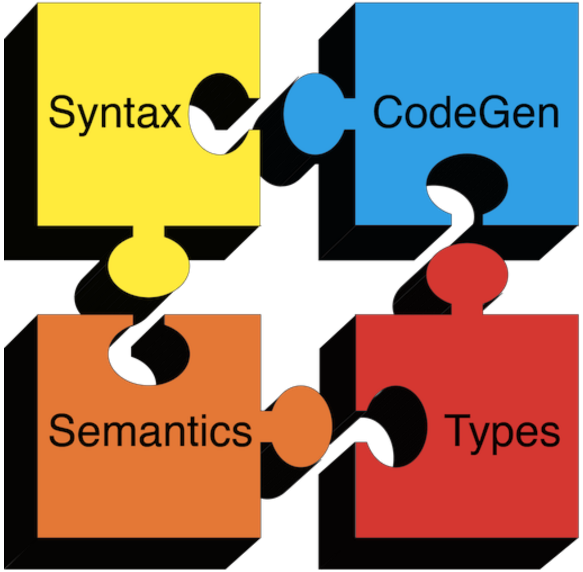


Compilers

Recognizing Handles

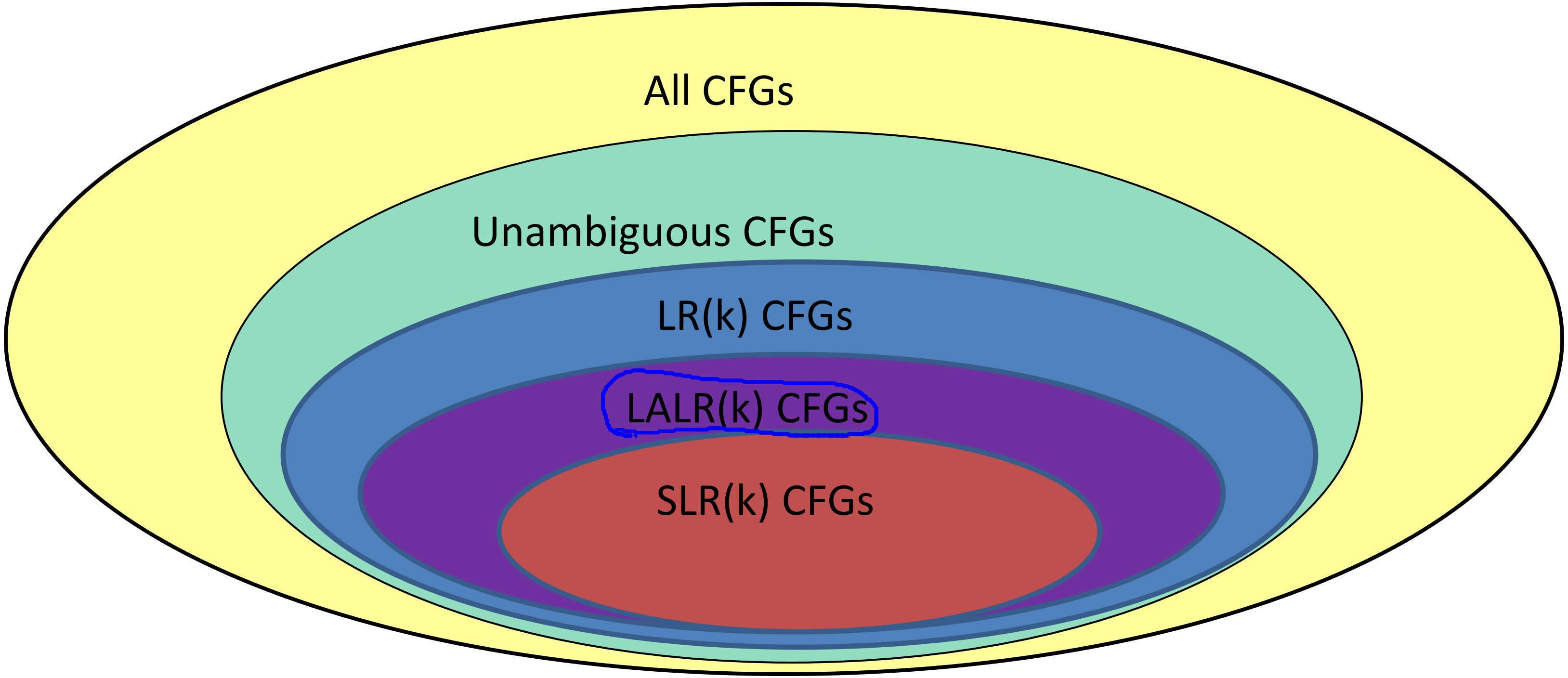
Alex Aiken

Recognizing Handles

- Bad News
  - There are no known efficient algorithms to recognize handles

- Good News
  - There are good heuristics for guessing handles
  - On some CFGs, the heuristics always guess correctly

Recognizing Handles



Recognizing Handles

- It is not obvious how to detect handles

- At each step the parser sees only the stack, not the entire input; start with that . . .

stack rest of input

α is a viable prefix if there is an ω such that $\alpha | \omega$ is a state of a shift-reduce parser

- What does this mean? A few things:
  - A viable prefix does not extend past the right end of the handle
  - It's a viable prefix because it is a prefix of the handle
  - As long as a parser has viable prefixes on the stack no parsing error has been detected

$\underbrace{\alpha | \omega}$

Important Fact #3 about bottom-up parsing:

For any grammar, the set of viable prefixes is a regular language

Recognizing Handles

- Important Fact #3 is non-obvious

- We show how to compute automata that accept viable prefixes

Recognizing Handles

- An item is a production with a "." somewhere on the rhs

- The items for $T \to (E)$ are

  $T \to .(E)$

  $T \to (.E)$

  $T \to (E.)$

  $T \to (E).$

Recognizing Handles

- The only item for $X \rightarrow \varepsilon$ is $X \rightarrow .$

- Items are often called "LR(0) items"

Recognizing Handles

- The problem in recognizing viable prefixes is that the stack has only bits and pieces of the rhs of productions
  - If it had a complete rhs, we could reduce

- These bits and pieces are always *prefixes* of rhs of productions

Recognizing Handles

Consider the input (int)

E → T + E | T
T → int * T | int | (E)

stack input

– Then (E |) is a state of a shift-reduce parse

– (E is a prefix of the rhs of T → (E)
  • Will be reduced after the next shift

– Item T → (E.) says that so far we have seen (E of this production and hope to see)

Recognizing Handles

- The stack may have many prefixes of rhs's

  $$\underbrace{\underline{\text{Prefix}_1}\ \underline{\text{Prefix}_2}\ \ldots\ \underline{\text{Prefix}_{n-1}}\underline{\text{Prefix}_n}}$$

- Let $\underline{\text{Prefix}_i}$ be a prefix of rhs of $\underbrace{X_i \rightarrow \alpha_i}$
  - $\underline{\text{Prefix}_i}$ will eventually reduce to $\underbrace{X_i}$
  - The missing part of α_{i-1} starts with X_i
  - i.e. there is a $\underbrace{X_{i-1} \rightarrow \underbrace{\text{Prefix}_{i-1}\ X_i\ \beta}}$ for some β

- Recursively, $\underbrace{\text{Prefix}_{k+1}\ldots\text{Prefix}_n}$ eventually reduces to the missing part of $\underbrace{\alpha_k}$

Recognizing Handles

stack

Consider the string (int * int):

(int * | int) is a state of a shift-reduce parse

input

- "(" is a prefix of the rhs of $T \to (E)$
- "ε" is a prefix of the rhs of $E \to T$
- "int *" is a prefix of the rhs of $T \to \text{int} * T$

Recognizing Handles

The "stack of items"

$T \to (.E)$

$E \to .T$

$T \to int * .T$

Says

We've seen "(" of $T \to (E)$

We've seen ε of $E \to T$

We've seen int * of $T \to int * T$

Recognizing Handles

Idea: To recognize viable prefixes, we must

- Recognize a sequence of partial rhs's of productions, where

- Each partial rhs can eventually reduce to part of the missing suffix of its predecessor