

CS 164

- o Please read all instructions (including these) carefully.
- o Please print your name at the bottom of each page on the exam. i
- o There are seven questions on the exam, each worth between 10 and 20 points. You have 1 hour and 20 minutes to work on the exam, so you should plan to spend approximately 12 minutes on each question.

Midterm Examination

- o The exam is closed book, but you may refer to your two sheets of prepared notes.
- o Please write your answers in the space provided on the exam, and clearly mark your solutions. You may use the backs of the exam pages as scratch paper. Please do not use any additional scratch paper.
- o Solutions will be graded on correctness and clarity. There are no "tricky" problems on the exam—each problem has a relatively simple and straightforward solution. You may get as few as 0 points for a question if your solution is far more complicated than necessary.

NAME:

SID or SS#:

Problem, Max points, Points

1 15

2 15

3 20

4 10

5 20

6 10

7 10

Fall 94 page 1 of 8

CS 164 Handout 11 1. Regular Expressions (15 points) Consider a language where real constants are defined as follows: A real constant contains a decimal point or E notation, or both. For instance, 0.01, 2.71821828, -1.2E12, and 7E-5 are real constants. The symbol '-' denotes unary minus and may appear before the number or on the exponent. There is no unary '+' operation. There must be at least one digit to left of the decimal

point, but there might be no digits to the right of the decimal point. The exponent following the "E" is a (possibly negative) integer. Write a regular expression for such real constants. Use the standard regular expression notation described by the following grammar: $R \rightarrow (\epsilon | \text{char} | R + R | R^* | RR | (R))$ You may define names for regular expressions you want to use in more than one place (e.g., $\text{foo} = R$). $\text{digit} = 0+1+2+3+4+5+6+7+8+9$ $\text{posint} = \text{digit}^*$ $\text{int} = ((\epsilon | + | -)) \text{posint}$ $\text{exp} = E \text{int}$ $\text{frac} = \text{.digit}^*$ $\text{real} = (\text{int} \text{frac} (\text{exp} + \epsilon)) + (\text{int} (\text{frac} + \epsilon) \text{exp})$ Fall 94 page 2 of 8

CS 164 handout 11

2. Finite Automata (15 points)

Consider a DFA with a start state s_0 and a transition function trans . For a state s and input character c , $\text{trans}(s, c) = s'$ if there is a transition from state s to state s' on character c . If there is no transition from s on c then $\text{trans}(s, c) = \text{none}$. The following algorithm simulates the behavior of such a DFA, accepting if the input string is accepted by the DFA and rejecting

otherwise.

state $\leftarrow s_0$

while there's input left do:

char \leftarrow next input character

if $\text{trans}(\text{state}, \text{char}) = \text{none}$ then stop and reject

state $\leftarrow \text{trans}(\text{state}, \text{char})$

(end of loop)

accept if state is a final state, otherwise reject

Now consider an NFA with a start state s_0 and a transition function trans . In this case, for a state s and input character c (we now allow $c = \epsilon$), $\text{trans}(s, c)$ is the set of states s' for which there is a transition from s to s' on c . In the style of the algorithm above, give a (deterministic) algorithm that simulates the behavior of such an NFA. You may use the epsilon-closure operation described in class and in the text.

Simulate subset construction.

Use state-set instead of state,

state-set $\leftarrow \epsilon\text{-closure}(s_0)$

while there's still input left do:

char \leftarrow next input character

if $\epsilon\text{-closure}(\text{Union } s \text{ subset state-set } \text{trans}(s, \text{char})) = \emptyset$ then stop and reject state-set - $\epsilon\text{-closure}(\text{Union } s \text{ subset state-set } \text{trans}(s, \text{char}))$

(end of loop)

accept if there's a final state in state-set, otherwise reject.

Fall 94 page 3 of 8

CS 164 Handout 11

3. Grammars (20 points)

Consider the following grammar. The nonterminals are E, T, and L. The terminals are +, id, (,), and ;. The start symbol is E.

$$E \rightarrow i \mid E+T \mid T$$

$$T \rightarrow id \mid id() \mid id(L)$$

$$L \rightarrow E; \mid L E$$

Give an LL(1) grammar that generates the same language as this grammar. As part of your work please show that your grammar is LL(1).

(a) Eliminate left recursion:

$$E \rightarrow TE$$

$$E' \rightarrow +TE \mid \epsilon$$

$$T \rightarrow id \mid id() \mid id(L)$$

$$L \rightarrow E; \mid L E$$

(b) Left factor:

$$E \rightarrow TE'$$

$$E' \rightarrow +TE' \mid \epsilon$$

$$T \rightarrow id T'$$

$$T' \rightarrow \epsilon \mid () \mid L$$

$$L \rightarrow EL'$$

$$L' \rightarrow E; \mid L E$$

$L' \rightarrow \epsilon$; $L \text{ epsilon}$

(c) Check that it's LL(1). For this part you just needed to give enough information to show that there would be no conflicts in the parsing table. The following is sufficient:

$E \rightarrow +TE \text{ epsilon}$ $\text{First}(+TE) = \{+\}$

$\text{Follow}(E) = \{ \$,), ; \}$

$T \rightarrow \epsilon$ $\text{First}(T) = \{ (\}$

$\text{Follow}(T) = \{ +, \$,), ; \}$

$T \rightarrow)L$ $\text{First}(T) = \{) \}$

$\text{Follow}(L) = \{ \text{id} \}$

$L \rightarrow ;L \text{ epsilon}$ $\text{First}(L) = \{ ; \}$

$\text{Follow}(L) = \{ \}$

Fall 94 page 4 of 8

CS 164 Handout 11

4. Parsing (10 points)

In both parts of this question, we are looking for clarity and brevity as well as the right idea. Suppose you are writing a parser for a programming language that includes the following syntax for looping constructs:

Loop \rightarrow do stmt while expr

| do stmt until expr

| do stmt forever

(a) (5 points) A predictive parser (i.e., a top-down parser with no backtracking) can't use this grammar. Give a brief (a couple of sentences) explanation of this fact.

When trying to expand a production for non-terminal "loop I I", the parser cannot decide which of the three productions to expand using only the next few input tokens.

(b) (5 points) Give a brief explanation of why a bottom-up parser does not have difficulty with this grammar.

When a bottom-up parser must decide which of the three productions to choose (reduce), the "while", "until", or "forever" has already been read and shifted onto the stack.

Fall 94 page 5 of 8

CS 164 Handout 11

5. Parsing (20 points)

Consider the following grammar. The nonterminals are S' and S . The terminals are op and x . The start symbol is S' .

$S' \rightarrow S$

$S \rightarrow S OP SIX$

(a) (15 points) Draw the DFA built from sets of LR(O) items for this grammar. Show the contents of each state. (Note: Don't augment the grammar with a new start symbol.)

(b) (3 points) Is this grammar SLR(1)? Briefly explain why or why not.

(c) (2 points) Is this grammar LR(1)? Briefly explain why or why not.

Fall 94

page 6 of 8

CS 164 Handout 11

6. Bison and Abstract Syntax 'JPrees (10 points)

Consider the following constructors for a tree language:

Expression app(Expression, Expression); Expression lambda(Expression, Expression); Expression ido;

Now consider the following Bison grammar:

```
%token ID LAMBDA
```

```
%type <Expression> Expr
```

```
%%
```

```
Expr : ID
```

```
{ $$ = id() ; }
```

```
| '(' LAMBDA ID '!' Expr ')'
```

```
{ $$ = lambda( id(), $5); }
```

```
| (' Expr Expr ')
```

```
{ $$ = app($2, $3); }
```

Draw the abstract syntax tree that would be produced when parsing the sequence of tokens below. Label all the nodes of your AST with the appropriate constructor.

```
(( ( LAMBDA ID . ( ID ID ) ) ( LAMBDA ID . ID ) ) )
```

Fall 94 page 7 of 8

CS 164

Handout 11

7. Type Checking (10 points)

Suppose we want to design a type checker for Scheme programs. In Scheme, functions can be passed as arguments to and returned as results from functions. Recall the type checking rule for function application given in class:

$$\frac{A \vdash f : t_1 \rightarrow t_2 \quad A \vdash x : t_1}{A \vdash f(x) : t_2}$$

Self-application occurs when a function is called with itself as a parameter. This is, if f is a function taking a function as a parameter, the $f(f)$ is an instance of self-application. Briefly explain why type checking of self-application must always fail using the type checking rule above.

**Posted by HKN (Electrical Engineering and Computer Science Honor Society)
University of California at Berkeley
If you have any questions about these online exams
please contact examfile@hkn.eecs.berkeley.edu.**