1 Requirements

The process of **requirements analysis** and **system spcification** involves examining the behaviour of the proposes system and asking questions which enable a **correct** specification to be developed.

- requirements analysis
- system specification
- understand behaviour
- ask questions
- enable correct specification to be developed
- think above code level
- start with specification then implementation
- specification drives design
- testing drives implemetation

Key Questions: 1)Writing a spec to drive design. Using math to write formal spec. Then model checking. 2)Saftefy critical obligations.

2 Thank Above Code Level by Leslie Lamport

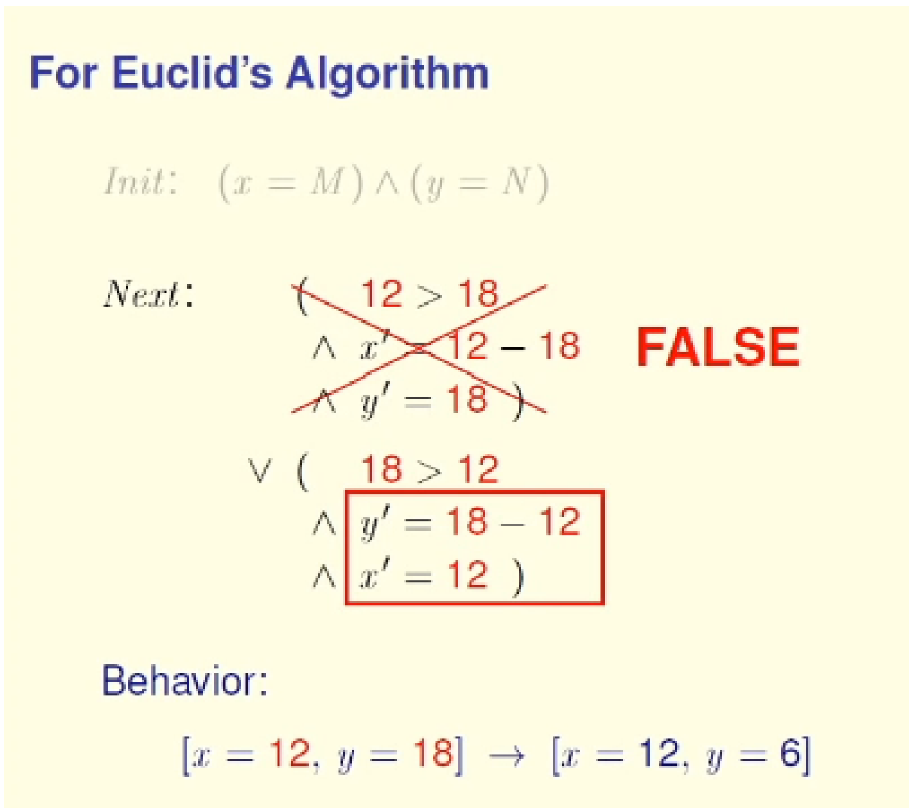
1. If thinking without writing then you're only pretending to think.
2. What to write? b̄lueprints of Programs called specifications.
3. Specifications? S̄pectrum of blueprints (i.e. simple or complex, formal(mathematical) or informal(prose)).
4. The best type of specifications are in middle called Mathematical Prose.
5. Code for concurrent or distributed systems is going to be complex, subtle and critical.
6. Thus, there is a need for tools to check your blueprint.
7. Therefore, if you are going to use tools then you're going to need a formal language. This is because tools dont understand prose.
8. So how to write a Spec?
    - Writing requires thinking. So how to think about programs?
    - You should think about programs like Scientists. S̄cientific thinking M̄ake Mathematical Models of Reality.

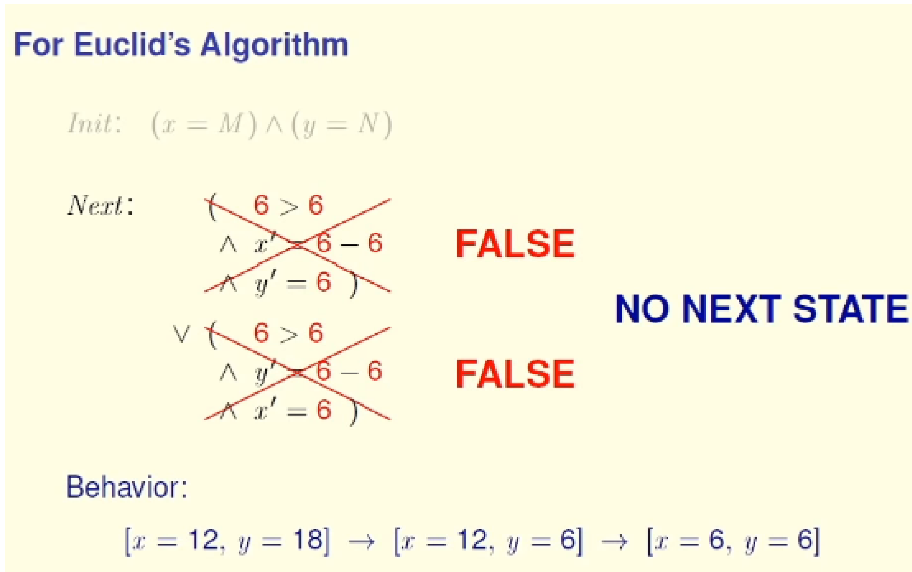
- In CS, reality consists of digital systems. p̄rocessor chip or computer executing a program.

9. So, Models? F̄unctions and Sequences of States are the two most useful basic models, according to Leslie Lamport.
10. Function? w̄e can model a program as a function that maps input ōutput, or multiple inputs m̄ultiple outputs.
11. In Math, function is a simply a set of ordered pairs.
    - $< 0, 0 >, < 1, 1 >, < 2, 4 >$
    - square(2) = 4
    - all the first elements of those pairs compose the domain of the function.
12. To define a function, we specify it's domain:
    - Domain of square = NAT (0,1,2,3,4,5) vs NAT1 which excludes 0.
    - square(x) = x^2 for each x in its domain. (this is the definiton!)
13. Fuctions in math are simpler than functions in programming languages.
14. The Functions Model's main limitation is īts specifies what a program does. Its does not specify how it does it.
15. Futhemore, other limitations are:
    - how do we specifiy some programs that dont just map inputs to outputs.
    - how do we specifiy programs, like OS which we assume run forver (b/c its convinient to think like this mathematicially).
    - Thus, we use the Standard Behavioural Model P̄rogram execution is represented by a behaviour.
    - Behaviour is a sequence of staes.
    - A state is an assignments of values to variables.
    - Thus, a Program is modelled as a set of behaviours. t̄he behaviours that represent all possible executions of the program.
16. So how do we Specify a set of behaviours?
    - by specifying a Saftey Property and Lebniz Property.
    - In practise, specifying saftey just turns out to be more important (b/c thats is where most erros are likely to occur).
17. How to Specify Saftey Property?
    - Two things:

(a) The set of all possible inital states.

(b) The next-state relation, describing all possible successor states of any state.

- So what Language should use to write these two things? w̄e use MATH. i.e. as show in his Euclid's Algorithm example, Lamport generates two formulas- one for set of initial states and other for next-state relation.
- So how does this work? i.e. How do we get behaviours out of those formulas.
- 

- 

- To Model non-determinism, use just have a next-state relation that allows multiple next-states for a current state. (there nothing magic or difficult about non-determinism)

18. What about Formal Specs? w̄e need formal specs ONLY to apply tools.

19. So we need a formal language called TLA+.

20. You can model check TLA+ specs c̄hecks all possible execution of program, on a very small model

21. It is extreamly EFFECTIVE and EASY to do.

    - You basically tell the model checker what the model is.
    - Models are instantiating value of constants. For eg, in Euclid's algorithm we'd have to tell checker what M and N are.
    - Benefit: You can write formal correctness proofs and check them mechanically in TLA+.
    - We do this by writing proofs in TLA+ then use theorm prover to check the proofs.
    - TLA+ is formal (language of mathematica) and PLUSCAL is puedocode. It looks like a programming language.

22. Everyone thinks they are thinking but you're not writing down your thoughts, then you're fooling yourself.

23. What programmer should know about thinking? ȳou should think before you code w̄rite a spec before you code.

24. What code should you specify? āny peice of code that someone else might want to use or modify.(eg: entire programmer system, class, method, or peice of code inside a method.)
25. What should you specify about the code? W̄hat it does, and how it does it. This called an Algorithrm or high-level design.
26. How should you think about or specify your code? ābove the code level!- in terms of states and behaviours or functions. Mathematically/Rigiours.
27. Should be thinking Mahtematically, eventhough, your're writing specs informally which psudocode i.e. PlusCal.
28. How do you learn to write specs? b̄y learning how to write formal specs t̄his will help you write infromal specs, which you will actually write.
29. You learn to write programs by writing them, running them and correcting your errors.
30. You can learn to write formal specs by writing them, running them with model check and correcting your errors.
31. TLA+ is a therefore a language for wiriting formal specification- it is great for **learning how to think mathemaitically.**
32. Writing Specs is hard b/c Thinking is hard. There is no royal road to Mathematics.