p 2q 6
AJRGM

SOLUTION NOTES

(These are some of the things that might occur in a good answer.)

Discuss

(a) your program is faulty

Even if the program is faulty, if all else is correct the system could not produce this result. Thus is because the program goes though a type-checker, and the ": int" part of the result is the type-checker claiming that your program can produce only integers as output. It makes this claim having inspected the program and seen that the functiom which produced the printed answer is indeed a function that can only produce integers, whatever arguments it receives.

(b) the computer's calculations with real numbers is inaccurate

This might be an explanation for some languages, but not for ML. For example, in another language, if a typechecker sees the program "(b * x)/x" where it knows that b is an integer variable and x is a real variable, it could conclude that the result of evaluating the expression must be an integer, so (see (a)) it will cause this type to be printed after whatever value is computed at run time; then, unfortunately, a faulty run-time calculation with b = 2 and x = 4.702, done with floating point aritmetic, might produce the actual value 1.99. But ML could never ascribe the type "int" to an expression involving real-number division "/".

(c) the ML implementation is faulty

This is the most likely reason. The simplest way it could cause the response "1.99 : int" is if the print routine is at fault. Thus, although the type-checker concluded that the type of the result should be "real", amd told this to the print routine, the print routine did some silly table-lookup and printed "int" instead. But it is very unlikely that such a bad print routine could have survived debugging! A more likely explanation is that the type-checker itself is faulty.

(d) ML is a badly designed language.

Even if true, this is unlikely to be the reason. It is just possible though; the design of ML, which specifies how the typechecker should work, might have been wrongly formulated so that it specifies that the type "int" should be inferred from programs like the one in (b) above.

Discuss

"A type system is a way of ensuring that every program does what it is supposed to do."

This is partly correct, depending on what "supposed to do" means.

If it means "supposed to apply every function only on inputs that make sense for that function" then it is very nearly correct, though in may not pick up division by zero.

Agian, If it just means "supposed to compute an integer as result" then the type system does ensure it, by means of the type-checker, provided that the program terminates. (A type system can't ensure termination.) Of course there will be programs that the typechecker rejects (for other reasons) but which would always produce in integer if they were allowed to run; so typecheckers may reject programs that work.

But the statement is incorrect if it means for example "supposed to produce a list of numbers in ascending order"; this kind of specification can't be checked by a type-checker.

But there are sophisticated specifications that _can_ be checked in a type system. For example, using abstract types we can ensure that a program performs mathematical proofs that are guaranteed to be correct. So if your specification is "prove Fermat's last theorem" and the program terminates with a proof of the appropriate mathematical formulae, then this is guaranteed to be a correct proof. (If the abstract types were set up correctly.) What is not guaranteed is that is will produce a proof of the correct formula, or that it will produce any result at all (it may not terminate!)