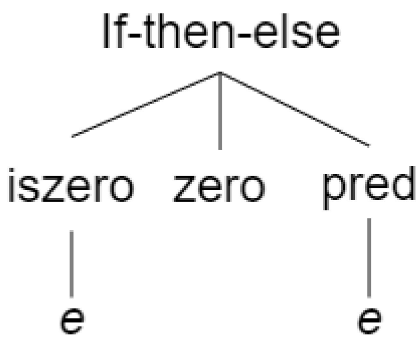
Exercise Sheet 1 - Logic

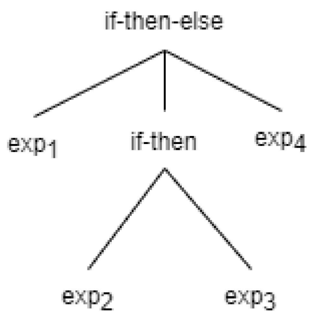
1. Consider the following language for arithmetic expressions that contains a nullary (arity 0) op erator zero, a unary (arity 1) operator succ (successor), a unary operator pred (predecessor), a unary operator iszero (is-zero check), a ternary (arity 3) infix operator if-then-else, a nullary operator true, and a nullary operator false.**

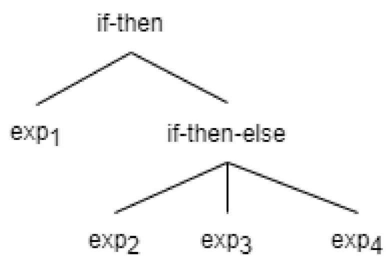
- **Define a BNF for this language. Your BNF should contain a single rule (of the form lhs ::= rhs1 | · · · | rhsn).**
  - *exp* ::= zero|succ(*exp*)|pred(*exp*)|iszero(*exp*)|if *exp* then *exp* else *exp*|true|false
- **Indicate whether some expressions can be ambiguous**
  - No expression is ambiguous
- **Let e be an arithmetic expression. Using this grammar, write down another expression thatreturns zero if e is zero, and otherwise returns e's predecessor. In addition, write down the parse tree corresponding to this expression.**
  - if iszero(*e*) then zero else pred(*e*)



2. Some language also support "if-then" expressions where the infix "if-then" operator takes two arguments: a condition and a "then" branch.

- **Add an infix binary (arity 2) "if-then" operator to your language**
  - *exp* ::= zero|succ(*exp*)|pred(*exp*)|iszero(*exp*)|if *exp* then *exp* else *exp*|true|false|if *exp* then *exp*
- **Indicate whether some expressions can be ambiguous.**
  - There is ambiguous as the "if-then-else" can be mixed up with "if-then"
- **In case some expressions are ambiguous, write down the parse trees corresponding to two different ways an ambiguous expression can be derived.**
  - if exp_1 then if exp_2 then exp_3 else exp_4





3. To use this language as part of a logical system, we can for example add an equality operator.

- **Add a new rule to your BNF for stating equalities between arithmetic expressions.**
  - *exp* ::= zero|succ(*exp*)|pred(*exp*)|iszero(*exp*)|if *exp* then *exp* else *exp*|true|false|if *exp* then *exp*|equal
    equal ::= *exp*=*exp*

- **Define an axiom schema that states that "the expression that given an expression e, checks whether e is zero, and if it is returns zero, else returns e's predecessor" is equal to "e's predecessor", and indicate which variables are metavariables in your axiom, if any.**
  - if iszero(*e*) then zero else pred(*e*)=pred(*e*)
    e should be the metavariables
- **Provide 2 different instances of this axiom.**
  - if iszero(zero) then zero else pred(zero)=pred(zero)
    if iszero(succ(*e*)) then zero else pred(succ(*e*))=pred(succ(*e*))