Project: UPL to Isabelle/HOL compiler

February 4, 2026

1 Introduction

Finish project until end of February. Write interim report. Write final report.

Goal of the project is to implement a compiler from UPL to Isabelle/HOL that translates UPL source code into Isabelle/HOL code. More specifically, the goal is to write Scala 2 code that extends the UPL code base to translate UPL file (.p extenstion) to Isabelle/HOL files (.thy).

Isabelle compiler translates UPL AST into surface-level Isabelle AST and unparses it into Isabelle source code.
Not the Isabelle AST from implementation of Isabelle/HOL, but an intermediate representation (IR) for Isabelle/HOL that is surface-level for the purpose of code generation.

Create a first prototype of an UPL to Isabelle/HOL compiler that represents a first-in road with a few working examples, e.g. the AI search algorithm.

Isabelle surface-level syntax. internal versus external.

Previous Work

Compiler to JavaScript in UPL code base, two files: `Javascript.scala`, `Compiler.scala`

2 UPL AST and Language constructs

Broadly speaking, UPL has types, terms, and language constructs (while, for, assignment).
The following table lists all the UPL language constructs and the Isabelle/HOL equivalents, if they exist.

UPL	Isabelle/HOL
modules	theories
abstract? theories	locales
concrete theories?	interpretations
project	
inheritance	
dependent types	
quotient types	
model types	
type of terms	locales/inductive types

Table 1: UPL language constructs and their Isabelle/HOL equivalents

Representation of UPL AST in Syntax.scala. It includes, declarations, expressions, modules, theories, types, terms, for, while-statements, ...

3 Current Issues

1. How to write a complex number in UPL? Extend test cases and compilation into Isabelle/HOL. + Issue with NumberValue
2. Implement remaining AST:
   (a) At least throw `IError("Not yet implemented")`
   (b) Imperative language features. Mutable variables, assignments, loops. Search for Isabelle/HOL library that simulates imperative programming. AI.p contains imperative algorithms.
   (c) Implement abstract and concrete theory compilation with locales and locale interpretation
3. Subtyping in UPL versus Isabelle/HOL. Isabelle elaborator, coercions, `nat -> int`.
4. Command-line flag `--isabelle` allows passing a main expression that is translated to Isabelle/HOL and evaluated in the context of the project by calling Isabelle. Inside Isabelle expressions can be evaluated with the `value` command.
5. Recursive functions. Currently, UPL functions are generically translated to Isabelle functions (`fun` keyword). Examine `primrec` and other keywords for function declarations. Analyzer that detects recursive functions. Mutual recursion. Future work.
6. Typeclasses. UPL has not typeclasses. Isabelle has typeclasses. Implement Isabelle typeclasses for UPL that can then be compiled to idiomatic Isabelle. (User-generated stubs?)
7. Isabelle/HOL monads.

4 Results

Within UPL project within `package info.kwarc.p`: files `Isabelle.scala`, `IsabelleCompiler.scala`, `IsabelleSyntax.scala`
Extended files `Main.scala`, `Project.scala`, `Traverser.scala`.

Called with command-line option/flag `--isabelle` in `Main`.
Translates project file or source file/folder
Executes toplevel call/main passed as cli argument or in project file with Isabelle `value`.

Advantage of extending the UPL code base versus a standalone program: re-use existing code.

Translation happens after parsing (AST/Syntax) and checking (merging of vardecls with same name & type inference) but before interpretation. Catches type-checking errors but not run-time errors.

4.1 Files

read .p, folder output .thy files.
file names are the UPL module names. In Isabelle, file and theory names must be the same.

4.2 UPL parsing and checking

Compile checked UPL vocabulary to Isabelle/HOL. Translating only parsed but not checked content creates multiple problems. First, type-checking errors are not detected and erroneous UPL code is translated. Second, variable declarations with the same name that are merged to a single declaration during checking must be translated to a single Isabelle/HOL declaration. Third, unknown types are inferred during checking, which simplifies the traverser for package indexing.

4.3 FeatureTraverser for package indexing

Need a method that indexes the packages that are imported by the Isabelle theory. The default imports only the "Main" package. For rational, real, and complex numbers the main package must be changed to "Complex_Main". UPL bags are translated to Isabelle/HOL multisets, which require the package "Isabelle-HOL.Multiset".

Imperative UPL language constructs like for- and while-statements, require Isabelle/HOL package for imperative programming "_".

Currently the traverser only pattern matches the type. Extending to for- and while-statements might additionally match on expression.

4.4 UPL AST

Implement syntax representation (IsabelleSyntax.scala) and case-matching (IsabelleCompiler.scala) for the whole UPL AST in the file Syntax.scala.

4.4.1 Types

Implemented: UnknownTypeContainer, UnknownType, EmptyType, UnitType, BoolType, StringType, NumberType, IntervalType, FunType, (SimpleFunType,) ProdType, CollectionType, ProofType (inside and outside locales)
Not yet implemented: Type, ClassType, AnyStructure, ExprsOver, BaseType, AnyType, ExceptionType,

4.4.2 Expressions

Implemented: Lambda, Application, Tuple, CollectionValue, Quantifier, Equality, UnitValue, BoolValue, NumberValue, RealValue, RatValue, IntValue, FloatValue, Real, ApproxReal, Rat, StringValue, String, BaseOperator, UndefinedValue,
Not yet implemented: This, Instance, ExprOver, Eval,
Projection, ListElem, Assert, BaseValue

4.4.3 Standard Programming Language Objects/Constructs

Implemented: VarDecl, Block, IfThenElse,
Not yet implemented: Assign, Match, MatchCase, For, While, Return

4.4.4 Operators

Implemented: Operator, PseudoOperator, GeneralInfixOperator, PseudoInfixOperator, PseudoPrefixOperator, PseudoPost, PseudoBind, PseudoCircum, InfixOperator, PrefixOperator, Associativity, NumberOperator, Arithmetic, Connective, And, Or, Not, Implies, Plus, Minus, Times, Divide, Minimum, Maximum, Power, UMinus, Less, LessEq, Greater, GreaterEq, CollectionOperator, Concat, In, Cons, Snoc, Equal, Inequal,
Not yet implemented: InferenceCallbacks,

4.4.5 Collections

CollectionType, CollectionKind, CollectionValue

4.4.6 Proof types and terms

UPL axioms and proofs (`|-`) can be inside theories or outside theories in the toplevel module. Treat differently, outside theory as `theorem` with associated `apply`proof. Inside theory as locale `fixes` and `assume`.

UPL proof types and terms. Compile UPL proof terms `???` to Isabelle/HOL `auto` or `sledgehammer`.

4.5 UPL modules & theories versus Isabelle theories & locales

4.5.1 UPL modules to Isabelle theories

`findPackages` finds Isabelle packages that need to be imported in the Isabelle theory for the UPL module.
Isabelle Tutorial: What is in `Main`?

Default return value "Main"

4.5.2 UPL theories to Isabelle locales

Working example from Isabelle locale tutorial pdf.

4.6 Tests

test20-29 collection types and values. option, list, set, bag, ulist.
test30-xx traverser