COP-5555 PROGRAMMING LANGUAGE PRINCIPLES

NOTES ON THE MECHANICAL EVALUATION OF APPLICATIVE EXPRESSSIONS

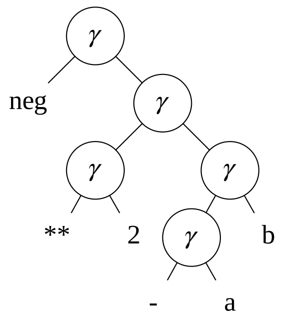
The substitution mechanism for evaluating applicative expressions is convenient for humans, but inconvenient for machines, since the arguments involved may be arbitrarily complex. Here we introduce a mechanical means for evaluating applicative expressions: the CSE Machine. It is an abstract machine (implemented via software in the PAL compiler-interpreter) with the following components:

C	- Control	- contains a sequence of operations
S	- Stack	- contains operands
E	- Environment	- Initially, PE. Updated as evaluation proceeds
M	- Machine	

The primitive environment (PE) is assumed to be a collection of objects and operations that behave in accordance with a common-sense set of rules. Actually, another mechanism is used to specify these ‘‘common-sense’’ rules; it is called the grammatical axiomatization of RPAL’s Universe of Discourse. In any event, one "looks up" names in the PE (e.g. "+"); in return one obtains the "real" object, be it an integer, a truthvalue, or an operation.

We begin by taking a given RPAL program’s standardized syntax tree, and flattening it to a "control structure" (rather than to an applicative expression) by a simple pre-order tree walk.

Example: Evaluate -2 ** (a-b), in an environment in which a=6 and b=1.



Flattened structure: γ neg γ γ ** 2 γ γ - a b.

This control structure is placed on the Control of the CSE Machine, which operates (vaguely) as follows:

1) Remove the right-most item from the control.
2) If it’s a name (variable, constant, primitive operation) then lookup the name in the current environment, and push the result on the stack.
3) If it is γ, then pop the stack (obtaining the rator), pop it again (obtaining the rand), apply the rator to the rand, and push the result.
4) Stop if the control is empty: the value on the stack is the result.

In our case, recall that a=6 and b=1. We will deal later with how these names get associated with these values.

CONTROL	STACK	ENV
γ neg γ γ ** 2 γ γ - a b		PE
γ neg γ γ ** 2 γ γ - a	1	
γ neg γ γ ** 2 γ γ -	6 1	
γ neg γ γ ** 2 γ γ	Minus 6 1	
γ neg γ γ ** 2 γ	Minus6 1	
γ neg γ γ ** 2	5	
γ neg γ γ **	2 5	
γ neg γ γ	Exp 2 5	
γ neg γ	Exp2 5	
γ neg	32	
γ	Neg 32	
	-32	

Here, Minus is a function that subtracts its second argument from its first one. Minus6 is a function that subtracts its argument from six. Exp is (in a similar fashion) the exponentiation function, and Exp2 is the function that raises two to the power of its argument. The difference between an operator on the control and the operator on the stack is a subtle one: on the control, we have only the **name** of the operator (e.g. "neg", "**"). When this name surfaces on the right-most end of the control, we look it up in the primitive environment, and obtain the corresponding function. In the case of "neg", the function obtained is (arbitrarily) called Neg, and it is stacked. In general, then, the control contains only γ's and names, whereas the stack contains "real" objects (integers, truthvalues, functions, etc.).

Evaluating λ expressions.

So far, we have seen that every intermediate value in an applicative expression is actually computed, and held (at least temporarily) on the stack. In a λ-expression such as "(λx.λy.x+y) 2 3", the question is: "What is the intermediate value for "λy.x+y" ? The answer is that the value is **contextually dependent** upon the larger expression that "λy.x+y" is embedded in. In this particular case, that value can be described, in English, as follows: "the value is a function that expects an argument y, and will add y to x. The function will operate in an environment in which x=2".

In general, several environments may be created during the course of evaluating an applicative expression. In other words, different environments apply in different parts of the expression. These environments will be numbered consecutively as they are created.

We will use a single symbol to represent a λ-expression, both on the control, and on the stack. The symbol is ${}^{i}\lambda_{k}^{x}$, where i is the environment, k is the control structure of the function's body, and x is the function's bound variable. Notice that both k and x are static, i.e. they are known before the evaluation of the expression begins. i is determined during the evaluation. The λ-expression becomes a λ **closure** when its environment is determined.

Control structures (indicated by δ_i's) are generated from the standardized tree as follows:

1) Begin with δ_0: perform a pre-order walk of the standardized tree. For each node:

   a) if it is a name, add it to the current control structure.

   b) if it is a γ, add it to the current control structure.

   c) If it is a λ, add λ_k^x to the current control structure, where k is a new index, and x is the node's left kid. Generate control structure δ_k by traversing the λ node's right kid.

Examples:

Applicative expression	Control Structures
$(\lambda x.x\text{-}1)4 * 2$	$\delta_0 = \gamma\ \gamma\ *\ \gamma\ \lambda_1^x\ 4\ 2$
	$\delta_1 = \gamma\ \gamma\ -\ x\ 1$
$(\lambda x.\lambda w.x+w)\ 5\ 6$	$\delta_0 = \gamma\ \gamma\ \lambda_1^x\ 5\ 6$
	$\delta_1 = \lambda_2^w$
	$\delta_2 = \gamma\ \gamma\ +\ x\ w$
$(\lambda x.1+(\lambda w.\text{-}w)x)[(\lambda z.2*z)7]$	$\delta_0 = \gamma\ \lambda_1^x\ \gamma\ \lambda_3^z\ 7$
	$\delta_1 = \gamma\ \gamma\ +\ 1\ \gamma\ \lambda_2^w\ x$
	$\delta_2 = \gamma\ \text{neg}\ w$
	$\delta_3 = \gamma\ \gamma\ *\ 2\ z$

We will now need environment markers on both the control and the stack, and we will need to keep track of the information available in the various environments that are created.

Here are the rules for operating the CSE Machine:

	CONTROL	STACK	ENV
Initial Configuration	$e_0\ \delta_0$	e_0	e_0 = PE
CSE Rule 1 (stacking a name)	.... Name	....	where Ob=Lookup(Name,e_c)
	....	Ob	where e_c = current environment
CSE Rule 2 (stacking a λ)	.... λ_k^x	....	
	....	${}^c\lambda_k^x$	where e_c = current environment
CSE Rule 3 (Applying a rator)	.... γ	Rator Rand	
	....	Result	where Result=Apply[Rator,Rand]
CSE Rule 4 (Applying a λ-closure)	.... γ	${}^c\lambda_k^x$ Rand	e_n = [Rand/x]e_c
	.... $e_n\ \delta_k$	e_n	
CSE Rule 5 (exit from environment)	.... e_n	value e_n	
	....	value	

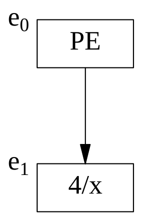
Example 1:

Applicative expression	Control Structures
$(\lambda x.x\text{-}1)4 * 2$	$\delta_0 = \gamma\ \gamma\ *\ \gamma\ \lambda_1^x\ 4\ 2$
	$\delta_1 = \gamma\ \gamma\ -\ x\ 1$

RULE	CONTROL	STACK	ENV
1	$e_0\ \gamma\ \gamma\ *\ \gamma\ \lambda_1^x\ 4\ 2$	e_0	e_0 = PE
1	$e_0\ \gamma\ \gamma\ *\ \gamma\ \lambda_1^x\ 4$	$2\ e_0$	
2	$e_0\ \gamma\ \gamma\ *\ \gamma\ \lambda_1^x$	$4\ 2\ e_0$	
4	$e_0\ \gamma\ \gamma\ *\ \gamma$	${}^0\lambda_1^x\ 4\ 2\ e_0$	
1	$e_0\ \gamma\ \gamma\ *\ e_1\ \gamma\ \gamma\ -\ x\ 1$	$e_1\ 2\ e_0$	$e_1 = [4/x]e_0$
1	$e_0\ \gamma\ \gamma\ *\ e_1\ \gamma\ \gamma\ -\ x$	$1\ e_1\ 2\ e_0$	
1	$e_0\ \gamma\ \gamma\ *\ e_1\ \gamma\ \gamma\ -$	$4\ 1\ e_1\ 2\ e_0$	
3	$e_0\ \gamma\ \gamma\ *\ e_1\ \gamma\ \gamma$	$-\ 4\ 1\ e_1\ 2\ e_0$	
3	$e_0\ \gamma\ \gamma\ *\ e_1\ \gamma$	$(-4)\ 1\ e_1\ 2\ e_0$	
5	$e_0\ \gamma\ \gamma\ *\ e_1$	$3\ e_1\ 2\ e_0$	
1	$e_0\ \gamma\ \gamma\ *$	$3\ 2\ e_0$	
3	$e_0\ \gamma\ \gamma$	$*\ 3\ 2\ e_0$	
3	$e_0\ \gamma$	$(*3)\ 2\ e_0$	
5	e_0	$6\ e_0$	
		6	

Relationships among environments.

Environments that are created during the evaluation of an applicative expression exhibit a tree structure, since every environment that is opened is linked to a previously opened (but not necessarily currently active) environment. In this particular case:

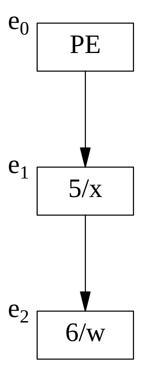


Example 2:

Applicative expression	Control Structures
$(\lambda x.\lambda w.x+w)\ 5\ 6$	$\delta_0 = \gamma\ \gamma\ \lambda_1^x\ 5\ 6$
	$\delta_1 = \lambda_2^w$
	$\delta_2 = \gamma\ \gamma\ +\ x\ w$

RULE	CONTROL	STACK	ENV
1	$e_0 \gamma\ \gamma\ \lambda_1^x\ 5\ 6$	e_0	e_0=PE
1	$e_0 \gamma\ \gamma\ \lambda_1^x\ 5$	$6\ e_0$	
2	$e_0 \gamma\ \gamma\ \lambda_1^x$	$5\ 6\ e_0$	
4	$e_0 \gamma\ \gamma$	${}^0\lambda_1^x\ 5\ 6\ e_0$	
2	$e_0 \gamma\ e_1\ \lambda_2^w$	$e_1\ 6\ e_0$	e_1=[5/x]e_0
5	$e_0 \gamma\ e_1$	${}^1\lambda_2^w\ e_1\ 6\ e_0$	
4	$e_0 \gamma$	${}^1\lambda_2^w\ 6\ e_0$	
1	$e_0\ e_2\ \gamma\ \gamma\ +\ x\ w$	$e_2\ e_0$	e_2=[6/w]e_1
1	$e_0\ e_2\ \gamma\ \gamma\ +\ x$	$6\ e_2\ e_0$	
1	$e_0\ e_2\ \gamma\ \gamma\ +$	$5\ 6\ e_2\ e_0$	
3	$e_0\ e_2\ \gamma\ \gamma$	$+\ 5\ 6\ e_2\ e_0$	
3	$e_0\ e_2\ \gamma$	$(+5)\ 6\ e_2\ e_0$	
5	$e_0\ e_2$	$11\ e_2\ e_0$	
5	e_0	$11\ e_0$	
		11	

Environment Structure:

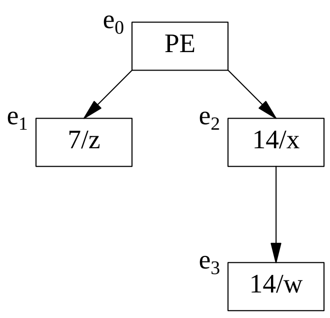


Example 3:

Applicative expression	Control Structures
$(\lambda x.1+(\lambda w.-w)x)[(\lambda z.2*z)7]$	$\delta_0 = \gamma\ \lambda_1^x\ \gamma\ \lambda_3^z\ 7$
	$\delta_1 = \gamma\ \gamma\ +\ 1\ \gamma\ \lambda_2^w\ x$
	$\delta_2 = \gamma$ neg w
	$\delta_3 = \gamma\ \gamma\ *\ 2\ z$

RULE	CONTROL	STACK	ENV
1	$e_0\ \gamma\ \lambda_1^x\ \gamma\ \lambda_3^z\ 7$	e_0	e_0=PE
2	$e_0\ \gamma\ \lambda_1^x\ \gamma\ \lambda_3^z$	$7\ e_0$	
4	$e_0\ \gamma\ \lambda_1^x\ \gamma$	${}^0\lambda_3^z\ 7\ e_0$	
1,1,1	$e_0\ \gamma\ \lambda_1^x\ e_1\ \gamma\ \gamma\ *\ 2\ z$	$e_1\ e_0$	$e_1=[7/z]e_0$
3,3	$e_0\ \gamma\ \lambda_1^x\ e_1\ \gamma\ \gamma$	$*\ 2\ 7\ e_1\ e_0$	
5	$e_0\ \gamma\ \lambda_1^x\ e_1$	$14\ e_1\ e_0$	
2	$e_0\ \gamma\ \lambda_1^x$	$14\ e_0$	
4	$e_0\ \gamma$	${}^0\lambda_1^x\ 14\ e_0$	
1	$e_0\ e_2\ \gamma\ \gamma\ +\ 1\ \gamma\ \lambda_2^w\ x$	$e_2\ e_0$	$e_2=[14/x]e_0$
2	$e_0\ e_2\ \gamma\ \gamma\ +\ 1\ \gamma\ \lambda_2^w$	$14\ e_2\ e_0$	
4	$e_0\ e_2\ \gamma\ \gamma\ +\ 1\ \gamma$	${}^2\lambda_2^w\ 14\ e_2\ e_0$	
1,1,3	$e_0\ e_2\ \gamma\ \gamma\ +\ 1\ e_3\ \gamma$ neg w	$e_3\ e_2\ e_0$	$e_3=[14/w]e_2$
5	$e_0\ e_2\ \gamma\ \gamma\ +\ 1\ e_3$	$-14\ e_3\ e_2\ e_0$	
1,1,3,3	$e_0\ e_2\ \gamma\ \gamma\ +\ 1$	$-14\ e_2\ e_0$	
5	$e_0\ e_2$	$-13\ e_2\ e_0$	
5	e_0	$-13\ e_0$	
		-13	

Environment Structure:



Optimizations for the CSE Machine.

The above five rules for operating the CSE Machine are sufficient for carrying out the evaluation of any applicative expression, but they are rather clumsy for unary and binary operators, for conditionals, for tuples, and for n-ary functions. Here are additional rules for operating the CSE Machine more efficiently.

CSE Rules 6 and 7: Unary and Binary Operators.

In the control structures, abbreviate:

$\gamma\ \gamma$ +	to	+
$\gamma\ \gamma$ -	to	-
	. . .	(other binary operators)
γ neg	to	neg
γ not	to	not
	. . .	(other unary operators)

	CONTROL	STACK	ENV
CSE Rule 6 (binop)	.... binop	Rand Rand	
	....	Result	where Result=Apply[binop,Rand,Rand]
CSE Rule 7 (unop)	.... unop	Rand	
	....	Result	where Result=Apply[unop,Rand]

CSE Rule 8: Conditional.

For an expression of the form "B $\rightarrow$ E_1 | E_2", generate the following control structures:

$$\delta_{\text{then}}\ \delta_{\text{else}}\ \beta\ B$$
$$\delta_{\text{then}} = \text{ control structure for } E_1$$
$$\delta_{\text{else}} = \text{ control structure for } E_2$$

Here, "B" will be evaluated first, leaving (we hope) a truthvalue on the top of the stack. Then, β is an operator that pops the truthvalue from the stack, and based on it, keeps one of the δ's and discards the other. **Note**: Here, β and δ have nothing to do with the so-named reductions for applicative expressions. Also note that with these control structures, it is no longer necessary to standardize the "$\rightarrow$" node in the AST. Primitive operator "Cond" is no longer necessary.

	CONTROL	STACK	ENV
CSE Rule 8 (Conditional)	.... $\delta_{\text{then}}\ \delta_{\text{else}}\ \beta$	true	
	.... δ_{then}	....	
	.... $\delta_{\text{then}}\ \delta_{\text{else}}\ \beta$	false	
	.... δ_{else}	....	

Example

Applicative expression	Control Structures
(λn.n<0 $\rightarrow$ -n | n) (-3)	$\delta_0 = \gamma\ \lambda_1^{n}$ neg 3
	$\delta_1 = \delta_2\ \delta_3\ \beta$ < n 0
	δ_2 = neg n
	δ_3 = n

RULE	CONTROL	STACK	ENV
1	$e_0\ \gamma\ \lambda_1^n$ neg 3	e_0	e_0=PE
7	$e_0\ \gamma\ \lambda_1^n$ neg	3 e_0	
2	$e_0\ \gamma\ \lambda_1^n$	-3 e_0	
4	$e_0\ \gamma$	${}^0\lambda_1^n$ -3 e_0	
1,1	$e_0\ e_1\ \delta_2\ \delta_3\ \beta$ < n 0	$e_1\ e_0$	e_1=[-3/n]e_0
6	$e_0\ e_1\ \delta_2\ \delta_3\ \beta$ <	-3 0 $e_1\ e_0$	
8	$e_0\ e_1\ \delta_2\ \delta_3\ \beta$	true $e_1\ e_0$	
1,7	$e_0\ e_1$ neg n	$e_1\ e_0$	
5,5	$e_0\ e_1$	3 $e_1\ e_0$	
		3	

CSE Rules 9 and 10: Tuples.

For a tuple of the form (E_1, E_2, ..., E_n), generate the control structure "τ_n E_1 ... E_n". Here τ_n pops the n top-most values from the stack, makes a tuple with them, and pushes the tuple on the stack. Note that tuple elements are evaluated **right-to-left** this way.

	CONTROL	STACK	ENV
CSE Rule 9 (tuple formation)	.... τ_n	V_1 ... V_n	
	....	(V_1,...,V_n)	
CSE Rule 10 (tuple selection)	.... γ	(V_1,...,V_n) I	
	....	V_I	

CSE Rule 11: n-ary functions.

For an expression of the form λ(x,y).E, simply allow each environment to bind more than one variable to a value. It will no longer be necessary to standardize subtrees in which there is a "," node to the left of a λ node.

	CONTROL	STACK	ENV
CSE Rule 11 (n-ary function)	.... γ	${}^c\lambda_k^{V_1,...,V_n}$ (val_1, . . . , val_n)	e_m=[val_1/V_1]...[val_n/V_n]e_c
	.... e_m δ_k	e_m	

Example:

Applicative expression	Control Structures
(λ(x,y).x+y) (5,6)	$\delta_0 = \gamma\ \lambda_1^{x,y}\ \tau_2$ 5 6
	δ_1 = + x y

RULE	CONTROL	STACK	ENV
1,1	$e_0\ \gamma\ \lambda_1^{x,y}\ \tau_2$ 5 6	e_0	e_0=PE
9	$e_0\ \gamma\ \lambda_1^{x,y}\ \tau_2$	5 6 e_0	
2	$e_0\ \gamma\ \lambda_1^{x,y}$	(5,6) e_0	
11	$e_0\ \gamma$	${}^0\lambda_1^{x,y}$ (5,6) e_0	
1,1	$e_0\ e_1$ + x y	$e_1\ e_0$	e_1=[5/x][6/y]e_0
6	$e_0\ e_1$ +	5 6 $e_1\ e_0$	
5,5	$e_0\ e_1$	11 $e_1\ e_0$	
		11	