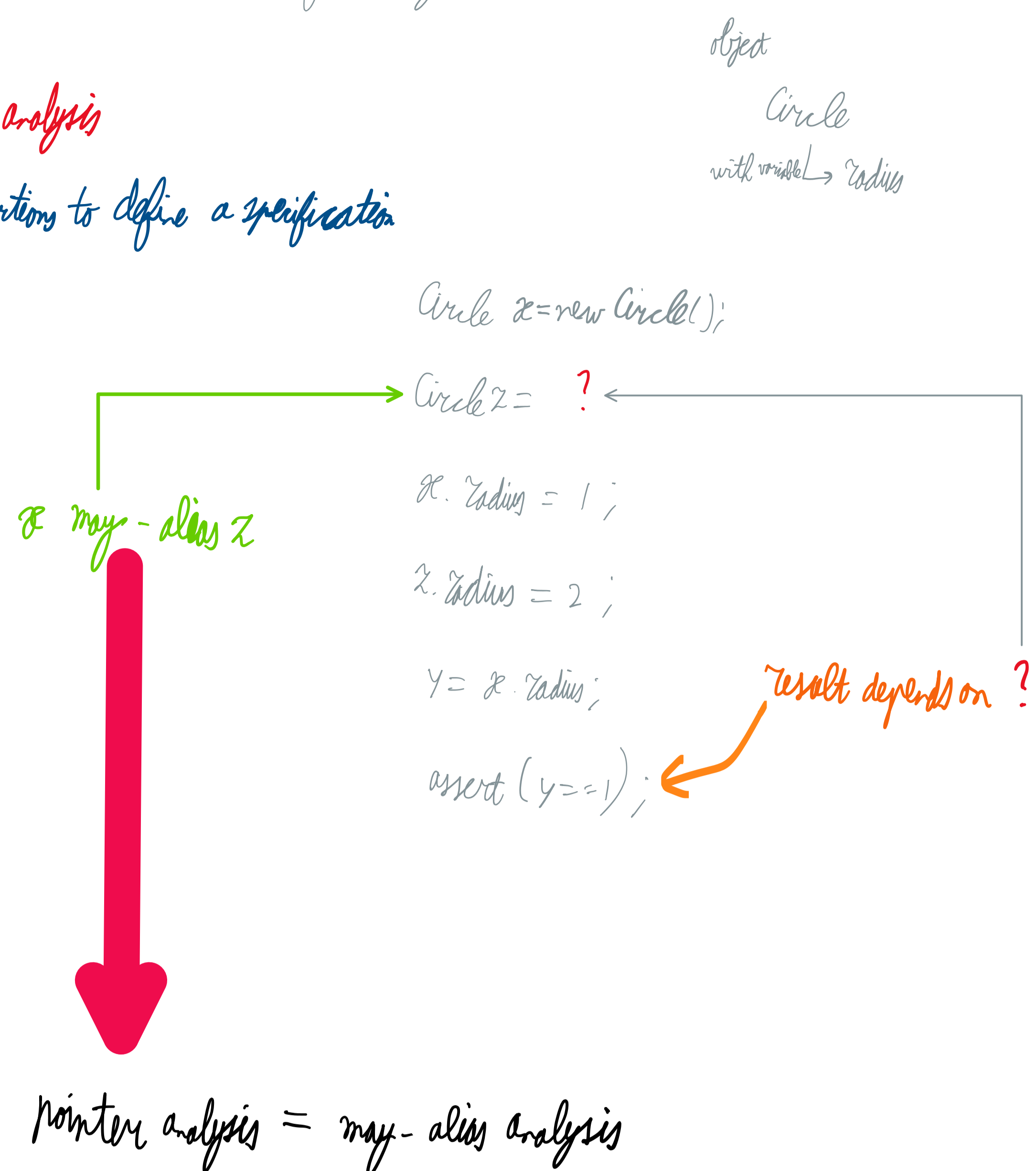


"reflect on arguments" question from homework: could be an exam question

pointer analysis usually done in addition to data flow analysis

problems with pointer analysis

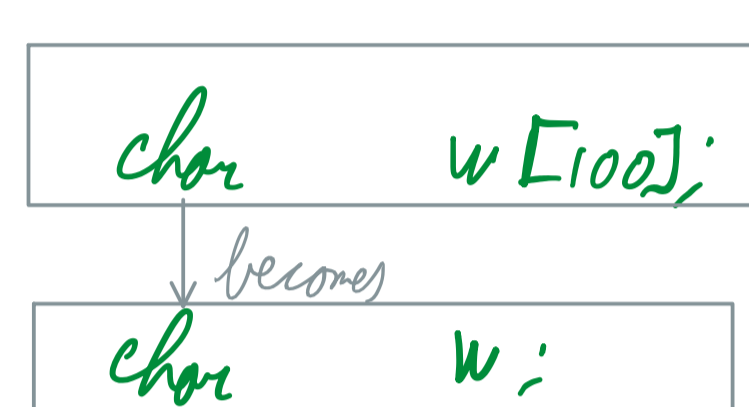
assume we use assertions to define a specification



Situations like this one make pointer analysis difficult

since there are infinitely many ways of referring to the same memory element

rough solution: simplify program by aggregating groups of pointers into a single pointer



if there is pointer aliasing in the original program, then this will still be the case in the simplified program

field-insensitive aggregation: merge all fields of each object  $o_1.F_1 = o_2.F_2$

field-based aggregation: merge all fields of all objects  $o_1.F_1 = o_2.F_1$

field-sensitive: each field of each object kept separate

fuzzing: testing 'smootherly' for errors

↓  
 there is no 'theory' behind this

+ scales well

+ no false alarms

- more expensive

- not guaranteed to find all bugs

+ does not require code

- test may launch attack (in production environment, so use in production environment)

- does not (fully) explain cause of bugs

Run program with generated input, possibly syntactically/semantically invalid

Reduce search space of inputs to 'interesting' e.g. for attacker candidates

testing can be adaptive  $\rightarrow$  previous results influence newly chosen inputs

heuristics:

- testing should be smarter than random/brute force

- code coverage

$\rightarrow$  find input values which are almost good

- configs which found bugs are favorable

- avoid redundancy / useless test cases

- test cases should evaluate sufficiently quickly

- provide feedback to design test cases

code coverage: metric to quantify extent to which a program's code is tested

function coverage: which functions were called?

statement coverage: which statements were executed?

branch coverage: which branches were taken

PUT = Program Under Test

fuzzing: execution of PUT using inputs sampled from input space that protrudes the expected input space of the PUT

fuzz testing: use of fuzzing to test if a PUT violates a security policy

fuzzer: program that performs fuzz testing on a PUT

fuzz campaign: specific execution of a fuzzer on a PUT with a specific security policy