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 → interpreter ← data, interpreter → result
    - adv/dis see (pp. 1-5)
    - LISP, Javascript, PHP, python, ...
  - Hybrid → after intermediate code gen, → interpreter ← data, interpreter → 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 ;

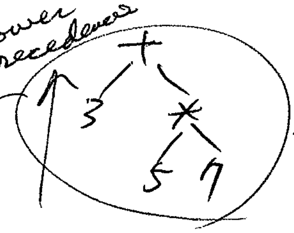
index — id; = — equal sign; 2 — int_literal; * — mult_op; count — id; + — plus_op; 17 — int_literal

— lexemes — actual instance (value)

— tokens — category

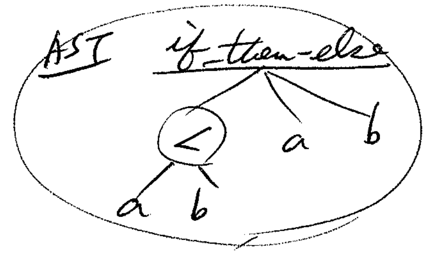
— expression notation

- 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) $E \to E - E'$
$\quad | E'$
$E' \to 0 | 1 | \cdots | 9$

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

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

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

— from prog. assignments

include ( ) and ^ to the expr 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
- { } — ϕ or more
- [ ] — optional

1 chart for 1 non T.

BNF → EBNF

eg) 2 products ← optional [ ]

BNF. EBNF

— Semantics

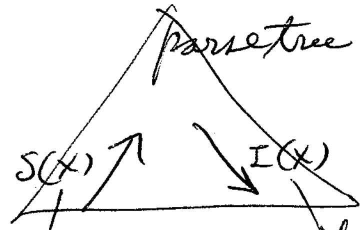
— attribute grammar — CFG + static semantics

ex) type checking

ex) Ada

proc. name
≡) statements
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 — synthesiz

static semantics — checking at compile time

— dynamic semantics — describing meaning of constructs.

— state of a program

σ — set of $<V, val>$

$\sigma = \{<v_1, val_1>, <v_2, val_2>, \dots\}$

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

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

Assignment st (s.target = s.source)

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

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

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

- 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

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

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

$\Rightarrow$ what is the final state σ'?

— 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

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

— rule of Consequence

Ex 1

— Type systems and Semantics — partly from Ch 6

exclude — type checking functions ...