Solutions

Marking: **M1** a correct method, **A1** the correct answer, **B1** an independent mark.

Each answer shows the steps, then the **result**.

2.1

E ⇒ E + T ⇒ T + T ⇒ F + T ⇒ id + T

⇒ id + T * F ⇒ id + F * F

⇒ id + id * F ⇒ id + id * id

2.2

`id + id * id` has a tree with `+` at the root

and a tree with `*` at the root: two trees, so ambiguous

2.3

`S → a S b | ε`

`S → ( S ) S | ε`

2.4

push one symbol for each `a`

pop one for each `b`; reject on an `a` after a `b`, or a `b` with an empty stack

accept when the input ends and the stack is empty

3.1

(a) the `else` can belong to the inner `if`, or to the outer `if`, so there are two trees

(b) the `else` goes with the nearest unmatched `if`

(c) `S → M | U`; `M → if E then M else M | other`; `U → if E then S | if E then M else U`

(d) the indentation shows which `if` the `else` belongs to

3.2

$a^n b^n$: context-free

even number of 1s: regular (a 2-state DFA)

$a^n b^n c^n$: not context-free

palindromes: context-free

(e) a pushdown automaton

3.3

(a) `block → NAME { items }`; `items → item items | ε`; `item → block | NAME = VALUE ;`

exercises.lol exercises.com.cn

(b) blocks can nest to any depth, like balanced brackets; a finite automaton cannot count unbounded depth, so the language is not regular

(c) both choices start with NAME, so factor it out: `item → NAME rest`; `rest → { items } | = VALUE ;`

(d) use the grammar with a small recursive-descent parser (one function per rule), or an existing format such as JSON with a ready-made parser, which gives clear error messages