# review for Exam1

## Ch1.

- Language eval. criteria
  - readability
  - writability
  - reliability — more reliable → high cost
- Lang. categories
  - imperative
  - object-oriented
  - functional
  - logic (rule based)
  - concurrent
  - other categories
    - Scripting lang. — perl, javascript, Ruby
    - mark up/programming hybrid
      - HTML
      - XML, JSTL, XSLT
- Lang. implementation
  - compilation — phases
  - interpretation — at run time, (source, data → interpreter → result) — LISP, javascript, PHP, python, ...
    - adv/dis see (pp. 1-5)
  - Hybrid — after intermediate code gen, (interpreter ← data → result) — perl, java (virtual machine)
  - preprocessors
    - (invoked before compile, for macro expansion, library, include xx.h)

---

## Ch2 — skip

## Ch3 — Syntax & Semantics

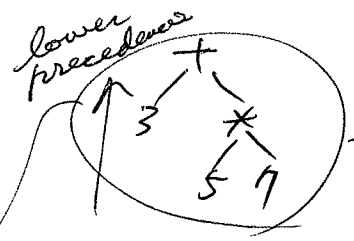
— token/lexeme

ex) index = 2 * count + 17 ; — lexemes — actual instance (value)

id, equal sign, int_literal, mult_op, id, plus_op, int_literal — tokens — category

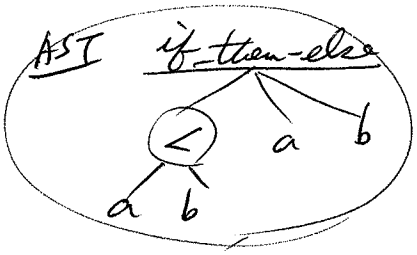
— expression notation

- prefix — +3*5 7
- infix — 3+5*7 (can use ( ))
- postfix — 3 5 7 * +
- AST




— parse tree should resemble AST

mixfix — ex) if a<b then a else b




- precedence — lower prec. high
- associativity
  - left-assoc → left-rec G
  - right-assoc → right-rec G.

— CFG = (V, T, P, S) — ex)

$E \to E+T \mid E-T \mid T$
$T \to T*F \mid T/F \mid F$
$F \to Num$
$Num \to 0 \mid 1 \mid \cdots \mid 9$

— parse tree — G and string (AST — string only)

— derivation
- right-most
- left-most — $E \Rightarrow E+T \Rightarrow T+T \Rightarrow F+T$ ⋮ all terminals

— Syntactic ambiguity — ∃ multiple parse tree (or derivations)

ex) $E \to E-E \mid 0 \mid 1 \mid \cdots \mid 9$

string 2-4-5

— writing unambiguous G.

- keep precedence
- reflect associativity — left-assoc → left-rec G
  right-assoc → right-rec G

eg) ^ power.

eg) $E \to E - E' \mid E'$
$E' \to 0 \mid 1 \mid \cdots \mid 9$

— conversion of left-rec G → right-rec G without violating the associativity of operation.

$$\boxed{A \to A\alpha \mid \beta \Rightarrow \begin{cases} A \to \beta A' \\ A' \to \alpha A' \mid \varepsilon \end{cases}}$$ — 2 prod. case.

eg) $E \to E \underbrace{+T}_{\alpha'} \mid \underbrace{T}_{\beta'} \Rightarrow \begin{cases} E \to \overset{\beta}{T} E' \\ E' \to \underbrace{+T}_{\alpha'} E' \mid \varepsilon \end{cases}$

— from prog. assignments

include ( ) and a to the appr G.

— dangling-else ambiguity

Solutions
1. if ---- end — Modula-2, Ada
2. Compiler matches else to the nearest if — C/C++
3. Java way — separate products for if-state, if-else-state.

— BNF / EBNF / Syntax chart

( ) group
{ } 0 or more
[ ] optional

1 chart for 1 non-T.

BNF → EBNF

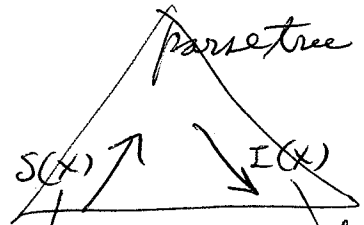
eg) 2 products ← optional [ ]
BNF. EBNF

## Semantics

— attribute grammar — CFG + static semantics

ex) type checking

ex) Ada [ proc. name ≡ ) statement end name ] — must be same




synthesized attribute

inherited attribute ) → or → (from siblings)

for each syntax rule — production — ex) <assign> → <var> = <expr>

predicate (semantic rule) — ex) <expr>. expected type ← <var>. act type

inherited

synthes

static semantics — checking at compile time.

---

dynamic semantics — describing meaning of constructs.

— state of a program

$\sigma$ — set of $\langle V, val \rangle$

$\sigma = \{ \langle v_1, val_1 \rangle, \langle v_2, val_2 \rangle, \ldots \}$

system state

— operational semantics — state change defines the meaning of the statement.

— Axiomatic "

based on formal logic (predicate calculus)

formal verification

## — Operational semantics

— Addition operation

$$\frac{\sigma(e_1) \Rightarrow v_1, \quad \sigma(e_2) \Rightarrow v_2}{\sigma(e_1 + e_2) \Rightarrow v_1 + v_2}$$

— Assignment st (s.target = s.source)

$$\frac{\sigma(s.\text{source}) \Rightarrow v}{\sigma(s.\text{target} = s.\text{source};) \Rightarrow \sigma \cup \{\langle s.\text{target}, v\rangle\}}$$

eg) $\sigma = \{\ldots, \langle x, 3\rangle, \ldots\}$

$$\frac{x = 5;}{\sigma' = \sigma \cup \{\langle x, 5\rangle\} \Rightarrow \{\ldots, \langle x, 5\rangle, \ldots\}}$$

— Sequence of statements (S1; S2)

— Conditionals (if (s.test) s.thenpart else s.elsepart)

[ true case, false case ] rules

— loops

[ True case, false case ] rules

eg) $\sigma = \{\ldots, \langle x, 7\rangle, \ldots\}$

```
while (x<100)
  x = 2*x;
```

⇒ what is the final state $\sigma'$?

- Axiomatic Semantics

  precond / post cond.

- partial correctness proof rules

  - Composition rule

$$\frac{\{p\}\ S_1\ \{Q\},\ \{Q\}\ S_2\ \{R\}}{\{p\}\ S_1 ; S_2\ \{R\}}$$

premise

Conclusion

  - Conditional rule

$$\frac{\{p \wedge E\}\ S_1\ \{Q\},\ \{p \wedge \neg E\}\ S_2\ \{Q\}}{\{p\}\ \text{if } E \text{ then } S_1 \text{ else } S_2\ \{Q\}}$$

  - while rule

$$\frac{\{p \wedge E\}\ S\ \{p\}}{\{p\}\ \text{while } E \text{ do } S\ \{p \wedge \neg E\}}$$

  - Assignment Axiom

target = source;

$$\frac{\text{True}}{\{Q[\text{target}/\text{source}]\}\ S\ \{Q\}}$$

  - rule of consequence

- type systems and semantics (partly from Ch6.)

  - statically typed lang. / dynamically typed lang.
  - strongly typed lang.
  - type errors (3)
  - func. for { type-map building / name uniq. checking