COMPUTER SCIENCE TRIPOS Part IB – 2025 – Paper 4

1 Compiler Construction (jdy22)

Here is a fragment of a lexer specification for a programming language:

`int`	⇒	`INT`	the type name `int`
`[a-z]+`	⇒	`IDENT`	a user-defined identifier
`type`	⇒	`TYPE`	the keyword `type`
`[ \n]+`	⇒	*skip*	whitespace

The lexing rules map regular expressions to tokens or to the special action *skip*.

(*a*) The specification is ambiguous: multiple lexing rules match the same string.

(*i*) Give an example of ambiguity that is resolved by the *first match* rule. [1 mark]

(*ii*) Give an example of ambiguity that is resolved by the *longest match* rule. [1 mark]

(*iii*) Identify an ineffectual lexing rule in the specification. [1 mark]

(*b*) Convert the specification to a tagged deterministic finite automaton (DFA). [7 marks]

(*c*) Give a new lexing rule that skips over comments. Comments start with the two-character sequence `{-` and end with the two-character sequence `-}`, and can contain any characters except the closing sequence `-}`.

You can use standard operators ($\emptyset$, `c`, ϵ, $r_1|r_2$, r_1r_2, $r*$, $r+$) and positive and negative character sets such as `[a-z]` and `[^a-z]`. [3 marks]

(*d*) One way of constructing lexer DFAs is to use *derivatives*. The derivative of a regular expression r with respect to a character c is another regular expression $\partial_c\, r$ that matches s if r matches cs.

(*i*) Define ∂_c for sequencing r_1r_2, intersection $r_1 \& r_2$ and negation $\neg r$. [3 marks]

(*ii*) Give a set of derivatives that can be used to construct a DFA for the lexer specification, including the lexing rule for comments from part (*c*). Omit cases that produce $\emptyset$. You do not need to actually construct the DFA. [4 marks]