Text Processing in APL

Dyalog ‘22

Aaron Hsu - aaron@dyalog.com

Is APL only about numbers?

Text

IN → Trees → Out

$\mathbb{N} \to$ Trees

□S □R

Limited

Sharp Corners

Comp. Sci.?

Grammars

Context-free

Context-sensitive

PEG

Parsing Expression Grammar

```
Seq      ← S1 S2
Choice   ← A | B
```

Recursive Descent

```
S   ← ϵ | (char | Par | Brk) S
Par ← ‘(‘ S ‘)’
Brk ← ‘[‘ S ‘]’
```

Usability?

Errors
AST Creation
Auxiliary Data

Tracking Flow

```
S   ← ϵ | (char | Par | Brk) S
Par ← ‘(‘ S ‘)’
Brk ← ‘[‘ S ‘]’
```

```
S   ← (char | Par | Brk) S | ϵ
Par ← ‘(‘ S ‘)’
Brk ← ‘[‘ S ‘]’
```

```
S   ← (Par | Brk | char) S | ϵ
Par ← ‘(‘ S ‘)’
Brk ← ‘[‘ S ‘]’
```

Performance?

Easy to Explode, Hard to Catch

Interpreter Overhead

```
      old←{OP.ps ⎕SRC t0009}
      new←{codfns.PS ⎕SRC t0009}
      cmpx 'newθ' 'oldθ'
  newθ → 2.4E¯2 |     0% ⎕
* oldθ → 1.5E0  | +6151% ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
```

Sharp Corners, still.

Data-parallel
Idiomatic
Flexible/Scalable

Error Handling
Context Sensitivity

Avoids sharp corners

Linear Data-flow
“Micro pass”

Linearize the Grammar Dependencies

```
S   ← (Par | Brk | char) S | ϵ
Par ← ‘(‘ S ‘)’
Brk ← ‘[‘ S ‘]’
```

```
      x←'kdfl(kkdf(ksdk[ksd(ksfl]ksk)ksd))'
      d←+⍀(o←x∊'([')+-c←x∊')]'
      2{p[ω]←α[α⍸ω]}⌿⊢∘⊂⌸¯1⌽d⊣p←⍳≢d
      ⍉x,⍪x[p]
kdfl(kkdf(ksdk[ksd(ksfl]ksk)ksd))
kdfl((((((((((([[[[(((((([[[[((((((
      ⊃'()'∧.=c∘⌿¨x[p]x
0 0 1 1
      codfns.(dwv pp3) p
 o  o  o  o  o
             │
             ├─┬─┬─┬─┬───────────────────────────────────────┐
             │ │ │ │ │                                       │
             o o o o o                                       o
                     │
                     ├─┬─┬─┬─┬───────────────────────┬─┬─┬─┐
                     │ │ │ │ │                       │ │ │ │
                     o o o o o                       o o o o
                             │
                             ├─┬─┬─┬─────────┬─┬─┬─┐
                             │ │ │ │         │ │ │ │
                             o o o o         o o o o
                                   │
                                   ├─┬─┬─┬─┐
                                   │ │ │ │ │
                                   o o o o o
```

```
PEG'Mop     ← Pmop , Afx                                        : 20∘⌽         '
PEG'Pdop1   ← dop1                                              : 3P           '
PEG'Dop1    ← Pdop1 , Afx                                       : 80∘⌽         '
PEG'Pdop2   ← dop2                                              : 3P           '
PEG'Vop     ← Atom , Pdop2 , Afx                                : 50∘⌽         '
PEG'Pdop3   ← dop3                                              : 3P           '
PEG'Dop3    ← Pdop3 , Atom                                      : 70∘⌽         '
PEG'Bop     ← rbrk , Ex , lbrk , (4 Lbrk) , Afx                 : 50∘⌽         '
PEG'JotDP   ← dot , jot                                         : 3P∘⌽         '
PEG'JotDot  ← Fnp , JotDP                                       : 20           '
PEG'Fop     ← Fnp , (Dop1 | Dop3 ?)                             : MkAST        '
PEG'Afx     ← Mop | JotDot | Fop | Vop | Bop                                   '
PEG'Trn     ← Afx , (Afx | Idx | Atom , (∇ ?) ?)                : 5F∘⌽         '
PEG'Bind    ← gets , Symbol [αα]                                : αα B         '
PEG'Gets    ← ϵ                                                 : ⊣αα P{,''←''}'
PEG'Mname   ← Afx , (1 Name)                                    : 4E Atn∘⌽     '
PEG'Ogets   ← Afx , (3 Gets)                                    : 20           '
PEG'Mbrk    ← Ogets , Brk , (1 Name)                            : 4E∘(1∘↓)Atn∘⌽'
PEG'Mget    ← Mname | Mbrk                                                     '
PEG'Bget    ← 2 Gets , Brk , (1 Name)                           : 4E∘(1∘↓)Atn∘⌽'
PEG'ExHd    ← Asgn | (1 Bind) | App , ∇ ?                                      '
PEG'Ex      ← IAx , ExHd                                        : MkAST        '
```

```
Fn←{a(i d)←⍵ ⋄ 0=≢a:0 θ α(i d)
 0=≢ss←(4⊃z)⌿⍨m←(((N∆⍳'F')=1⊃⊢)∧¯1=2⊃⊢)⊢z←⍪⌿↑a:0(,⊂z)α(i d)
 0<c←r⊃⍨0,pi←⊃⍒⊃r←↓⍉↑ps←α∘Fa¨ss,¨⊂⊂d:pi⊃ps
 0(,⊂(⊂¨¨z)((⊃⍪⌿)⊣@{m})¨⍨↓(m⌿0⊃z)+@0⍉↑⊃¨1⊃r)α(i d)}
FnType←{⌈⌿2,3 4×¯1≠(¯1,⍨1⊃⍵)['αα' 'ωω'⍳⍨⊃⍵]}
PEG'ClrEnv ← (Alp[¯1]),(Alp,Alp[¯1]),(Omg[¯1]),(Omg,Omg[¯1])↓                 '
PEG'Fax     ← lbrc , (Gex | Ex | Fex Stmts rbrc) → Fn               : (FnType α)F  '
PEG'FaFnW   ← Omg[1]↓ , Fax []                                                 '
PEG'FaFnA   ← Omg[1] , (Alp[1])↓ , Fax []                                      '
PEG'FaFn    ← FaFnW | FaFnA                                                    '
PEG'FaMopV  ← Alp,Alp[1]↓ , FaFn []                                            '
PEG'FaMopF  ← Alp,Alp[2]↓ , FaFn []                                            '
PEG'FaMop   ← FaMopV , (FaMopF ?) | FaMopF                                     '
PEG'FaDopV  ← Omg,Omg[1]↓ , FaMop []                                           '
PEG'FaDopF  ← Omg,Omg[2]↓ , FaMop []                                           '
PEG'FaDop   ← FaDopV , (FaDopF ?) | FaDopF                                     '
PEG'Fa      ← ClrEnv , (FaFn | FaMop | FaDop) []                               '
PEG'Nlrp    ← sep | rbrc ↑ Slrp (lbrc Blrp rbrc)                               '
PEG'Stmt    ← sep | (αα , (sep | lbrc) ⌽ Nlrp)                                 '
PEG'Stmts   ← ωω | (αα Stmt , ∇)                                               '
PEG'Ns      ← nss , (Ex | Fex Stmts nse) , eot → Fn                  : (¯1+⊣)0F⊢  '
```

Compute parent vector from d
Compute the nameclass of dfns
Nest top-level root lines as Z nodes
Wrap all dfns expression bodies
Drop any Z nodes that are empty
Parse :Namespace
Parse guards
Parse brackets and parentheses
Parse ;
Mark system variables
Mark primitives
Parse niladic tokens

Unify atomic array values
Mark bindable nodes
Wrap bindings into B nodes
Wrap functions as closures
Link variables to their bindings
Infer types of bindings
Parse strands
Parse [] operator
Parse function expressions
Parse assignments
Parse expressions
Simplify and Optimize the AST

```
⍝ Link variables to their bindings
mk←{⍺[⍵],⍪n[⍵]}
_←{
    ⍝ Link local variables with their local bindings
    vb[i]←fb[fr⍳rf mk⊢i←⍸(t=V)∧vb=¯1]
    vb[i]←fb[fr⍳rfn mk⊢i←⍸(t=V)∧vb=¯1]
    b←vb[i←i⌿⍨vb[i]≠¯1]
    vb[i⌿⍨(rz[i]<rz[b])∨(rz[i]=rz[b])∧i≥b]←¯1

    ⍝ Mark free variables with their scope before binding
    lx[i←⍸(t=V)∧vb=¯1]←1

    ⍝ Add free variables to closures
    i←i⌿⍨k[rfn[i]]≠0 ⋄ ci←p[rfn[i]] ⋄ vb[i]←(≢p)+⍳≢i
    p,←ci ⋄ vb lx,←(≢ci)⍴¨¯1 0 ⋄ rf rfn(⊣,I)←⊂ci
    t k n pos end(⊣,I)←⊂i
i}⍣{0=≢⍺}⍬
```

```
⍝ Link variables to their bindings
mk←{⍺[⍵],⍪n[⍵]}
_←{
    ⍝ Link local variables with their local bindings
    vb[i]←fb[fr⍳rf mk⊢i←⍸(t=V)∧vb=¯1]
    vb[i]←fb[fr⍳rfn mk⊢i←⍸(t=V)∧vb=¯1]
    b←vb[i←i⌿⍨vb[i]≠¯1]
    vb[i⌿⍨(rz[i]<rz[b])∨(rz[i]=rz[b])∧i≥b]←¯1

    ⍝ Mark free variables with their scope before binding
    lx[i←⍸(t=V)∧vb=¯1]←1

    ⍝ Add free variables to closures
    i←i⌿⍨k[rfn[i]]≠0 ⋄ ci←p[rfn[i]] ⋄ vb[i]←(≢p)+⍳≢i
    p,←ci ⋄ vb lx,←(≢ci)⍴¨¯1 0 ⋄ rf rfn(⊣,I)←⊂ci
    t k n pos end(⊣,I)←⊂i
i}⍣{0=≢⍺}⍬
```

```
⍝ Link variables to their bindings
mk←{α[ω],⍪n[ω]}
_←{
    ⍝ Link local variables with their local bindings
    vb[i]←fb[fr⍳rf mk⊢i←⍸(t=V)∧vb=¯1]
    vb[i]←fb[fr⍳rfn mk⊢i←⍸(t=V)∧vb=¯1]
    b←vb[i←i⌿⍨vb[i]≠¯1]
    vb[i⌿⍨(rz[i]<rz[b])∨(rz[i]=rz[b])∧i≥b]←¯1

    ⍝ Mark free variables with their scope before binding
    lx[i←⍸(t=V)∧vb=¯1]←1

    ⍝ Add free variables to closures
    i←i⌿⍨k[rfn[i]]≠0 ⋄ ci←p[rfn[i]] ⋄ vb[i]←(≢p)+⍳≢i
    p,←ci ⋄ vb lx,←(≢ci)⍴¨¯1 0 ⋄ rf rfn(⊣,I)←⊂ci
    t k n pos end(⊣,I)←⊂i
i}⍣{0=≢α}⍬
```

```apl
⍝ Link variables to their bindings
mk←{⍺[⍵],⍪n[⍵]}
_←{
    ⍝ Link local variables with their local bindings
    vb[i]←fb[fr⍳rf mk⊢i←⍸(t=V)∧vb=¯1]
    vb[i]←fb[fr⍳rfn mk⊢i←⍸(t=V)∧vb=¯1]
    b←vb[i←i⌿⍨vb[i]≠¯1]
    vb[i⌿⍨(rz[i]<rz[b])∨(rz[i]=rz[b])∧i≥b]←¯1

    ⍝ Mark free variables with their scope before binding
    lx[i←⍸(t=V)∧vb=¯1]←1

    ⍝ Add free variables to closures
    i←i⌿⍨k[rfn[i]]≠0 ⋄ ci←p[rfn[i]] ⋄ vb[i]←(≢p)+⍳≢i
    p,←ci ⋄ vb lx,←(≢ci)⍴⍤¯1 0 ⋄ rf rfn(⊣,I)←⊂ci
    t k n pos end(⊣,I)←⊂i
i}⍣{0=≢⍺}⍬
```

```
⍝ Link variables to their bindings
mk←{⍺[⍵],⍪n[⍵]}
_←{
    ⍝ Link local variables with their local bindings
    vb[i]←fb[fr⍳rf mk⊢i←⍸(t=V)∧vb=¯1]
    vb[i]←fb[fr⍳rfn mk⊢i←⍸(t=V)∧vb=¯1]
    b←vb[i←i⌿⍨vb[i]≠¯1]
    vb[i⌿⍨(rz[i]<rz[b])∨(rz[i]=rz[b])∧i≥b]←¯1

    ⍝ Mark free variables with their scope before binding
    lx[i←⍸(t=V)∧vb=¯1]←1

    ⍝ Add free variables to closures
    i←i⌿⍨k[rfn[i]]≠0 ⋄ ci←p[rfn[i]] ⋄ vb[i]←(≢p)+⍳≢i
    p,←ci ⋄ vb lx,←(≢ci)⍴¨¯1 0 ⋄ rf rfn(⊣,I)←⊂ci
    t k n pos end(⊣,I)←⊂i
i}⍣{0=≢⍺}⍬
```

Compute parent vector from d
Compute the nameclass of dfns
Nest top-level root lines as Z nodes
Wrap all dfns expression bodies
Drop any Z nodes that are empty
Parse :Namespace
Parse guards
Parse brackets and parentheses
Parse ;
Mark system variables
Mark primitives
Parse niladic tokens
Unify atomic array values
Mark bindable nodes
Wrap bindings into B nodes
Wrap functions as closures
Link variables to their bindings
Infer types of bindings
Parse strands
Parse [] operator
Parse function expressions
Parse assignments
Parse expressions
Simplify and Optimize the AST

```
⍝ Parse plural value sequences to A7 nodes
i←|i⊣km←0<i←∊p[i](⊂-⍤⊣,⊢)⌸i←⍸t[p]=Z
msk∧←⊃1 ¯1∨.⌽⊂msk←km∧(t[i]=A)∨(t[i]∊P V Z)∧k[i]=1
np←(≢p)+⍳≢ai←i⌿⍨am←2>⌿msk⍪0 ⋄ p←(np@ai⍳≢p)[p] ⋄ p,←ai
t k n lx pos end(⊣,I)←⊂ai
t k n lx pos(⊣@ai⍨)←A 7(⊂'')0(pos[i⌿⍨km←2<⌿0⍪msk])
p[msk⌿i]←ai[¯1++⍀km⌿⍨msk←msk∧~am]
```

```
⍝ Parse plural value sequences to A7 nodes
i←|i⊣km←0<i←∊p[i](⊂-⍤⊣,⊢)⌸i←⍸t[p]=Z
msk∧←⊃1 ¯1∨.⌽⊂msk←km∧(t[i]=A)∨(t[i]∊P V Z)∧k[i]=1
np←(≢p)+⍳≢ai←i⌿⍨am←2>⌿msk⍪0 ⋄ p←(np@ai⍳≢p)[p] ⋄ p,←ai
t k n lx pos end(⊣,I)←⊂ai
t k n lx pos(⊣@ai⍨)←A 7(⊂'')0(pos[i⌿⍨km←2<⌿0⍪msk])
p[msk⌿i]←ai[¯1++⍀km⌿⍨msk←msk∧~am]
```

```apl
⍝ Parse plural value sequences to A7 nodes
i←|i⊣km←0<i←∊p[i](⊂-⍤⊣,⊢)⌸i←⍸t[p]=Z
msk∧←⊃1 ¯1∨.⌽⊂msk←km∧(t[i]=A)∨(t[i]∊P V Z)∧k[i]=1
np←(≢p)+⍳≢ai←i⌿⍨am←2>⌿msk⍪0 ⋄ p←(np@ai⍳≢p)[p] ⋄ p,←ai
t k n lx pos end(⊣,I)←⊂ai
t k n lx pos(⊣@ai⍨)←A 7(⊂'')0(pos[i⌿⍨km←2<⌿0⍪msk])
p[msk⌿i]←ai[¯1++⍀km⌿⍨msk←msk∧~am]
```

```apl
⍝ Parse plural value sequences to A7 nodes
i←|i⊣km←0<i←∊p[i](⊂-⍤⊣,⊢)⌸i←⍸t[p]=Z
msk∧←⊃1 ¯1∨.⌽⊂msk←km∧(t[i]=A)∨(t[i]∊P V Z)∧k[i]=1
np←(≢p)+⍳≢ai←i⌿⍨am←2>⌿msk⍪0 ⋄ p←(np@ai⍳≢p)[p] ⋄ p,←ai
t k n lx pos end(⊣,I)←⊂ai
t k n lx pos(⊣@ai⍨)←A 7(⊂'')0(pos[i⌿⍨km←2<⌿0⍪msk])
p[msk⌿i]←ai[¯1++⍀km⌿⍨msk←msk∧~am]
```

Flexible
Easy to grow

Avoids:
Cognitive context-switching
Domain segregation

Maps well to APL performance model

Fear not.

Thank you.