COP-5555 PROGRAMMING LANGUAGE PRINCIPLES

NOTES ON RECURSION AND FIXED-POINT THEORY.

Our purpose is to describe the meaning, or the semantics, of the keyword "rec" in RPAL. As is often the case, an analogy is useful. The meaning of an RPAL program of the form "let x=P in Q", as we now know, is the value of applicative expression "(λx.Q)P". In fact, "let x=P in Q" is not the only program with that meaning (for example consider "Q where x=P"). Since all RPAL programs (except those containing "rec") can be similarly transformed to λ-expressions whose evaluation will yield the value of the program, we can say that RPAL programs are "syntactically sugared" λ-expressions, i.e. RPAL programs are simply λ-expressions, presented in another form. Thus the question:

What is the meaning of an RPAL program of the form "let rec f n = P in Q" ?

can be re-phrased as:

What λ-expression, if any, is the "de-sugarized" version of "let rec f n = P in Q" ?

To answer that question, we must first develop some intuition. Consider the most popular example of recursive computation: the factorial function:

let rec f n = n eq 0 $\rightarrow$ 1 | n*f(n-1) in f 3

If we assume the "rec" keyword were not present, then the corresponding applicative expression would be

(λf.f 3) [λn.n eq 0 $\rightarrow$ 1 | n*f(n-1)]

It is easy to see that the last occurrence of "f" in this expression is free, and that the evaluation of the expression would not succeed because of that "f" being undefined. Thus the purpose of the "rec" keyword is to "magically" make that occurrence of "f" bound to the bracketed expression. With the "rec", somehow,

	(λf.f 3) [λn.n eq 0 $\rightarrow$ 1 | n*f(n-1)]
$=>_{\beta}$	(λn.n eq 0 $\rightarrow$ 1 | n*f(n-1)) 3
$=>_{\beta}$	3 eq 0 $\rightarrow$ 1 | 3*f(3-1)
$=>_{\delta}$	3*f(2)
$=>_{magic}$	3*(λn.n eq 0 $\rightarrow$ 1 | n*f(n-1)) 2
$=>_{\beta}$	3*(2 eq 0 $\rightarrow$ 1 | 2*f(2-1))
$=>_{\delta}$	3*2*f(1)
$=>_{magic}$	3*2*(λn.n eq 0 $\rightarrow$ 1 | n*f(n-1)) 1
$=>_{\beta}$	3*2*(1 eq 0 $\rightarrow$ 1 | 1*f(1-1))
$=>_{\delta}$	3*2*1*f(0)
$=>_{magic}$	3*2*1*(λn.n eq 0 $\rightarrow$ 1 | n*f(n-1)) 0
$=>_{\beta}$	3*2*1*(0 eq 0 $\rightarrow$ 1 | 0*f(0-1))
$=>_{\delta}$	3*2*1*1
$=>_{\delta}$	6

To dispel the magic, we need some mathematics.

Definition:

Let F be a function and w a value. w is called a **fixed-point** of F iff F w = w.

Example 1:

$S = \lambda x.x^2 - 6$ has two fixed points, 3 and -2, because

$$(\lambda x.x^2 - 6)\ 3 \ =>_\beta 3^2 - 6 \ =>_\delta 3$$
$$(\lambda x.x^2 - 6)\ -2 \ =>_\beta (-2)^2 - 6 \ =>_\delta -2$$

There are no more fixed-points, because -2 and 3 are the only solutions to $x^2 - x - 6 = 0$.

Example 2:

$T = \lambda f.\lambda().(f\ nil)^2 - 6$ has two fixed points, $\lambda().3$ and $\lambda().-2$, because

$$(\lambda f.\lambda().(f\ nil)^2 - 6)\ \lambda().3 \ =>_\beta \lambda().((\lambda().3)\ nil)^2 - 6 \ =>_\beta \lambda().3^2 - 6 \ =>_\delta \lambda().3$$
$$(\lambda f.\lambda().(f\ nil)^2 - 6)\ \lambda().-2 \ =>_\beta \lambda().((\lambda().-2)\ nil)^2 - 6 \ =>_\beta \lambda().-2^2 - 6 \ =>_\delta \lambda().-2$$

Note that fixed points can be functions. In fact, the fixed points we are interested in can **only** be functions.

Example 3:

Consider $U = \lambda f.\lambda n.n\ eq\ 0 \rightarrow 1 \mid n*f(n-1)$. The function $\lambda x.x^2$ is **not** a fixed point of U, but the Factorial function is. Proof of this is left to the reader.

Back to magic dispelling:

	let rec f n = P in Q	
=>	let rec f = λn.P in Q	(fcn_form)
=>	let rec f = (λf.λn.P)f in Q	(abstraction)
=>	let rec f = F f in Q	

where $F = \lambda f.\lambda n.P$. Note that now there are no free occurrences of f in F, and there is only one free occurrence of f in the entire expression. Further, in this last expression (i.e. let rec f = F f in Q) the presence of the "rec" means that both f's should be the same; without the "rec" they would be different. With "rec", we want f to be a fixed point of F !! Let us then re-phrase the expression as

let f = A_fixed_point_of F in Q.

This operator, "A-fixed_point_of" is called a **fixed point finder**, and we will identify it from now on as "Y". Hence our expression becomes

	let f = Y F in Q
=>	(λf.Q) (Y F)
=	(λf.Q) (Y (λf.λn.P))

Now there are no free occurrences of f !! In summary, the problem of explaining the purpose of "rec" is equivalent to finding a fixed point of F. Such a fixed point finder is called Y; we must now find a suitable Y.

How do we find a suitable Y ?

WE DON'T NEED TO !! We only need to use some of its characteristics:

1) f = Y F

2) F f = f

Substituting 1) in 2), we obtain

$$\boxed{F\ (Y\ F) = Y\ F}$$

This is called the **fixed point identity**, and it is all that is required to evaluate λ expressions that contain Y's, which are "de-sugared" RPAL programs that contain rec's. Let's extend some earlier definitions:

Definition. Axiom ρ (rho): Y F = $>_\rho$ F (Y F).

Definition. (Extension): An applicative expression M is **not** in normal form if it is of the form Y N.

Definition. (Extension): A reduction sequence is in normal order iff at every step we reduce the left-most Y or λ.

We can now return to the factorial function, and evaluate the applicative expression (again). This time, there is no magic.

$$\begin{aligned}
& (\lambda f.f\ 3)\ (Y\ F), \text{ where } F = \lambda f.\lambda n.n \text{ eq } 0 \rightarrow 1 \mid n*f(n-1) \\
=>_\beta\ & Y\ F\ 3 \\
=>_\rho\ & F\ (Y\ F)\ 3 \\
=\ & (\lambda f.\lambda n.n \text{ eq } 0 \rightarrow 1 \mid n*f(n-1))\ (Y\ F)\ 3 \\
=>_\beta\ & (\lambda n.n \text{ eq } 0 \rightarrow 1 \mid n*\ Y\ F\ (n-1))\ 3 \\
=>_{\beta,\delta}\ & 3*\ Y\ F\ (2) \\
=>_\rho\ & 3*\ F\ (Y\ F)\ 2 \\
=\ & 3*\ (\lambda f.\lambda n.n \text{ eq } 0 \rightarrow 1 \mid n*f(n-1))\ (Y\ F)\ 2 \\
=>_\beta\ & 3*\ (\lambda n.n \text{ eq } 0 \rightarrow 1 \mid n*\ Y\ F\ (n-1))\ 2 \\
=>_{\beta,\delta}\ & 3*2*\ Y\ F\ (1) \\
=>_\rho\ & 3*2*\ F\ (Y\ F)\ 1 \\
=\ & 3*2*\ (\lambda f.\lambda n.n \text{ eq } 0 \rightarrow 1 \mid n*f(n-1))\ (Y\ F)\ 1 \\
=>_\beta\ & 3*2*\ (\lambda n.n \text{ eq } 0 \rightarrow 1 \mid n*\ Y\ F\ (n-1))\ 1 \\
=>_{\beta,\delta}\ & 3*2*1*\ Y\ F\ (0) \\
=>_\rho\ & 3*2*1*\ F\ (Y\ F)\ 0 \\
=\ & 3*2*1*\ (\lambda f.\lambda n.n \text{ eq } 0 \rightarrow 1 \mid n*f(n-1))\ (Y\ F)\ 0 \\
=>_\beta\ & 3*2*1*\ (\lambda n.n \text{ eq } 0 \rightarrow 1 \mid n*\ Y\ F\ (n-1))\ 0 \\
=>_{\beta,\delta}\ & 3*2*1*1 \\
=>_\delta\ & 6
\end{aligned}$$

Notice that normal order is required to evaluate the expression; in programming language order the evaluation would not terminate.

We have now seen Y at work, but the question is, "Is YF really the factorial function ?" In other words, we have seen that YF (for this particular F) can be used to compute factorial of three, but can it be used to compute factorial of k, for any k? Let's try it by induction.

Base Case:

$$\begin{aligned}
Y\ F\ 0\ =>_\rho\ & F\ (Y\ F)\ 0 \\
=\ & (\lambda f.\lambda n.n \text{ eq } 0 \rightarrow 1 \mid n*f(n-1))\ (Y\ F)\ 0
\end{aligned}$$

$$
\begin{aligned}
&= >_{\beta} \quad (\lambda n.n \text{ eq } 0 \rightarrow 1 \mid n\ (Y\ F)\ (n\text{-}1))\ 0 \\
&= >_{\beta,\delta} \quad 1
\end{aligned}
$$

Inductive step: for k > 0,

$$
\begin{aligned}
Y\ F\ k &= >_{\rho} \quad F\ (Y\ F)\ k \\
&= \quad (\lambda f.\lambda n.n \text{ eq } 0 \rightarrow 1 \mid n*f(n\text{-}1))\ (Y\ F)\ k \\
&= >_{\beta} \quad (\lambda n.n \text{ eq } 0 \rightarrow 1 \mid n*(Y\ F)\ (n\text{-}1))\ k \\
&= >_{\beta} \quad k \text{ eq } 0 \rightarrow 1 \mid k * (Y\ F)\ (k\text{-}1) \\
&= >_{\delta} \quad k * (Y\ F)\ (k\text{-}1)
\end{aligned}
$$

Recursion via lambda-calculus.

Recursion can be acheived using the fixed-point identity. The question now is whether it can also be acheived using lambda-calculus alone, i.e. does there exist a suitable λ-expression that can be used as the value of Y? After some trials and errors, someone came up with this:

$$Y = \lambda F.(\lambda f.f\ f)\ (\lambda g.\ F\ (g\ g))$$

This satisfies the fixed-point identity:

$$
\begin{aligned}
Y\ F &= \quad (\lambda F.(\lambda f.f\ f)\ (\lambda g.\ F(g\ g))\ )\ F \\
&= >_{\beta} \quad (\lambda f.f\ f)\ (\lambda g.F(g\ g)) \\
&= >_{\beta} \quad (\lambda g.F(g\ g))\ (\lambda g.F(g\ g)) \\
&= >_{\beta} \quad F\ (\ (\lambda g.F(g\ g))\ (\lambda g.F(g\ g))\ ) \\
&< =_{\beta} \quad F\ (\ (\lambda f.f\ f)\ (\lambda g.F(g\ g))\ ) \\
&< =_{\beta} \quad F\ (\ (\lambda F.(\lambda f.f\ f)\ (\lambda g.F(g\ g))\ )\ F\ ) \\
&= \quad F\ (Y\ F)
\end{aligned}
$$

Notice again how normal order of evaluation is required.

Recursion in the CSE Machine

We finally come to what matters: getting recursion to work. We have several alternatives for implementing recursion in the CSE machine:

1.- Use $Y = \lambda F.(\lambda f.f\ f)\ (\lambda g.F(g\ g))$. This is pretty tedious on paper, and it won't work on the CSE machine because it requires normal order of evaluation, and the CSE machine uses programming language order.

2.- Try delaying the evaluation of the arguments in Y above, in a fashion similar to the trick used for modeling the conditional. We could use $Z = \lambda F.(\lambda f.f\ f)\ (\lambda h.F(\lambda x.h\ h\ x))$, but it would be EXTREMELY tedious.

3.- Try a variation of the fixed-point identity, our method of choice.

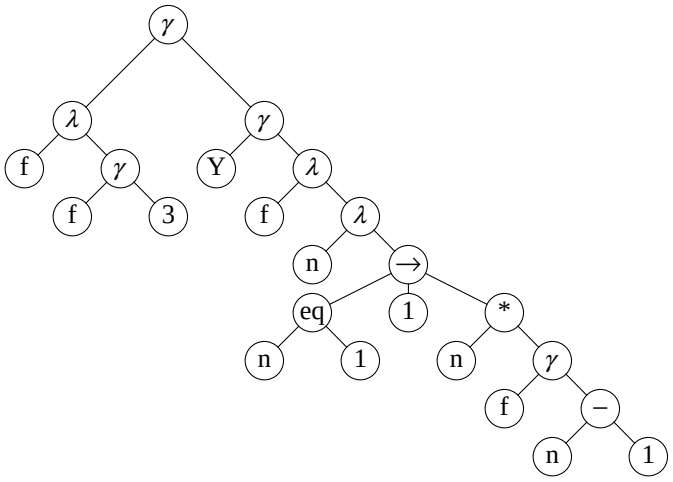
	CONTROL	STACK	ENV
CSE Rule 12	.... γ	Y $^{c}\lambda_i^{\nu}$	
	....	$^{c}\eta_i^{\nu}$	

CSE Rule 13	.... γ	${}^{c}\eta_i^{\nu}$ R ....
	.... γ γ	${}^{c}\lambda_i^{\nu}$ ${}^{c}\eta_i^{\nu}$ R ....

Example (the factorial function):

let rec f n = n eq 1 -> 1 | n * f (n-1) in f 3

Standardized tree:



Control Structures:

$\delta_0 = \gamma\ \lambda_1^{f}\ \gamma$ Y λ_2^{f}
$\delta_1 = \gamma$ f 3
$\delta_2 = \lambda_3^{n}$
$\delta_3 = \delta_4\ \delta_5\ \beta$ eq n 1
$\delta_4 = 1$
$\delta_5 = *$ n γ f - n 1

RULE	CONTROL	STACK	ENV
2	$e_0\ \gamma\ \lambda_1^{f}\ \gamma$ Y λ_2^{f}	e_0	e_0=PE
1	$e_0\ \gamma\ \lambda_1^{f}\ \gamma$ Y	${}^{0}\lambda_2^{f}\ e_0$	
12	$e_0\ \gamma\ \lambda_1^{f}\ \gamma$	Y ${}^{0}\lambda_2^{f}\ e_0$	
2	$e_0\ \gamma\ \lambda_1^{f}$	${}^{0}\eta_2^{f}\ e_0$	
4	$e_0\ \gamma$	${}^{0}\lambda_1^{f}\ {}^{0}\eta_2^{f}\ e_0$	
1	$e_0\ e_1\ \gamma$ f 3	$e_1\ e_0$	$e_1=[{}^{0}\eta_2^{f}/f]e_0$
1	$e_0\ e_1\ \gamma$ f	3 $e_1\ e_0$	
13	$e_0\ e_1\ \gamma$	${}^{0}\eta_2^{f}$ 3 $e_1\ e_0$	
4	$e_0\ e_1\ \gamma\ \gamma$	${}^{0}\lambda_2^{f}\ {}^{0}\eta_2^{f}$ 3 $e_1\ e_0$	

2	$e_0\ e_1\ \gamma\ e_2\ \lambda_3^n$	$e_2\ 3\ e_1\ e_0$	$e_2=[{}^0\eta_2^f/f]e_0$
5	$e_0\ e_1\ \gamma\ e_2$	${}^2\lambda_3^n\ e_2\ 3\ e_1\ e_0$	
4	$e_0\ e_1\ \gamma$	${}^2\lambda_3^n\ 3\ e_1\ e_0$	
1	$e_0\ e_1\ e_3\ \delta_4\ \delta_5\ \beta$ eq n 1	$e_3\ e_1\ e_0$	$e_3=[3/n]e_2$
1	$e_0\ e_1\ e_3\ \delta_4\ \delta_5\ \beta$ eq n	$1\ e_3\ e_1\ e_0$	
6	$e_0\ e_1\ e_3\ \delta_4\ \delta_5\ \beta$ eq	$3\ 1\ e_3\ e_1\ e_0$	
8	$e_0\ e_1\ e_3\ \delta_4\ \delta_5\ \beta$	false $e_3\ e_1\ e_0$	
1	$e_0\ e_1\ e_3$ * n γ f - n 1	$e_3\ e_1\ e_0$	
1	$e_0\ e_1\ e_3$ * n γ f - n	$1\ e_3\ e_1\ e_0$	
6	$e_0\ e_1\ e_3$ * n γ f -	$3\ 1\ e_3\ e_1\ e_0$	
1	$e_0\ e_1\ e_3$ * n γ f	$2\ e_3\ e_1\ e_0$	
13	$e_0\ e_1\ e_3$ * n γ	${}^0\eta_2^f\ 2\ e_3\ e_1\ e_0$	
4	$e_0\ e_1\ e_3$ * n $\gamma\ \gamma$	${}^0\lambda_2^f\ {}^0\eta_2^f\ 2\ e_3\ e_1\ e_0$	
2	$e_0\ e_1\ e_3$ * n $\gamma\ e_4\ \lambda_3^n$	$e_4\ 2\ e_3\ e_1\ e_0$	$e_4=[{}^0\eta_2^f/f]e_0$
5	$e_0\ e_1\ e_3$ * n $\gamma\ e_4$	${}^4\lambda_3^n\ e_4\ 2\ e_3\ e_1\ e_0$	
4	$e_0\ e_1\ e_3$ * n γ	${}^4\lambda_3^n\ 2\ e_3\ e_1\ e_0$	
1	$e_0\ e_1\ e_3$ * n $e_5\ \delta_4\ \delta_5\ \beta$ eq n 1	$e_5\ e_3\ e_1\ e_0$	$e_5=[2/n]e_4$
1	$e_0\ e_1\ e_3$ * n $e_5\ \delta_4\ \delta_5\ \beta$ eq n	$1\ e_5\ e_3\ e_1\ e_0$	
6	$e_0\ e_1\ e_3$ * n $e_5\ \delta_4\ \delta_5\ \beta$ eq	$2\ 1\ e_5\ e_3\ e_1\ e_0$	
8	$e_0\ e_1\ e_3$ * n $e_5\ \delta_4\ \delta_5\ \beta$	false $e_5\ e_3\ e_1\ e_0$	
1	$e_0\ e_1\ e_3$ * n e_5 * n γ f - n 1	$e_5\ e_3\ e_1\ e_0$	
1	$e_0\ e_1\ e_3$ * n e_5 * n γ f - n	$1\ e_5\ e_3\ e_1\ e_0$	
6	$e_0\ e_1\ e_3$ * n e_5 * n γ f -	$2\ 1\ e_5\ e_3\ e_1\ e_0$	
1	$e_0\ e_1\ e_3$ * n e_5 * n γ f	$1\ e_5\ e_3\ e_1\ e_0$	
13	$e_0\ e_1\ e_3$ * n e_5 * n γ	${}^0\eta_2^f\ 1\ e_5\ e_3\ e_1\ e_0$	
4	$e_0\ e_1\ e_3$ * n e_5 * n $\gamma\ \gamma$	${}^0\lambda_2^f\ {}^0\eta_2^f\ 1\ e_5\ e_3\ e_1\ e_0$	
2	$e_0\ e_1\ e_3$ * n e_5 * n $\gamma\ e_6\ \lambda_3^n$	$e_6\ 1\ e_5\ e_3\ e_1\ e_0$	$e_6=[{}^0\eta_2^f/f]e_0$
5	$e_0\ e_1\ e_3$ * n e_5 * n $\gamma\ e_6$	${}^6\lambda_3^n\ e_6\ 1\ e_5\ e_3\ e_1\ e_0$	
4	$e_0\ e_1\ e_3$ * n e_5 * n γ	${}^6\lambda_3^n\ 1\ e_5\ e_3\ e_1\ e_0$	
1	$e_0\ e_1\ e_3$ * n e_5 * n $e_7\ \delta_4\ \delta_5\ \beta$ eq n 1	$e_7\ e_5\ e_3\ e_1\ e_0$	$e_7=[1/n]e_6$
1	$e_0\ e_1\ e_3$ * n e_5 * n $e_7\ \delta_4\ \delta_5\ \beta$ eq n	$1\ e_7\ e_5\ e_3\ e_1\ e_0$	
6	$e_0\ e_1\ e_3$ * n e_5 * n $e_7\ \delta_4\ \delta_5\ \beta$ eq	$1\ 1\ e_7\ e_5\ e_3\ e_1\ e_0$	
8	$e_0\ e_1\ e_3$ * n e_5 * n $e_7\ \delta_4\ \delta_5\ \beta$	true $e_7\ e_5\ e_3\ e_1\ e_0$	
1	$e_0\ e_1\ e_3$ * n e_5 * n e_7 1	$e_7\ e_5\ e_3\ e_1\ e_0$	
5	$e_0\ e_1\ e_3$ * n e_5 * n e_7	$1\ e_7\ e_5\ e_3\ e_1\ e_0$	
1	$e_0\ e_1\ e_3$ * n e_5 * n	$1\ e_5\ e_3\ e_1\ e_0$	
6	$e_0\ e_1\ e_3$ * n e_5 *	$2\ 1\ e_5\ e_3\ e_1\ e_0$	
5	$e_0\ e_1\ e_3$ * n e_5	$2\ e_5\ e_3\ e_1\ e_0$	
1	$e_0\ e_1\ e_3$ * n	$2\ e_3\ e_1\ e_0$	
6	$e_0\ e_1\ e_3$ *	$3\ 2\ e_3\ e_1\ e_0$	
5	$e_0\ e_1\ e_3$	$6\ e_3\ e_1\ e_0$	
5	$e_0\ e_1$	$6\ e_1\ e_0$	
5	e_0	$6\ e_0$	
-	-	6	