Solutions

2.1

- Identifiers to (type, scope); built during the semantic walk as declarations are seen ✓.

2.2

- One: `x = 6` – the literal product folds ✓.

2.3

- $m \times n$ direct compilers become $m + n$ components: each frontend targets the one IR, each backend starts from it ✓.

3.1

- IR: `t1 = b * c; t2 = a + t1; t3 = d / e; t4 = t2 - t3; r = t4` ✓.
- Five instructions, four temps ✓.
- Checks: $b \times c$ (int $\times$ int), $a + t_1$ (int + int), d/e (int / int), $t_2 - t_3$ (int $-$ int) – all pass ✓.
- With `d: str`: `t3 = d / e` fails – reported at `d`'s lexed position ✓.

3.2

- (1) `y = 6 + x` – the literal product folds at IR build: one instruction ✓.
- (2) `t1 = x * 3; y = t1 + 2` – `x * 3` is not literal: no fold ✓.
- Rule: fold only when every operand of the node is a literal ✓.
- The IR argument: without it every (language, target) pair needs its own full compiler – mn of them; the IR makes each side write once against a fixed middle ✓.

exercises.lol exercises.com.cn