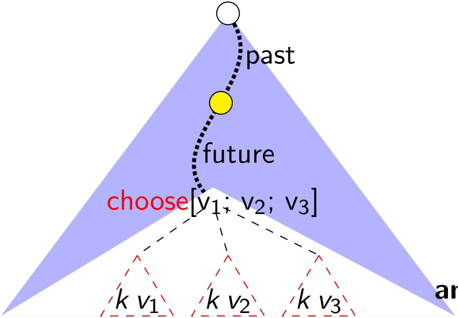
```
let rec with_choice f = loop f []
and loop f acc =
  let r =
    try [f ()] with Empty → [] in
  let acc = r @ acc in
```

with_choice (3/3)



```
let rec with_choice f = loop f []
and loop f acc =
  let r =
    try [f ()] with Empty → [] in
  let acc = r @ acc in
  match next_path !past with
  | None → List.rev acc
  | Some path →
    past := [];
    future := List.rev path;
    loop f acc
```

with_choice (3/3)



```
let rec with_choice f = loop f []
and loop f acc =
  let r =
    try [f ()] with Empty → [] in
  let acc = r @ acc in
  match next_path !past with
  | None → List.rev acc
  | Some path →
    past := [];
    future := List.rev path;
    loop f acc
and next_path = function
  | [] → None
  | i::is →
    match next_idx i with
    | Some i' → Some (i'::is)
    | None → next_path is
```

Delimited continuations

Jimmy extended this idea to implement *delimited continuations*.

Delimited continuations

Jimmy extended this idea to implement *delimited continuations*.

Implementation:

Delimited continuations

Jimmy extended this idea to implement *delimited continuations*.

Implementation:

- surprisingly similar to choose (shift) and with_choice (reset)

Delimited continuations

Jimmy extended this idea to implement *delimited continuations*.

Implementation:

- surprisingly similar to choose (shift) and with_choice (reset)
- ... yet very hard to understand

Delimited continuations

Jimmy extended this idea to implement *delimited continuations*.

Implementation:

- surprisingly similar to choose (shift) and with_choice (reset)
- ... yet very hard to understand

Not in this talk!

Delimited continuations

Jimmy extended this idea to implement *delimited continuations*.

Implementation:

- surprisingly similar to choose (shift) and with_choice (reset)
- ... yet very hard to understand

Not in this talk!

`https://arxiv.org/abs/1710.10385`

Section 2

Non-determinism: correctness proof

Continuation machines

$$\boxed{(t, K, s, R)} \qquad (t, \mathsf{halt}, \emptyset, \emptyset)$$

$$\begin{array}{ll} t, u ::= & \\ & \mid x, y, z \dots \\ & \mid n \in \mathbb{N} \\ & \mid \mathsf{S}\ t \\ & \mid \mathsf{let}\ x\ =\ t\ \mathsf{in}\ t' \\ & \mid \mathsf{choose}\ x\ y \end{array}$$

$$\begin{array}{ll} K ::= & \\ & \mid \mathsf{S}\ K \\ & \mid \mathsf{let}\ x\ =\ \Box\ \mathsf{in}\ (t, K) \\ & \mid \mathsf{halt} \end{array}$$

$$\begin{array}{lcl} s & ::= & \emptyset \mid (t, K).s \\ R & ::= & \emptyset \mid n.R \end{array}$$

Continuation machines

$$\boxed{(t, K, s, R)} \qquad (t, \mathsf{halt}, \emptyset, \emptyset)$$

$$\begin{array}{lcl}
(\mathsf{S}\ t, K, s, R) & \rightarrow & (t, \mathsf{S}\ K, s, R) \\
(n, \mathsf{S}\ K, s, R) & \rightarrow & (n + 1, K, s, R)
\end{array}$$

Continuation machines

$$\boxed{(t, K, s, R)} \qquad (t, \mathsf{halt}, \emptyset, \emptyset)$$

$$\begin{array}{lcl}
(\mathsf{S}\ t, K, s, R) & \rightarrow & (t, \mathsf{S}\ K, s, R) \\
(n, \mathsf{S}\ K, s, R) & \rightarrow & (n+1, K, s, R) \\
(\mathsf{let}\ x\ =\ t\ \mathsf{in}\ t', K, s, R) & \rightarrow & (t, (\mathsf{let}\ x\ =\ \Box\ \mathsf{in}\ (t', K)), s, R)
\end{array}$$

Continuation machines

$$\boxed{(t, K, s, R)} \qquad (t, \mathsf{halt}, \emptyset, \emptyset)$$

$$\begin{array}{lcl}
(\mathsf{S}\ t, K, s, R) & \to & (t, \mathsf{S}\ K, s, R) \\
(n, \mathsf{S}\ K, s, R) & \to & (n+1, K, s, R) \\
(\mathsf{let}\ x\ =\ t\ \mathsf{in}\ t', K, s, R) & \to & (t, (\mathsf{let}\ x\ =\ \square\ \mathsf{in}\ (t', K)), s, R) \\
(n, \mathsf{let}\ x\ =\ \square\ \mathsf{in}\ (t', K), s, R) & \to & (t'[x \leftarrow n], K, s, R)
\end{array}$$

Continuation machines

$$\boxed{(t, K, s, R)} \qquad (t, \mathsf{halt}, \emptyset, \emptyset)$$

$$\begin{array}{lcl}
(\mathsf{S}\ t, K, s, R) & \rightarrow & (t, \mathsf{S}\ K, s, R) \\
(n, \mathsf{S}\ K, s, R) & \rightarrow & (n+1, K, s, R) \\
(\mathsf{let}\ x\ =\ t\ \mathsf{in}\ t', K, s, R) & \rightarrow & (t, (\mathsf{let}\ x\ =\ \square\ \mathsf{in}\ (t', K)), s, R) \\
(n, \mathsf{let}\ x\ =\ \square\ \mathsf{in}\ (t', K), s, R) & \rightarrow & (t'[x \leftarrow n], K, s, R)
\end{array}$$

$$(\mathsf{choose}\ n_1\ n_2, K, s, R) \quad \rightarrow \quad (n_1, K, (n_2, K).s, R)$$

Continuation machines

$$\boxed{(t, K, s, R)} \qquad (t, \mathsf{halt}, \emptyset, \emptyset)$$

$$\begin{array}{lcl}
(\mathsf{S}\ t, K, s, R) & \to & (t, \mathsf{S}\ K, s, R) \\
(n, \mathsf{S}\ K, s, R) & \to & (n+1, K, s, R) \\
(\mathsf{let}\ x\ =\ t\ \mathsf{in}\ t', K, s, R) & \to & (t, (\mathsf{let}\ x\ =\ \Box\ \mathsf{in}\ (t', K)), s, R) \\
(n, \mathsf{let}\ x\ =\ \Box\ \mathsf{in}\ (t', K), s, R) & \to & (t'[x \leftarrow n], K, s, R)
\end{array}$$

$$\begin{array}{lcl}
(\mathsf{choose}\ n_1\ n_2, K, s, R) & \to & (n_1, K, (n_2, K).s, R) \\
(n, \mathsf{halt}, (n', K).s, R) & \to & (n', K, s, n.R)
\end{array}$$

History machines

$$\boxed{(t, K, P, F, R)_u} \qquad (t, \mathsf{halt}, \emptyset, \emptyset, \emptyset)_t$$

$$i \;::=\; 1 \mid 2 \qquad \begin{array}{lcl} P & ::= & \emptyset \mid P.i \\ F & ::= & \emptyset \mid i.F \end{array}$$

History machines

$$\boxed{(t, K, P, F, R)_u} \qquad (t, \mathsf{halt}, \emptyset, \emptyset, \emptyset)_t$$

$$\begin{array}{ll}
(\mathsf{S}\ t, K, P, F, R)_u & \rightarrow (t, \mathsf{S}\ K, P, F, R)_u \\
(n, \mathsf{S}\ K, P, F, R)_u & \rightarrow (n+1, K, P, F, R)_u \\
(\mathsf{let}\ x\ =\ t\ \mathsf{in}\ t', K, P, F, R)_u & \rightarrow (t, \mathsf{let}\ x\ =\ \square\ \mathsf{in}\ (t', K), P, F, R)_u \\
(n, \mathsf{let}\ x\ =\ \square\ \mathsf{in}\ (t', K), P, F, R)_u & \rightarrow (t'[x \leftarrow n], K, P, F, R)_u
\end{array}$$

History machines

$$\boxed{(t, K, P, F, R)_u} \qquad (t, \mathsf{halt}, \emptyset, \emptyset, \emptyset)_t$$

$$\begin{array}{ll}
(\mathsf{S}\ t, K, P, F, R)_u & \to (t, \mathsf{S}\ K, P, F, R)_u \\
(n, \mathsf{S}\ K, P, F, R)_u & \to (n+1, K, P, F, R)_u \\
(\mathsf{let}\ x\ =\ t\ \mathsf{in}\ t', K, P, F, R)_u & \to (t, \mathsf{let}\ x\ =\ \square\ \mathsf{in}\ (t', K), P, F, R)_u \\
(n, \mathsf{let}\ x\ =\ \square\ \mathsf{in}\ (t', K), P, F, R)_u & \to (t'[x \leftarrow n], K, P, F, R)_u
\end{array}$$

$$(\mathsf{choose}\ n_1\ n_2, K, P, \emptyset, R)_u \quad \to \quad (\mathsf{choose}\ n_1\ n_2, K, P, 1.\emptyset, R)_u$$

History machines

$$\boxed{(t, K, P, F, R)_u} \qquad (t, \mathsf{halt}, \emptyset, \emptyset, \emptyset)_t$$

$$\begin{array}{ll}
(\mathsf{S}\ t, K, P, F, R)_u & \to (t, \mathsf{S}\ K, P, F, R)_u \\
(n, \mathsf{S}\ K, P, F, R)_u & \to (n+1, K, P, F, R)_u \\
(\mathsf{let}\ x\ =\ t\ \mathsf{in}\ t', K, P, F, R)_u & \to (t, \mathsf{let}\ x\ =\ \square\ \mathsf{in}\ (t', K), P, F, R)_u \\
(n, \mathsf{let}\ x\ =\ \square\ \mathsf{in}\ (t', K), P, F, R)_u & \to (t'[x \leftarrow n], K, P, F, R)_u
\end{array}$$

$$\begin{array}{lcl}
(\mathsf{choose}\ n_1\ n_2, K, P, \emptyset, R)_u & \to & (\mathsf{choose}\ n_1\ n_2, K, P, 1.\emptyset, R)_u \\
(\mathsf{choose}\ n_1\ n_2, K, P, (i.F), R)_u & \to & (n_i, K, (P.i), F, R)_u
\end{array}$$

History machines

$$\boxed{(t, K, P, F, R)_u} \qquad (t, \mathrm{halt}, \emptyset, \emptyset, \emptyset)_t$$

$$\begin{aligned}
&(\mathrm{S}\ t, K, P, F, R)_u && \rightarrow (t, \mathrm{S}\ K, P, F, R)_u \\
&(n, \mathrm{S}\ K, P, F, R)_u && \rightarrow (n+1, K, P, F, R)_u \\
&(\mathrm{let}\ x\ =\ t\ \mathrm{in}\ t', K, P, F, R)_u && \rightarrow (t, \mathrm{let}\ x\ =\ \square\ \mathrm{in}\ (t', K), P, F, R)_u \\
&(n, \mathrm{let}\ x\ =\ \square\ \mathrm{in}\ (t', K), P, F, R)_u && \rightarrow (t'[x \leftarrow n], K, P, F, R)_u
\end{aligned}$$

$$\begin{aligned}
&(\mathrm{choose}\ n_1\ \ n_2, K, P, \emptyset, R)_u && \rightarrow && (\mathrm{choose}\ n_1\ \ n_2, K, P, 1.\emptyset, R)_u \\
&(\mathrm{choose}\ n_1\ \ n_2, K, P, (i.F), R)_u && \rightarrow && (n_i, K, (P.i), F, R)_u \\
&(n, \mathrm{halt}, P, \emptyset, R)_u && \rightarrow && (u, \mathrm{halt}, \emptyset, P{+}1, n.R)_u
\end{aligned}$$

History machines

$$\boxed{(t, K, P, F, R)_u} \qquad (t, \mathsf{halt}, \emptyset, \emptyset, \emptyset)_t$$

$$\begin{array}{ll}
(\mathsf{S}\ t, K, P, F, R)_u & \rightarrow (t, \mathsf{S}\ K, P, F, R)_u \\
(n, \mathsf{S}\ K, P, F, R)_u & \rightarrow (n+1, K, P, F, R)_u \\
(\mathsf{let}\ x\ =\ t\ \mathsf{in}\ t', K, P, F, R)_u & \rightarrow (t, \mathsf{let}\ x\ =\ \Box\ \mathsf{in}\ (t', K), P, F, R)_u \\
(n, \mathsf{let}\ x\ =\ \Box\ \mathsf{in}\ (t', K), P, F, R)_u & \rightarrow (t'[x \leftarrow n], K, P, F, R)_u
\end{array}$$

$$\begin{array}{lcl}
(\mathsf{choose}\ n_1\ n_2, K, P, \emptyset, R)_u & \rightarrow & (\mathsf{choose}\ n_1\ n_2, K, P, 1.\emptyset, R)_u \\
(\mathsf{choose}\ n_1\ n_2, K, P, (i.F), R)_u & \rightarrow & (n_i, K, (P.i), F, R)_u \\
(n, \mathsf{halt}, P, \emptyset, R)_u & \rightarrow & (u, \mathsf{halt}, \emptyset, P{+}1, n.R)_u
\end{array}$$

$$\begin{array}{lcl}
P.1{+}1 & \stackrel{\text{def}}{=} & P.2 \\
P.2{+}1 & \stackrel{\text{def}}{=} & P{+}1
\end{array}$$

Proof

Theorem

$$(t, K, \emptyset, \emptyset) \to (n, \mathsf{halt}, \emptyset, R)$$

$$\Longrightarrow$$

$$(t, K, \emptyset, \emptyset, \emptyset)_t \to (n, \mathsf{halt}, 2^\star, \emptyset, R)$$

$$\begin{array}{lcl} (n, \mathsf{halt}, \emptyset, (n', K).s, R) & \to & (n', K, \emptyset, s, n.R) \\ (n, \mathsf{halt}, P, \emptyset, R)_u & \to & (u, \mathsf{halt}, \emptyset, P{+}1, n.R)_u \end{array}$$

$$(n, \mathsf{halt}, P, \emptyset, R)_u \to (u, \mathsf{halt}, \emptyset, P{+}1, n.R)_u \to^* (n', K, \emptyset, n.R)_u$$

$$\boxed{(t, K_P, F, s, R)_u}$$

(Witty transition slide)

Section 3

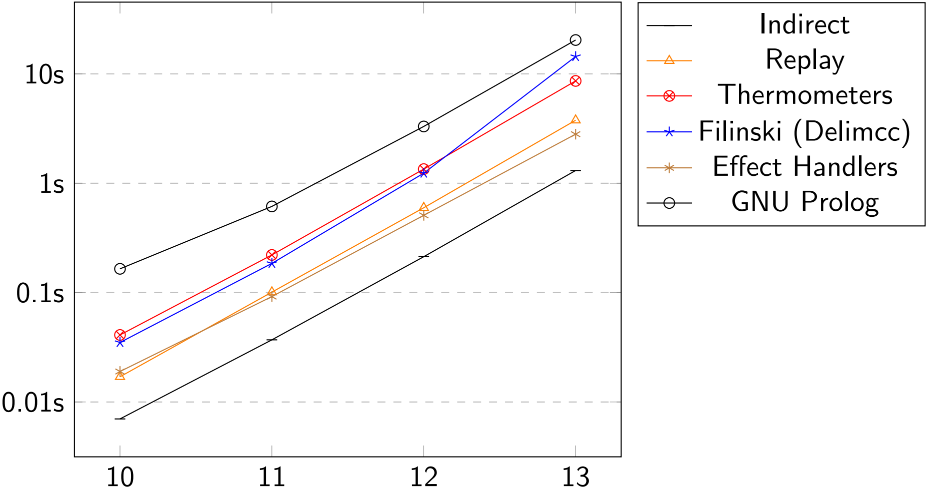
Benchmarks!

Worst case is very bad

```
with_choice (fun () →
  let v = long_pure_computation () in
  let i = choose [0; 1; 2; 3; 4; 5; 6; 7; 8; 9] in
  (i, v)
)
```

N queens

```
let n = int_of_string Sys.argv.(1)
let range = Array.init n (fun i → i) |> Array.to_list

let okay qs q =
  let rec okay i c = function
    | [] → true
    | x::xs →
      c <> x && (c−x) <> i && (c−x) <> −i && okay (i+1) c xs
  in okay 1 q qs

let rec enum_nqueens i qs =
  if i = n then qs else
    let q = choose (List.filter (okay qs) range) in
    enum_nqueens (i+1) (q :: qs)

let nb_sols = List.length (with_choice (fun () → enum_queens 0 []))
```



10s
1s
0.1s
0.01s
10
11
12
13
Indirect
Replay
Thermometers
Filinski (Delimcc)
Effect Handlers
GNU Prolog

Thanks. Any more questions?

```
queens(N, N, L, L).
queens(N, I, L, Res) :-
  I < N,
  choose_okay_in_range(0, N, C, L),
  I1 is I+1,
  queens(N, I1, [C|L], Res).

choose_okay_in_range(I, N, I, L) :- I < N, okay(1, I, L).
choose_okay_in_range(I, N, C, L) :-
  I < N, I1 is I+1, choose_okay_in_range(I1, N, C, L).

okay(_, _, []).
okay(I, C, [X|XS]) :-
  C =\= X, (C-X) =\= I, (X-C) =\= I, I1 is I+1, okay(I1, C, XS).

count(N, Count) :- aggregate_all(count, queens(N, 0, [], L), Count).
```

	10	**11**	**12**	**13**
Indirect	0.007s	0.037s	0.213s	1.308s
Replay	0.017s	0.101s	0.597s	3.768s
Therm.	0.041s	0.221s	1.347s	8.621s
Filinski (Delimcc)	0.035s	0.185s	1.236s	14.412s
Effect Handlers (Multicore OCaml)	0.019s	0.092s	0.509s	2.81s
Prolog search (GNU Prolog)	0.165s	0.614s	3.307s	20.401s

Proof: combined machines

$$\boxed{(t, K_P, F, s, R)_u} \qquad (t, \mathsf{halt}_\emptyset, \emptyset, \emptyset, \emptyset)_t$$

Proof: combined machines

$$\boxed{(t, K_P, F, s, R)_u} \qquad (t, \mathsf{halt}_\emptyset, \emptyset, \emptyset, \emptyset)_t$$

$$\begin{array}{lcl}
(\text{choose } n_1\ n_2, K_P, \emptyset, s, R)_u & \rightarrow & (n_1, K_{P.1}, \emptyset, (n_2, K_{P.2}).s, R)_u \\
(\text{choose } n_1\ n_2, K_P, i.F, s, R)_u & \rightarrow & (n_i, K_{P.i}, F, s, R)_u \\
(n, \mathsf{halt}_P, \emptyset, (n', K_{P'}).s, R)_u & \rightarrow & (n', K_{P'}, \emptyset, s, n.R)_u
\end{array}$$

Proof: combined machines

$$\boxed{(t, K_P, F, s, R)_u} \qquad (t, \mathrm{halt}_\emptyset, \emptyset, \emptyset, \emptyset)_t$$

$$\begin{array}{lcl}
(\text{choose } n_1\ n_2, K_P, \emptyset, s, R)_u & \rightarrow & (n_1, K_{P.1}, \emptyset, (n_2, K_{P.2}).s, R)_u \\
(\text{choose } n_1\ n_2, K_P, i.F, s, R)_u & \rightarrow & (n_i, K_{P.i}, F, s, R)_u \\
(n, \mathrm{halt}_P, \emptyset, (n', K_{P'}).s, R)_u & \rightarrow & (n', K_{P'}, \emptyset, s, n.R)_u
\end{array}$$

$$\begin{array}{lcl}
(\text{choose } n_1\ n_2, K, s, R) & \rightarrow & (n_1, K, (n_2, K).s, R) \\
(n, \mathrm{halt}, (n', K).s, R) & \rightarrow & (n', K, s, n.R)
\end{array}$$

$$\begin{array}{lcl}
(\text{choose } n_1\ n_2, K, P, \emptyset, R)_u & \rightarrow & (\text{choose } n_1\ n_2, K, P, 1.\emptyset, R)_u \\
(\text{choose } n_1\ n_2, K, P, (i.F), R)_u & \rightarrow & (n_i, K, (P.i), F, R)_u \\
(n, \mathrm{halt}, P, \emptyset, R)_u & \rightarrow & (u, \mathrm{halt}, \emptyset, P{+}1, n.R)_u
\end{array}$$

Proof: timeline and replay

$$\begin{array}{lcl}(n, \mathsf{halt}, P, \emptyset, R)_u & \rightarrow & (u, \mathsf{halt}, \emptyset, P{+}1, n.R)_u \\ (n, \mathsf{halt}_P, \emptyset, (n', K_{P'}).s, R)_u & \rightarrow & (n', K_{P'}, \emptyset, s, n.R)_u\end{array}$$

$$(n, \mathsf{halt}_P, \emptyset, (n', K_{P'}).s, R)_u \rightarrow (u, \mathsf{halt}_\emptyset, P', s, n.R)_u \rightarrow^* (n', K_{P'}, \emptyset, s, n.R)_u$$

Proof: timeline and replay

$$\begin{array}{lcl} (n, \mathsf{halt}, P, \emptyset, R)_u & \rightarrow & (u, \mathsf{halt}, \emptyset, P{+}1, n.R)_u \\ (n, \mathsf{halt}_P, \emptyset, (n', K_{P'}).s, R)_u & \rightarrow & (n', K_{P'}, \emptyset, s, n.R)_u \end{array}$$

$$(n, \mathsf{halt}_P, \emptyset, (n', K_{P'}).s, R)_u \rightarrow (u, \mathsf{halt}_\emptyset, P', s, n.R)_u \rightarrow^* (n', K_{P'}, \emptyset, s, n.R)_u$$

Timeline Invariant:

$$P' = P{+}1$$

$$(\text{choose } n_1 \ n_2, K_P, \emptyset, s, R)_u \quad \rightarrow \quad (n_1, K_{P.1}, \emptyset, (n_2, K_{P.2}).s, R)_u$$

Proof: timeline and replay

$$\begin{array}{lcl} (n, \mathsf{halt}, P, \emptyset, R)_u & \rightarrow & (u, \mathsf{halt}, \emptyset, P{+}1, n.R)_u \\ (n, \mathsf{halt}_P, \emptyset, (n', K_{P'}).s, R)_u & \rightarrow & (n', K_{P'}, \emptyset, s, n.R)_u \end{array}$$

$$(n, \mathsf{halt}_P, \emptyset, (n', K_{P'}).s, R)_u \rightarrow (u, \mathsf{halt}_\emptyset, P', s, n.R)_u \rightarrow^* (n', K_{P'}, \emptyset, s, n.R)_u$$

Timeline Invariant:

$$P' = P{+}1$$

$$(\mathsf{choose}\ n_1\ n_2, K_P, \emptyset, s, R)_u \quad \rightarrow \quad (n_1, K_{P.1}, \emptyset, (n_2, K_{P.2}).s, R)_u$$

Replay Theorem:

$$\mathrm{replay}(n, K_P, F, s, R)_u \overset{\mathrm{def}}{=} (u, \mathsf{halt}_\emptyset, (P.F), s, R)_u$$

$$(t, \mathsf{halt}_\emptyset, \emptyset, \emptyset)_t \rightarrow^* c \quad \Longrightarrow \quad \mathrm{replay}(c) \rightarrow^*_{\mathrm{pure}} c$$