Translation from ML AST to JS AST

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

[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] "_x" ([p_1] "_x" ([e_1] x None)
                                                ([p_2] "_x" ([e_2] x None)
                                                    ...
                                                    ([p_n] "_x" ([e_n] x None) Exp))); 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



[(e_1, ..., e_n)] =             [C p] = {_tag: "C"
    {_tag: "Tuple"                       _value: [p]}
     _arity: n
     _f1: [e_1]
     _f2: [e_2]
     ...}
```

```
[p] e_js s1_then s2_else
p ::=
    wild
    const                  [C] e_js s1 s2 = if (e_js == C) s1 else s2
    var                    [x] e_js s1 s2  = let x = e; s1
    CTor                   [C p] e_js s1 s2 = if(e.tag == "C")
                                                [p] e_js._value s1 s2
                                              else s2
                           //w/o repeating s2:
                           [C p] e_js s1 s2 = { let _valid = true
                                                if(e.tag == "C")
                                                    [p] e_js._value s1 (_valid = false)
                                                else  _valid = false
                                                if (!_valid) s2 }
    branch
    record
    tuple                  [(p_1,.., p_n)] e_js s1 s2 =
                                [p_1] e_js._f1
                                    ([p_2] e_js._f2
                                      (...([p_n] e_js._fn s1 s2)) s2) s2
                           //w/o repeating s2:
                           [(p_1,.., p_n)] e_js s1 s2 =
                               { let _valid = true
                                 [p_1] e_js._f1
                                       ([p_2] e_js._f2
                                         (...([p_n] e_js._fn s1 (_valid = false)))
                                           (_valid = false)) (_valid = false)
                                 if (!_valid) s2 }
[p when e] e_js s1 s2 = [p] e_js ([e] "_x" (if (_x) s1 else s2)) s2
//w/o repeating s2:
[p when e] e_js s1 s2 = { let _valid = true
                          [p] e_js ([e] "_x" (if (_x) s1 else (_valid = false)))
                                   (_valid = false)
                          if (!_valid) s2 }
```

Cases, in which we can avoid repeat "_valid = false", i.e.

```
[p] e s1 s2 = if (be_1[e] && be_2[e] && ..)
              {let fv_1 = ..
               let fv_2 = ..
               s1} else s2
```

```
sp ::=                          [sp] e s1 s2 = if (be) {s; s1}
    var x                                      else s2
    const C                     [sp] e ==> be, s
    CTop C x
    Tuple (x, y)

[sp when e] l s1 s2 = { let _valid = true
                        if (be[l]){s; [e] "_valid" (if (_valid) s1)}
                        else _valid = false
                        if (!_valid) s2 }
[x] e ==> true, let x = e
[C] e ==> (C == e), None
[C sp] e ==> (e.tag == "C" && be), s
             where [sp] e.value ==> be, s
[(sp_1, ..., sp_n)] e ==> (be_1 && .. && be_n); s_1; ...; s_n
                          [sp_1] e._f1 ==> be_1, s_1
                          ...
                          [sp_n] e._fn ==> be_n, s_n
```

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
```