4/10 - Higher Order Quantification

Higher-order quantification:

- When we write `some x: A | x in B` in Forge, this becomes a big disjunction of concrete possibilities of a's in A
    - One possibility per a in A (i.e. linear number of disjuncts)
    - If we specify bound to be 5 A, we only need to check 5 things
    - Looking for a singleton (1 col, 1 row)
    - This is *first-order quantification*
- However, if we write some y: A -> A | s in B->B, this becomes huge!
    - Now, we're looking for arbitrary relation
    - This will grow exponentially!
    - If we specify bound to be 5 A, we need to check any set of 5^2 tuples
        - So we need to check 2^(5^2) disjuncts!!
    - This is *second-order quantification*
- Higher-order quantification (HOQ) is useful!
    - E.g. checking the minimum-ness of a spanning tree
- Forge can't do higher-order universal quantification:(
    - There are tools that can work around this in clever ways - e.g. **Alloy***
    - They use a thing called Counterexample-Guided Inductive Synthesis (CEGIS)
- It can handle second-order existential quantification using skolemization

Skolemization:

- When we have some x : A->A | x in B->B,
- We can give a witness to the some quantifier (i.e. the thing we fill in for x to satisfy it)
    - It gives us a lot more instances since we can fill in different things for the $x
    - We get more information when looking at the information
- Forge can handle second-order existential quantification using skolemization

There are also tools called **SMT solvers**:

- These are SAT modulo theory solvers
- Support theories of e.g. integers, strings, etc.
    - Aware of how these domains work
- There are bindings for z3 for Python, Haskell, Racket, Java, etc.