Review

Wednesday, March 25, 2020 12:20 AM

$$[\![pe]\!]\,[p, s] = p_s$$

$$[\![int]\!]\,[q, i] = o$$

$$[\![pe]\!]\,[int, q] = int_q$$ compiling

$$[\![pe]\!]\,[pe, int] = \boxed{pe_{int}}$$ compiler

$$[\![pe]\!]\,[pe, pe] = pe_{pe}$$ compiler-compiler

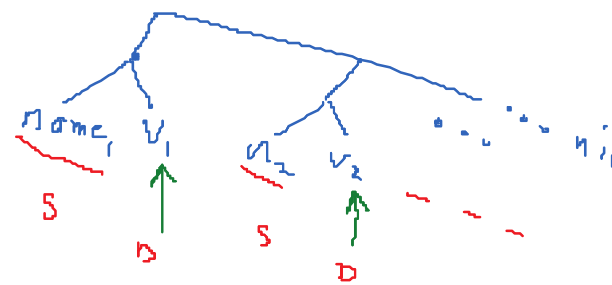
<u>Removal of interpretation overhead</u>

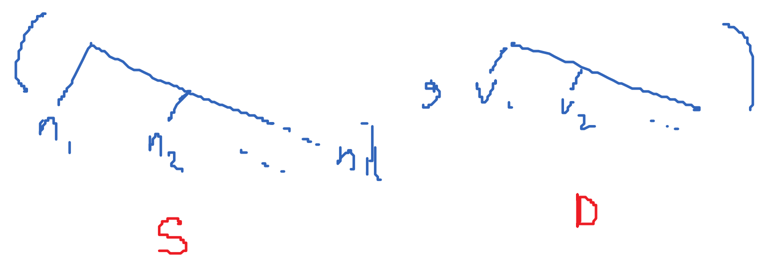
in-line substitution of functions called by int

elimination of cases not needed to interpret q

stores of q -- part of the information is known

e.g., binding list





Reparenthesization Principle

Wednesday, March 25, 2020 12:21 AM

Goal: Ensure that there is a concordance between the residual-program states and the subject-program states

Subject-program states	Residual-program states
(label, (vs, vd))	((label, vs), vd)
values of supplied variables / values of delayed variables	"compound label" (conceptually)

Theme: "Conversion of data to control"
("control" = position in the code)

⋮
l1: ...
⋮

⋮
(l1,[6,97,13]): ...
⋮
(l1,[3,99,17]): ...
⋮

pc == l1 + the value of vs ([3,99,17] or [6,97,13])

pc == (l1, [3,99,17])
or
pc == (l1, [6,97,13])

Concordance:

Subject-program trace	Residual-program trace
⋮	⋮
(l1, ([3,99,17], [2,6]))	((l1, [3,99,17]), [2,6])
⋮	⋮
(l59, ([12,99,15], [2,27]))	((l59, [12,99,15]), [2,27])
⋮	⋮
(l1, ([6,97,13], [2,25]))	((l1, [6,97,13]), [2,25])
⋮	⋮
(l1, ([3,99,17], [15,3]))	((l1, [3,99,17]), [15,3])
⋮	⋮

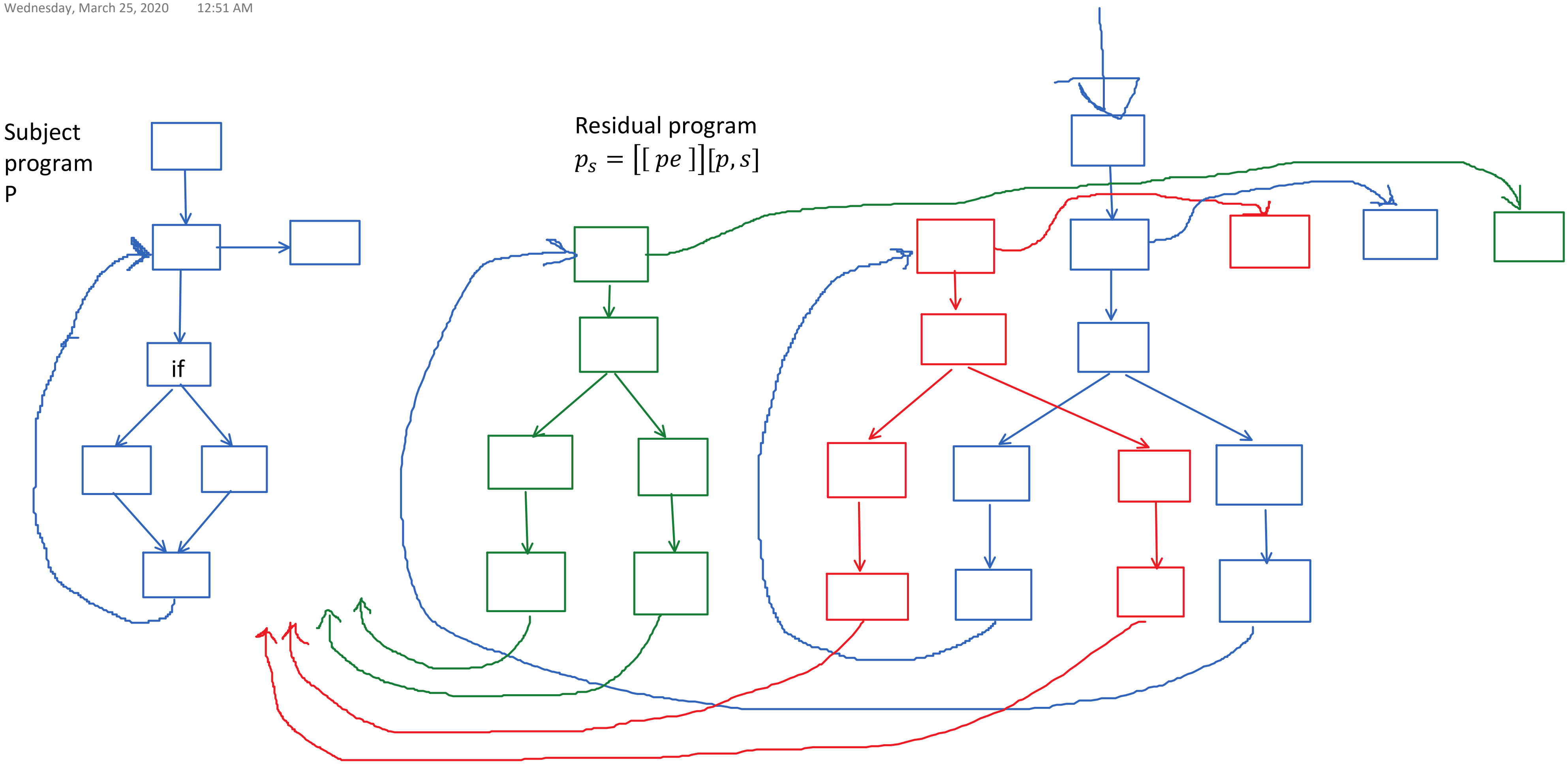
The trace visits label l1 three times

The trace visits label (l1, [6,97,13]) once

The trace visits label (l1, [3,99,17]) twice

What does a specialized program look like?

Wednesday, March 25, 2020 12:51 AM



Languages, languages, languages, . . .

Wednesday, March 25, 2020 1:06 AM

1. "surface syntax" of the language of subject programs

   L = A simple imperative flow-chart language with int and list data

Constructs of L
assignment
if cond goto label else label'
goto label
read of initial data
return final value
print

Also, as syntactic sugar
begin … end
while … do … od
repeat … until …

Data types	**Operators**
integers	plus, <, >, =, …
s-expressions	hd, tl, cons, nil, isnil

2. "deep syntax" of the language of subject programs

   s-expression representation of a program's control-flow graph (CFG) + algebraic data type to represent expressions

Algebraic datatype for representing L expressions

```
exp ::= ConstExpr(constant)
      |  IdentExpr(identifier)
      | Compound(op exp exp)
constant ::= <integer constants>
identifier ::= [a-zA-Z]+
operator ::= + | * | cons | hd | tl | …
```

3. meta-language in which to describe the partial-evaluation algorithm

   o pidgin Algol
   o tables of cases
   o informal graph diagrams
   o <hand-waving> + <smoke & mirrors>

meta-language permits deconstructing expressions via pattern matching:

```
cases e of
   ConstExpr(c): … expression involving e, c, …
   IdentExpr(i): … expression involving e, i, …
   Compound(o, a, b): … expression involving
                              e, o, a, b ….
end
```

Simplification (not the whole story of partial evaluation!!)

Wednesday, March 25, 2020 1:19 AM

```
simplify(e, store) =              // store is a map from names to values
cases e of
    ConstExpr(c): e
    IdentExpr(i): DefinedIn(i,store) ? ConstExpr(Lookup(i,store)) : e
    Compound(op, a, b):
        let v1 = simplify(a, store) and v2 = simplify(b, store) in
            cases v1 of
              ConstExpr(c1):
                cases v2 of
                    ConstExpr(c2): ConstExpr(funcof(op)(c1,c2))
                    default: Compound(op,v1,v2)
              default: Compound(op,v1,v2)
```

Example of an L-program (surface and deep syntax)

Wednesday, March 25, 2020 1:28 AM

Surface syntax:

```
        read(N)

begin: i := 1
       sum := 0
       goto loop

loop: if i > N goto end else body

body:  sum := sum + i
       i := i + 1
       goto loop

end:  return sum
```

Deep syntax:

```
( ( (Read N) )

  ( (Block begin
        (Assign i ConstExpr(1))
        (Assign sum ConstExpr(0))
        (goto loop)
    )
    (Block loop
        (Cond (Less i N)
            end
            body
        )
    )
    (Block body
        (Assign sum Compound(+, IdentExpr(sum), IdentExpr(i)))
        (Assign i Compound(+, IdentExpr(i), ConstExpr(1)))
    )
    (Block end
        (Return sum)
    )
  )
)
```

(Singleton) list of read statements

List of blocks

For partial evaluation, need >= 2 read statements

Wednesday, March 25, 2020 1:38 AM

```
        read(y)
        read(z)

begin:  goto q

q:      if y < 3 goto r else s

r:      y := y+1
        z := z+1
        goto q

s       return z
```

A trace:

Suppose that the input is y: 1, z: c, where c is some specific value

(begin, (1,c))
(q, (1, c))
(r, (1, c))
(q, (2, c+1))
(r, (2, c+1))
(q, (3, c+2))
(s, (3, c+2))

Reparenthesization

Wednesday, March 25, 2020 1:44 AM

state = (pp, (values of supplied vars, values of delayed vars))
$\simeq$ ((pp, values of supplies vars), value of delayed vars)

```
                                   (pp, vs1): -----------
                                              -----------
                                              -----------

pp: -----------
    -----------
    -----------                    (pp, vs2): -----------          "Polyvariant specialization"
    ------------                              -----------          = 1 to many
    ------------                              -----------

                                   (pp, vs3): -----------
                                              -----------
                                              -----------
```

Example: Specialize program w.r.t. y↦1

Wednesday, March 25, 2020 1:49 AM

```
        read(y)
        read(z)

begin:  goto q

q:      if y < 3 goto r else s

r:      y := y+1
        z := z+1
        goto q

s       return z
```

```
             read(z)
(begin, 1):  goto (q,1)
(q,1):       goto (r,1)
(r,1):       z := z+1
             goto (q,2)
(q,2):       goto (r,2)
(r,2):       z := z+1
             goto (q,3)
(q,3):       goto (s,3)
(s,3):       return z
```

Current partial state: y↦3

Worklist: { … }

(q,5): …

(s,5): …

(r,5): …

(q,6): …

(r,6): …

.

.

Trace (w.r.t. z↦c)

((begin,1), c)
((q,1), c)
((r,1), c)
((q,2), c+1)
((r,2), c+1)
((q,3), c+2)
((s,3), c+2)

Current state: z↦c+2

Trace of the original program

(begin, (1,c))
(q, (1, c))
(r, (1, c))
(q, (2, c+1))
(r, (2, c+1))
(q, (3, c+2))
(s, (3, c+2))

Transition compression

Wednesday, March 25, 2020 1:53 AM

Lots of gotos to gotos
Most correspond to actions in the original program on supplied quantities "swallowed by the partial evaluator" (i.e., performed at PE-time -- in particular, "y := y+1")

Compress the goto transitions

```
         read(z)
(r,1):   z := z+1
(r,2):   z := z+1
(s,3):  return z
```

Trace (w.r.t. z↦c)

((r,1), c)
((r,2), c+1)
((s,3), c+2)

Not as easy to make the correspondence with the trace of the original program

Two-phase partial evaluation

Wednesday, March 25, 2020 12:00 PM

1: Binding-time analysis (BTA)
2: Specialization

BTA:
division: labeling of variables/statements into S and D

uniform: each variable has the same S/D classification at all program points (think: "type")

```
read(z)    // D

...
z := 27    // S, but makes the division non-uniform
```

congruence: Variables classified S, can only depend on variables classified S.
- congruence analysis ~ taint analysis
- D leads to D

```
y := y+1    //   S <- S
z := z+1    //   D <- D

w := y+z    //   D <-  S+D
```