Practical Conversion from CPS to Direct Style

Kavon Farvardin and John Reppy

University of Chicago

MWPLS

December 2, 2016

CPS is great for compilers

- Evaluation order is made explicit.
- Control-flow is regularized.
- Useful for both high-level and low-level representations.
- Easily supports non-local control-flow; exceptions, call/cc, *etc.*

Bringing Continuations to LLVM

- Ongoing work to explore implementations of continuations.
- Native codegen is a pain; using LLVM is easier.
- Recent work: heap-allocated, first-class conts with LLVM
- How can a CPS-based compiler use LLVM with a stack?

CPS with a stack

```
fun fib (n, k) =
  if n <= 2 then k 1
  else
    cont minus2 f_1 =
      cont add f_2 =
        k (f_1 + f_2)
      in
        fib (n-2, add)
    in
      fib (n-1, minus2)
```

minus2's closure dies here

Downward funargs

Undoing CPS *in theory*

Key Observation*

Most continuations created by CPS are well-behaved.

* by Danvy, Kelsey, *etc.*

Undoing CPS *in practice*

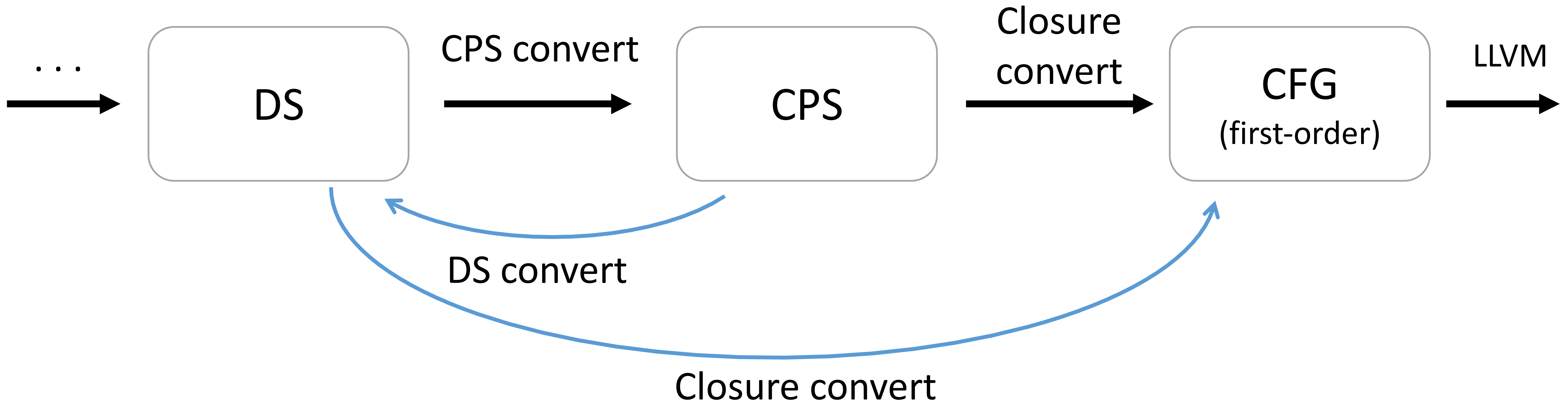
It starts with a good intermediate representation:

- Continuations and functions are different.
- Continuation parameters added by CPS are distinguished.

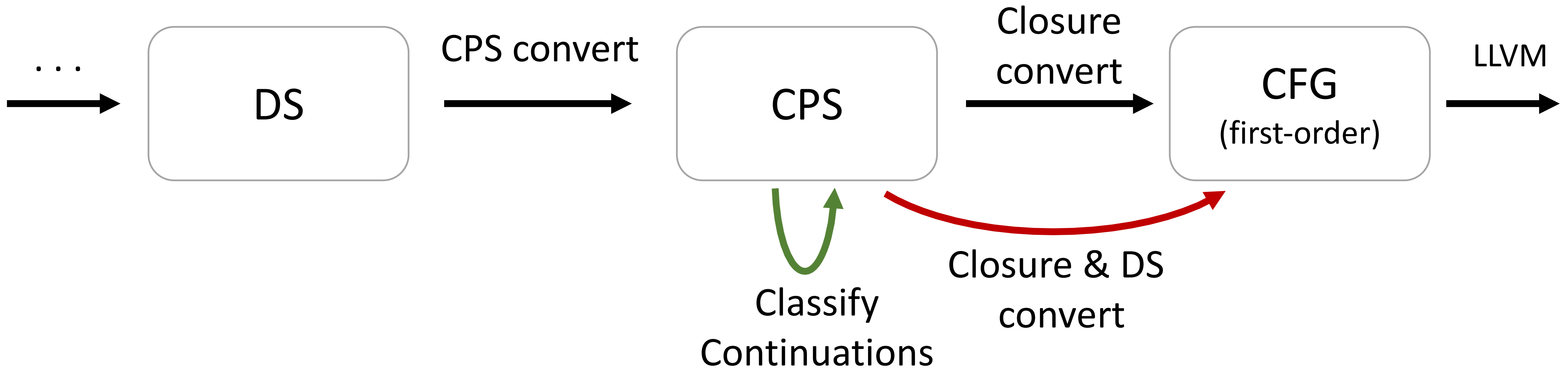
```
cont k () = _ in _        <->  throw k ()

fun f (x, y / k) = _      <->  f (1, 2 / k')
```

Noninvasive Compiler Upgrades



Noninvasive Compiler Upgrades



Classifying Continuations

Higher-order DS

```
fun g x = x
fun f x y = if x > 10 then h((g x) + y) else h x
```

```
fun g (x / k) = throw k x            ← Return throw
fun f (x, y / k) =
  cont doH z = h (z + y / k) in
  if x > 10
    then g (x / doH)                 ← Non-tail call
    else h (x / k)                   ← Tail call
```

CPS

Return continuations are only ever used or passed from the same function.

Converting to Direct Style

Higher-order CPS

```
fun g (x / k) = throw k x
fun f (x, y / k) =
  cont doH z = h (z + y / k) in
  if x > 10
    then g (x / doH)
    else h (x / k)
```

First-order DS

```
fun g (_, x) = return x
fun f (ep, x, y) =
  block doH (ep, z, y) =
    tailcall h (z + y)
  if x > 10
    then z = call g x
              goto doH (ep, z, y)
    else tailcall h x
```

Taming CPS Optimizations

- Arity raising
- Expansive inlining
- … maybe others?

```
fun foo t = let
   val x = #1(t)
   val t = #2(t)
   ...
```

Unbox tuple

```
fun foo x y = let
    ...
```

Taming CPS Optimizations

```
fun foo (_ / fooRet) =
  fun bar ( / barRet) = throw barRet ()

  fun g(_ / gRet) =
     if ...
     then bar( / gRet)
     else throw gRet ()

  cont joinK () =
    ...
    throw fooRet ()
  in
     g (_ / joinK)
```

Taming CPS Optimizations

```
fun foo (_ / fooRet) =
  fun bar ( / barRet) = throw barRet ()

  fun g(_ / gRet) =
     if ...
     then bar( / gRet)
     else throw gRet ()

  cont joinK () =
    ...
    throw fooRet ()
  in
     g (_ / joinK)
```

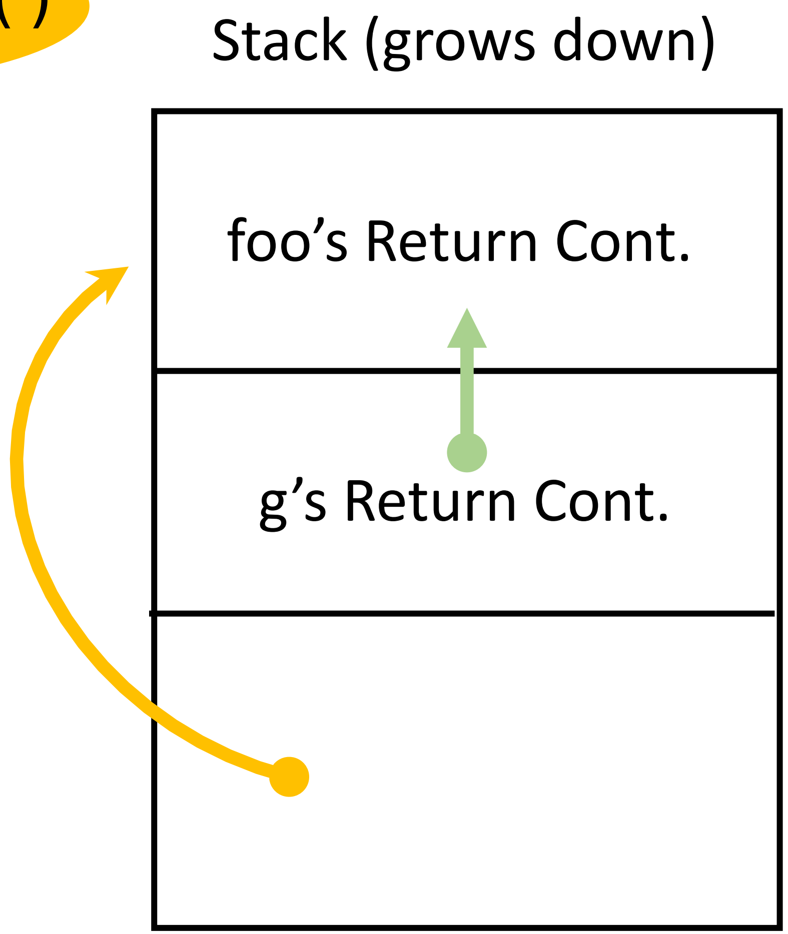
CFA says `barRet = joinK`,
so we inline the throw to `barRet`.

Taming CPS Optimizations

```
fun foo (_ / fooRet) =
  fun bar ( / ) = ... throw fooRet ()

  fun g(_ / gRet) =
     if ...
     then bar( / )
     else throw gRet ()

  cont joinK () =
    ...
    throw fooRet ()
  in
     g (_ / joinK)
```



Conclusion and Ongoing Work

- Direct style conversion can be done easily during closure conversion.
- Ongoing Work
  - Dedicated stack-based cont primitives (newStack, *etc.*)
  - Extending LLVM to support first-class stack-based conts.