Join Continuations

Consider:

```
treeprod = rec(λf.λtree.
                    if(isnat(tree),
                       tree,
                       f(left(tree))*f(right(tree))))
```

which multiplies all leaves of a tree, which are numbers.

You can do the “left” and “right” computations concurrently.

Tree Product Behavior

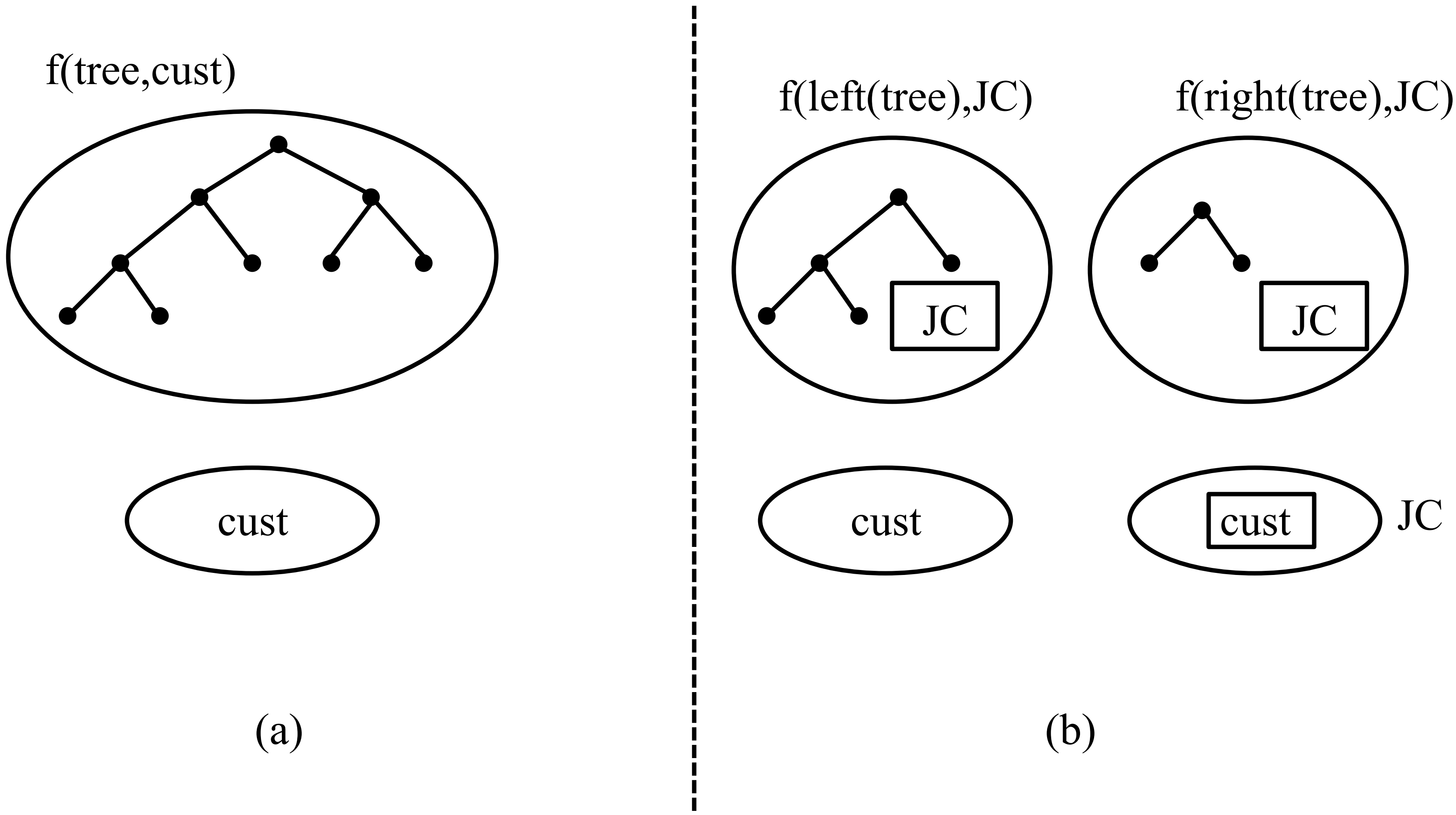
$B_{treeprod}$ =

```
  rec(λb.λm.
      seq(if(isnat(tree(m)),
             send(cust(m),tree(m)),
             let newcust=new(B_joincont(cust(m))),
                 lp = new(B_treeprod),
                 rp = new(B_treeprod) in
             seq(send(lp,
                 pr(left(tree(m)),newcust)),
                send(rp,
                 pr(right(tree(m)),newcust)))),
         ready(b)))
```

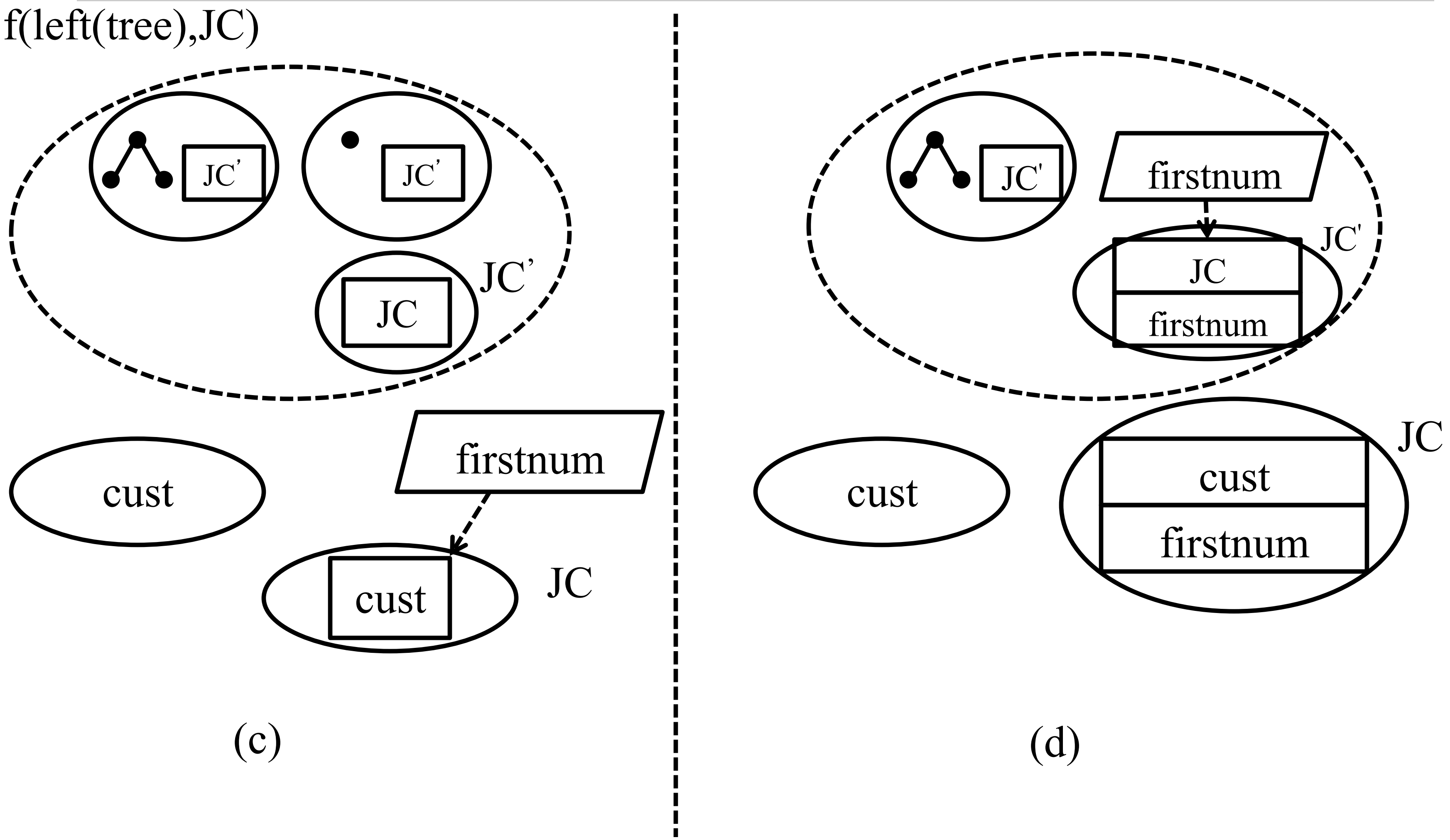
Tree Product (continued)

```
Bjoincont =
  λcust.λfirstnum.ready(λnum.
           seq(send(cust,firstnum*num),
                   ready(sink)))
```

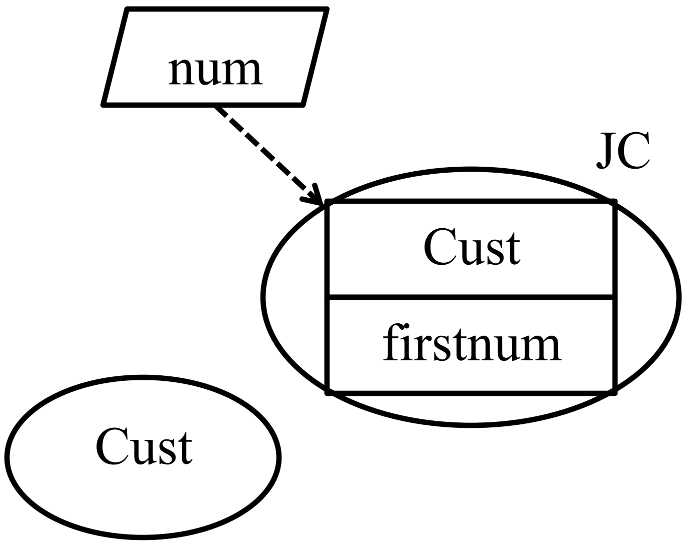
Sample Execution



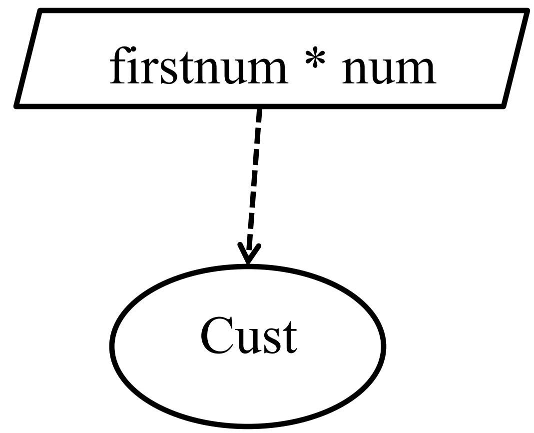
Sample Execution



Sample Execution



(e)



(f)

Operational Semantics for AMST Actor Language

- Operational semantics of actor model as a labeled transition relationship between actor configurations.
- Actor configurations model open system components:
  - Set of individually named actors
  - Messages "en-route"

Actor Configurations

$$\mathbf{k} = \boldsymbol{\alpha} \,||\, \boldsymbol{\mu}$$

$\boldsymbol{\alpha}$ is a function mapping actor names (represented as free variables) to actor states.

$\boldsymbol{\mu}$ is a multi-set of messages "en-route."

Syntactic restrictions on configurations

Given $A = Dom(\alpha)$:

- If a in A, then $fv(\alpha(a))$ is a subset of A.
- If $<a <= v>$ in μ, then $\{a\} \cup fv(v)$ is a subset of A.

Reduction contexts and redexes

Consider the expression:

`e = send(new(b5), a)`

- The redex `r` represents the next sub-expression to evaluate in a left-first call-by-value evaluation strategy.
- The reduction context `R` (or *continuation*) is represented as the surrounding expression with a *hole* replacing the redex.

`send(new(b5), a) = send(□,a)▶new(b5)◀`

`e = R▶r◀` where

`R = send(□,a)`

`r = new(b5)`

Labeled Transition Relation

$$\frac{e \rightarrow_{\lambda} e'}{\alpha, [\mathsf{R} \blacktriangleright e \blacktriangleleft]_a \parallel \mu \xrightarrow{[\mathbf{fun}:a]} \alpha, [\mathsf{R} \blacktriangleright e' \blacktriangleleft]_a \parallel \mu}$$

$$\alpha, [\mathsf{R} \blacktriangleright \mathtt{new}(b) \blacktriangleleft]_a \parallel \mu \xrightarrow{[\mathbf{new}:a,a']} \alpha, [\mathsf{R} \blacktriangleright a' \blacktriangleleft]_a, [\mathtt{ready}(b)]_{a'} \parallel \mu$$

a' fresh

$$\alpha, [\mathsf{R} \blacktriangleright \mathtt{send}(a', v) \blacktriangleleft]_a \parallel \mu \xrightarrow{[\mathbf{snd}:a]} \alpha, [\mathsf{R} \blacktriangleright \mathtt{nil} \blacktriangleleft]_a \parallel \mu \uplus \{\langle a' \Leftarrow v \rangle\}$$

$$\alpha, [\mathsf{R} \blacktriangleright \mathtt{ready}(b) \blacktriangleleft]_a \parallel \{\langle a \Leftarrow v \rangle\} \uplus \mu \xrightarrow{[\mathbf{rcv}:a,v]} \alpha, [b(v)]_a \parallel \mu$$

Exercises

- PDCS Exercise 4.6.6 (page 77).
- Using the AMST actor language operational semantics, illustrate the transitions from the following actor configuration:

  $$\kappa_0 = \alpha,[\texttt{send(new(}B_{\texttt{treeprod}}\texttt{), pr(pr(5,7)},c\texttt{)}\,)]_a \parallel \varnothing$$

- PDCS Exercise 4.6.7 (page 78).