SWEN-220
Mathematical Models of Software

Let and Functions
Let Statements

• In certain cases, expressions may appear repeatedly or as a sub expression of larger, more complicated expressions

• In such cases, we can factor these expressions out using **let**
Let Statements

- all a: Alias | let w = a.workaddress | a.address = some w => w else a.homeAddress

- all p: Married | let sp = p.spouse | p in Man => sp in Woman

- all a: Alias | let w = a.workAddr, h = a.homeAddr | a.address = (some w => w) else h
Functions

• Is a named **expression**, with zero or more arguments and an expression for the result

• Functions are good for expressions that you want to reuse in different context
Functions Example

sig Person {
    child : set Person
}
fun descendant (d: Person) : set Person{
    d.^child
}

Usage:
no p : Person | p in descendant[p]
no p : Person | p in p.descendant
Assertions vs. Facts vs. Predicates vs. Functions

- **Facts**: constraints that always hold
- **Predicates**: constraints that may hold, when called upon
- **Functions**: expressions that we want to reused in different context
- ** Assertions**: used to check if our constraints hold