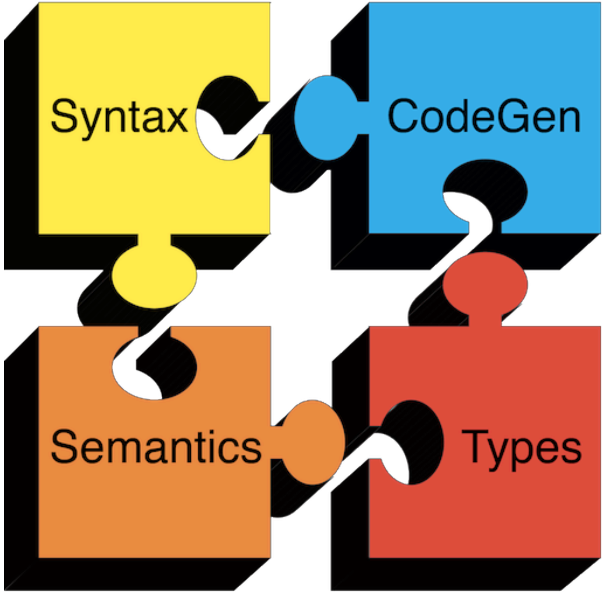

Compilers

Recursive Descent Algorithm: A Limitation

Alex
Aiken

RDA Limitation

E → T | T + E

T → int | int * T | (E)

```
bool term(TOKEN tok) { return *next++ == tok; }

bool E1() { return T(); }
bool E2() { return T() && term(PLUS) && E(); }

bool E()  {TOKEN *save = next; return     (next = save, E1())
                                        || (next = save,  E2());   }

bool T1() { return term(INT); }
bool T2() { return term(INT) && term(TIMES) && T(); }
bool T3() { return term(OPEN) && E() && term(CLOSE); }

bool T() { TOKEN *save = next; return    (next = save, T1())
                                        || (next = save,  T2())
                                        || (next = save,  T3()); }
```

RDA Limitation

- If a production for non-terminal X succeeds
  - Cannot backtrack to try a different production for X later

- General recursive-descent algorithms support such “full” backtracking
  - Can implement any grammar

RDA Limitations

- Presented recursive descent algorithm is not general
  - But is easy to implement by hand

- Sufficient for grammars where for any non-terminal at most one production can succeed

- The example grammar can be rewritten to work with the presented algorithm
  - ~~By *left factoring*~~, the topic of a future video