EXECUTION MONITOR AS PROGRAMMING LANGUAGE PRIMITIVE

Access Control Lists

```
type acl = Empty
                | AC of string * acl;;
(* Access control lists *)

let rec emCheck alist op =
  match alist with
    | Empty      -> false
    | AC(aop, als)-> if op = aop then true else emCheck als op;;
```

Interpreter: OCAML code (eval)

$$\frac{env(x) = v}{env \ \triangleright \ Den\ x \ \Rightarrow v}$$

$$\frac{env \triangleright \ e_1 \Rightarrow v_1 \, , env \triangleright \ e_2 \Rightarrow v_2}{env \triangleright \ Sum(e_1, e_2) \Rightarrow v_1 + v_2}$$

```
eval Den x, env ->lookup x env
```

```
eval Sum(e1, e2) env ->
                        eval e1 env + eval e2 env
```

Wrapped Interpreter: eval env acl

$$\frac{env(x) = v}{env\ acl\ \rhd Den\ x \Rightarrow v}$$

$$\frac{add \in al\ env \rhd\ e_1 \Rightarrow v_1\ , env \rhd\ e_2 \Rightarrow v_2}{env\ acl\ \rhd\ Sum(e_1, e_2) \Rightarrow v_1 + v_2}$$

```
eval Den x, env, al -> lookup x env
```

```
eval Sum(e1, e2) env al ->
    if (emCheck alist "add")
       then (eval e1 env al) + (eval e2 env al)
       else failwith("Sum not allowed")
```

The OCAML code

```
let rec eval exp env (alist:acl) = match exp with
  | Eint(n) -> n
  | Den x -> lookup env x
  | Let(i,e1,e2) -> let v1 = eval e1 env acl in eval e2 (bind env i v1) acl
  | Sum(e1,e2) ->  if (emCheck alist "add")
       then (eval e1 env alist) + (eval e2 env alist)
       else failwith("Sum not allowed")
  | Times(e1,e2) -> if (emCheck alist "prod")
      then  (eval e1 env alist) * (eval e2 env alist)
      else failwith("times not allowed")
  | Minus(e1,e2) -> if (emCheck alist  "sub")
      then (eval e1 env alist) - (eval e2 env alist)
      else failwith("Minus not allowed");;
```

ACL as local Policies within Execution Monitor. #take 1

letEM x = e1 with al in e2

Add a local access policy for the evaluation of e1

FUN+EM

OCAML type for arithmetic expressions with variables and execution monitor

```
type ide = string
type iexp =
  | Eint of int
  | Let of string *iexp * iexp
  | Den of ide
  | Sum of iexp * iexp
  | Times of iexp * iexp
  | Minus of iexp * iexp
  | LetEM of string * iexp * acl * iexp;;
```

Extending access control lists

```
let rec extend al1 al2 =
   match al2 with
     | Empty -> al1
     | AC(aop, als) -> AC(aop, (extend al1 als));;
```

From Operational semantics to OCAML Code

$$\frac{env\ acl[al] \triangleright e1 \Rightarrow v1 \quad env[x = v1], acl \triangleright e2 \Rightarrow v}{env, acl \triangleright LetEM\ x = e1\ with\ al\ in\ e2 \Rightarrow v}$$

```
LetEM(i,e1,al,e2) ->
   let newal = (extend alist al) in
        let v = (eval e1 env newal) in
                (eval e2 (bind env i v) alist)
```

The OCAML code

```
let rec eval iexp env (alist:acl) = match iexp with
 |  Eint(n) -> n
 | Den x -> lookup env x
 | Sum(e1,e2) ->  if (emCheck alist "add")
     then (eval e1 env alist) + (eval e2 env alist)
     else failwith("Sum not allowed")
 | Times(e1,e2) -> if (emCheck alist "prod")
    then  (eval e1 env alist) * (eval e2 env alist)
    else failwith("times not allowed")
 | Minus(e1,e2) -> if (emCheck alist  "sub")
    then (eval e1 env alist) - (eval e2 env alist)
    else failwith("Minus not allowed")
 | LetEM(i,e1,al,e2) -> let newal = (extend alist al) in
                        let v = (eval e1 env newal)
                            in (eval e2 (bind env i v) alist)
```

Execution Monitor take 2

letEM x = e1 with al in e2

Add a local access policy for the evaluation of e2

The OCAML code

```
let rec eval iexp env (alist:acl) = match iexp with
 | Eint(n) -> n
 | Den x -> lookup env x
 | Sum(e1,e2) ->  if (emCheck alist "add")
     then (eval e1 env alist) + (eval e2 env alist)
     else failwith("Sum not allowed")
 | Times(e1,e2) -> if (emCheck alist "prod")
    then  (eval e1 env alist) * (eval e2 env alist)
    else failwith("times not allowed")
 | Minus(e1,e2) -> if (emCheck alist  "sub")
    then (eval e1 env alist) - (eval e2 env alist)
    else failwith("Minus not allowed")
 | LetEM(i,e1,al,e2) -> let v = (eval e1 env acl)  in
                                  let newal = (extend alist al) in
                                       in (eval e2 (bind env i v) newal)
```

Execution Monitor take 3

letEM x = e1 with al in e2

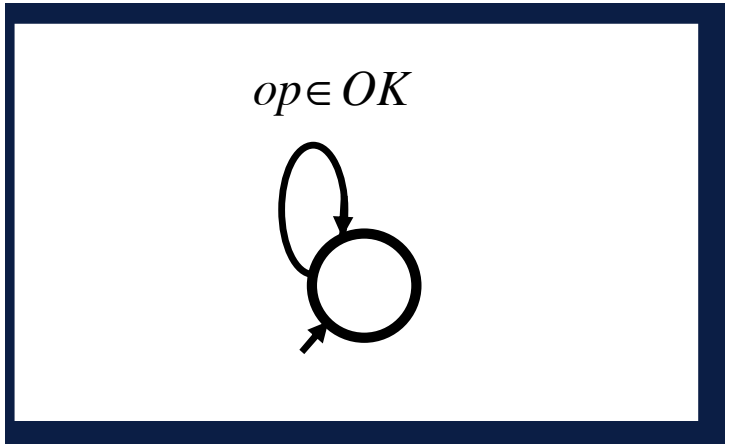
Add a local access policy for the evaluation of e 1 and e2

The OCAML code

```
let rec eval iexp env (alist:acl) = match iexp with
 | Eint(n) -> n
 | Den x -> lookup env x
 | Sum(e1,e2) ->  if (emCheck alist "add")
     then (eval e1 env alist) + (eval e2 env alist)
     else failwith("Sum not allowed")
 | Times(e1,e2) -> if (emCheck alist "prod")
    then  (eval e1 env alist) * (eval e2 env alist)
    else failwith("times not allowed")
 | Minus(e1,e2) -> if (emCheck alist  "sub")
    then (eval e1 env alist) - (eval e2 env alist)
    else failwith("Minus not allowed")
 | LetEM(i,e1,al,e2) -> let newal = (extend alist al) in
                                let v = (eval e1 env newal) in
                                    (eval e2 (bind env i v) newal)
```

Discussion

Simple Acces Control Policies are just Automata



… but EM

- Performs arbitrary computation to decide whether to allow event or halt
  - Can have side effects
  - Can change program flow

Exercize

1. Extend the simple language with execution monitor to include functions and function calls
2. Extende the language to include a variety of security policies rather than ACL.