# Translation from ML AST to JS AST

```
c ::=                   l ::=
    unit                    null
    bool                    bool
    int                     number
    float                   number
    char
    string                  string
    bytes

p ::=
    wild
    const               [C] = function(x){
                              if(x == C)
                                  return {valid:true, x:y}
                              else
                                  return {valid:false}}
    var                 [x]  = function(y){
                              return {valid:true}}
    CTor                [C p] = function(e){
                              if(e.tag == "C")
                                  return ([p])(e.value)
                              else
                                  return {valid:false}}
                         [p] = function(y){
                              return {valid:bool, x_1:.., x_2:.., ...}}
    branch
    record
    tuple               [(p_1,.., p_n)] = function(e){
                              if(e.tag == "Tuple" && e.arity == n)
                                  let r_1 = ([p_1])(e.f_1)
                                    ...
                                  let r_n = ([p_n])(e.f_n)
                                  if(r_1.valid && .. &&r_n.valid)
                                      return  {valid:true,
                                               x_1 = p_1.x_1,
                                               ..,
                                               x_n = p_n.x_n}
                              else
                                  return {valid:false}}
```

```
[e_ml] x_js s_js   //x_js -- where we save result of ml-expression translation
                   //s_js -- what we should do next
e ::=
    const                  [C] x s = const x = C; s
    var                    [x] y s = var y = x; s
    name
    let                    [let x = e_1 in e_2] y s = [e_1] x ([e_2] y s)
    app                    [f x] y s = var y = f [x]; s
    fun                    [fun x => e] f s = var f = function(x){
                                                  [e] r (return r)}
    match                  [match e with |p_i -> e_i] x s =
                                   [e] r (
                                       let r_1 = [p_1](r)
                                       if (r_1.valid){
                                           let fv(p_1) = r_1.fv(p_1)
                                           [e_1] x None
                                       } else ... ); s
    coerce
    CTor
    Seq
    Tuple
    Record
    Proj
    If                     [if e with e_1 else e_2] x s =
                                   [e] t (if(t){[e_1] x None} else {[e_2] x None}); s
    Raise
    Try
```

```
[match e with ]  =  let e = [e]
 | p_1 -> e_1       let p_1 = ([p_1])(e)
 | p_2 -> e_2       if (p_1.valid){
                        let x = p_1.x
                        return [e_1]
                    } else {
                        let p_2 = ([p_2])(e)
                        if (p_2.valid){
                            let y = p_2.y
                            let z = p_2.z
                            return [e_2]
                        } else throw NoMatch

[match e with]   =      switch ([e].tag)
 | C_1 x -> e_1         case "C_1": ...
   ...
 | C_n x -> e_n         case "C_n": ...
```

```
[(e_0, ..., e_(n-1))] =
    {_tag: "Tuple"
     _arity: n
     _f0: [e_0]
     _f1: [e_1]
     ...}

[C p] = {_tag: "C"
         _value: [p]}
```

# Types

```
t ::=
    int                          [int] = number
    bool                         [bool] = bool
    string                       [string] = string
    t_1*t_2*..*t_n               [(t_1*t_2*..*t_n)]  = { _tag: "Tuple",
                                                         _arity: 7,
                                                         _1: [t_1],
                                                         _2: [t_2],
                                                         ...}
    C t_1 ... t_n                [C t_1 ... t_n] = C <[t_1], ..., [t_n]>

C ::=
    type C x_1 ... x_n           [..] = type C <x_1 ... x_n> = { _tag: "Record",
     = {f_1:t_1, ..., f_n:t_n}                                   _f1: [t_1],
                                                                 ...
                                                                 _fn: [t_n]}
    type C x_1 ... x_n  = t      [..] = type C <x_1 ... x_n> = [t]
    type C x_1 ... x_n  =        [..] = type C_1 = {_tag: "C_1", _value: [t_1]}
        | C_1 of t_1                    ...
          ...                           type C_n = {_tag: "C_n", _value: [t_n]}
        | C_n of t_n                    type C = C_1 | C_2 | .. | C_n
```