Lab 07:

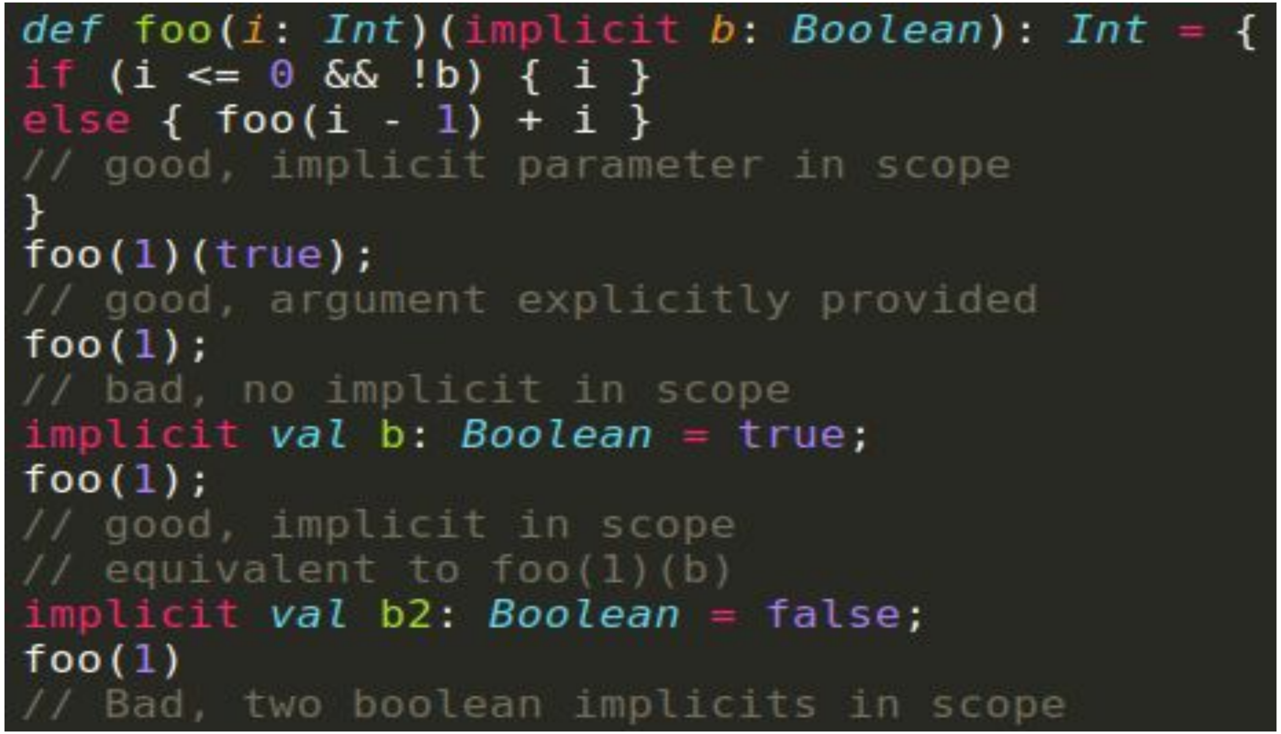
Implicit parameters & implicit conversion

Lucien Iseli, Florian Ravasi, Jules Gottraux

Overview

- What is it, what do they add to the code?
    - Implicit parameters
    - Implicit conversion
- How will we modify the 5-stage pipeline to achieve it?
    - Implicit parameters
    - Implicit conversion

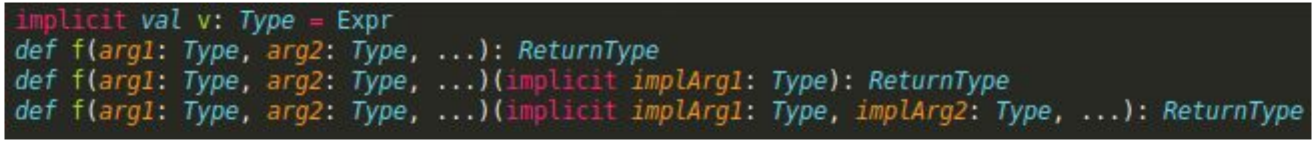
Implicit parameters



```scala
def foo(i: Int)(implicit b: Boolean): Int = {
if (i <= 0 && !b) { i }
else { foo(i - 1) + i }
// good, implicit parameter in scope
}
foo(1)(true);
// good, argument explicitly provided
foo(1);
// bad, no implicit in scope
implicit val b: Boolean = true;
foo(1);
// good, implicit in scope
// equivalent to foo(1)(b)
implicit val b2: Boolean = false;
foo(1)
// Bad, two boolean implicits in scope
```

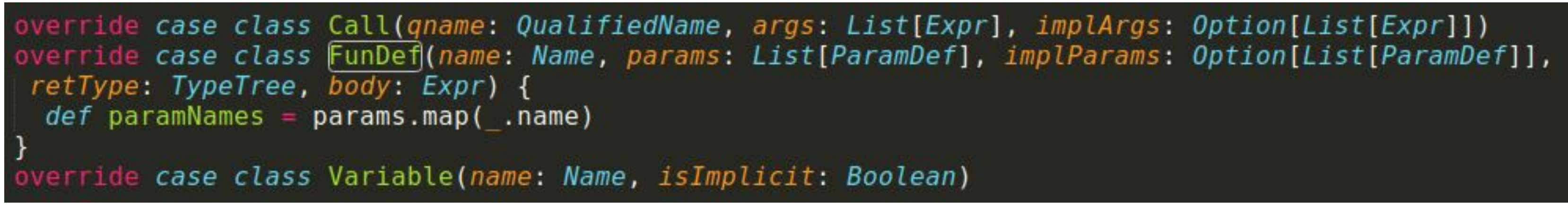
Implicit parameters - Lexer & Parser

- Lexer
  - Implicit token
- Parser
  - First part



```
implicit val v: Type = Expr
def f(arg1: Type, arg2: Type, ...): ReturnType
def f(arg1: Type, arg2: Type, ...)(implicit implArg1: Type): ReturnType
def f(arg1: Type, arg2: Type, ...)(implicit implArg1: Type, implArg2: Type, ...): ReturnType
```

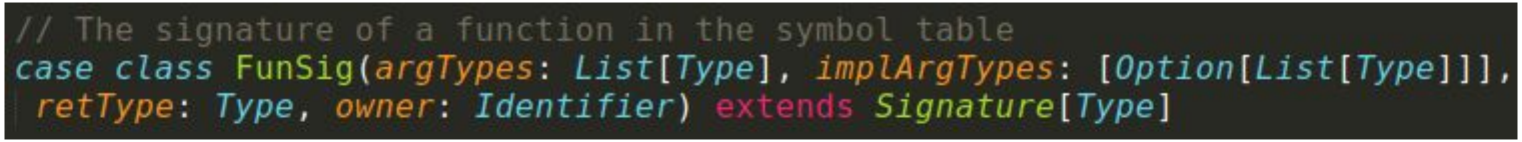
  - Second part



```
override case class Call(qname: QualifiedName, args: List[Expr], implArgs: Option[List[Expr]])
override case class FunDef(name: Name, params: List[ParamDef], implParams: Option[List[ParamDef]],
 retType: TypeTree, body: Expr) {
  def paramNames = params.map(_.name)
}
override case class Variable(name: Name, isImplicit: Boolean)
```

Implicit parameters - Name Analyzer

- FunSig in table will take a parameter for the implicit as well



```
// The signature of a function in the symbol table
case class FunSig(argTypes: List[Type], implArgTypes: [Option[List[Type]]],
 retType: Type, owner: Identifier) extends Signature[Type]
```

- In transformExpr, add a Map[Type, Identifier].
- N.Variable -> S.Variable : Add in the Map if implicit
- N.Call -> S.Call : Check with funSig in table and map

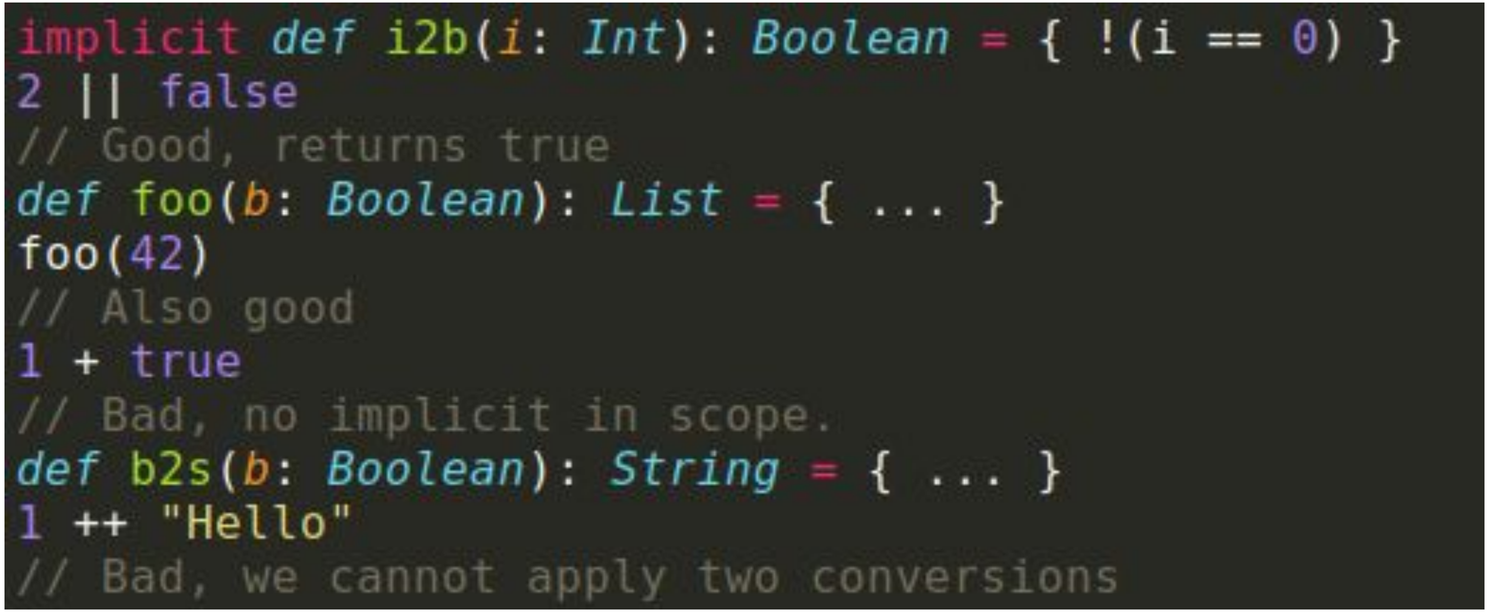
Implicit parameters - TypeChecker

Only small changes due to the table remain because syntax of S.TreeModule unchanged:

- TypeCheck the implicit args added in FunSig

- Conclusion for this first part

Implicit conversions

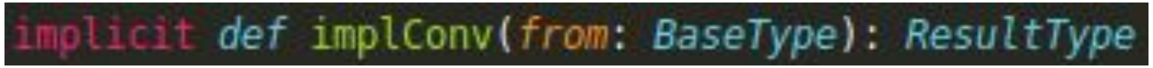


```
implicit def i2b(i: Int): Boolean = { !(i == 0) }
2 || false
// Good, returns true
def foo(b: Boolean): List = { ... }
foo(42)
// Also good
1 + true
// Bad, no implicit in scope.
def b2s(b: Boolean): String = { ... }
1 ++ "Hello"
// Bad, we cannot apply two conversions
```

Implicit conversions - Parser

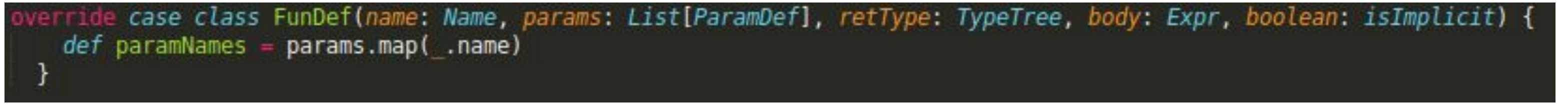
- Modification in lexer already done by the last feature

- Have to handle unique implicit definition of function



```
implicit def implConv(from: BaseType): ResultType
```

- Add boolean in N.FunDef



```
override case class FunDef(name: Name, params: List[ParamDef], retType: TypeTree, body: Expr, boolean: isImplicit) {
    def paramNames = params.map(_.name)
  }
```

Implicit conversions - Name Analyzer

- Create a Map[[Type, Type], Identifier] that stores implicit function in table

- When matching an implicit function, add it to table as normal function and to the map

Implicit conversions - Typechecker

- Replace field position with expression linked with the constraint
- Solve constraints not only solves but updates the expression when needed and returns it

- As a consequence, use that return value for the CodeGen

Conclusion