2/7 - Recursion & Prop. Semantics

Syntax vs semantics:

- Heute regnet es.
    - If we don't speak the language, it's just syntax
    - Knowing what it means → semantics of it ("It's raining today!")
    - There's some function that takes syntax and produces meaning.
        - Semantics normally defined recursively
            - Forge has no recursion!

> Can an object hold itself or multiple of itselves? -- Yes

- Build up recursion by using the fact that objects have fields

Need notion of variable that is a special case of our formula

- We can also define subformulas for our ops: And, Or, Not
- Is there such a thing as a formula that is not a variable and is not a boolean operator?
    - `wellformed` constraint: any formula is either a variable or an object of boolean syntax
- If we allow cycles, we run into issues since the `Not` operator will no longer be finite
  > How can we deal with cycles?
    - See if there are any self-loops
        - We need to make sure that when we take all edge relations into consideration, we will still have no loops!
        - Prevent self-loops in each relation with `no iden & ^<field>`
            - When there's multiple relations and you need to prevent cycles that use edges from multiple relations, first take the union of the relations, *then* apply ^.
                - `^(child + oleft + oright + ...)`

- Need to write a constraint that tells us the semantics of our formulas (i.e. what they mean)
- Every Formula is going to have a set of valuations that it is true with respect to
    - Every formula will be either true or false
        - This is what would normally be recursion
    - Not should return the opposite of its child
    - Relational operator that corresponds to boolean And: &
    - Set union (+) for Or, since it only needs one of the children to be true