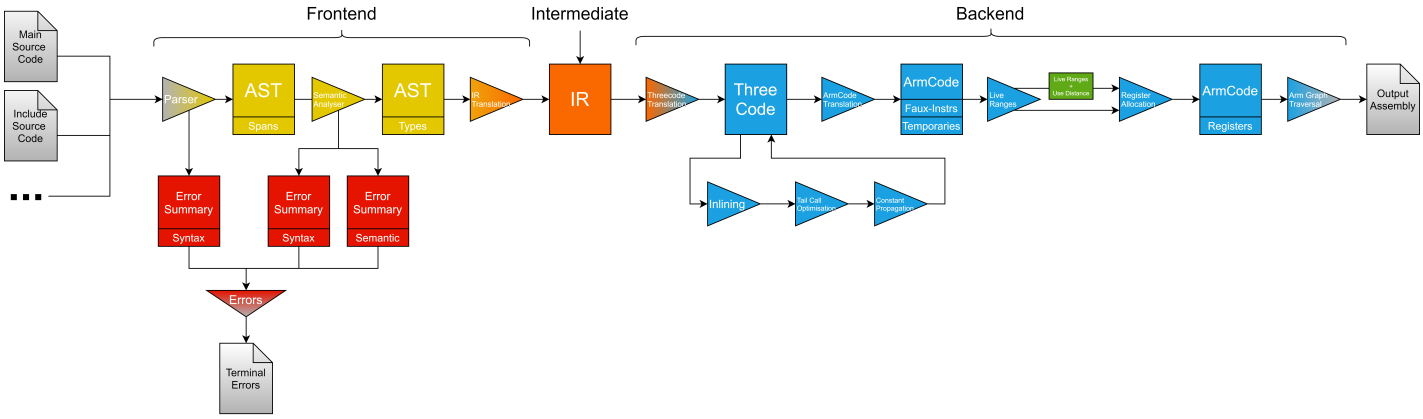
WACC

Jordan Hall, Bartłomiej Cieślar, Panayiotis Gavriil and Oliver Killane

14/03/22

Compiler Design



Our design attempts to maximize extensibility through modularity for every stage of the compiler. Each representation has has an interface and defined semantics, and hence we can easily swap out translations, optimisations, entire frontends or translation to architecture specific backends.

The compiler translates source code through 4 main representations; our **AST**, **IR**, **Three-Code** and **Arm-Code**.

AST

A tree based structure using an extended structure of the **WACC grammar** (allowing for more unary (fst, snd), binary operators (newpair) and types (full pair types)).

Each node in the three is wrapped by a generic. This allows the **AST** to be wrapped with different types of information, without having to change the structure.

Parsing of the input file (and included modules) produces an **AST** wrapped by spans (pointer to the section of relevant source code). Following a successful semantic analysis, the **AST** produced is wrapped by types (if one exists for a given node) to allow for easier translation to our **IR**.

IR

The intermediate representation comprises of a basic data section (for static, immutable data), and functions comprising of block graphs of basic statements.

Expressions can be nested, and come in three main types; **Pointer**, **Number** and **Boolean**.

The semantics of the language are also encoded here. All expressions can be reordered and short-circuited (hence language semantics for short-circuiting must be encoded in its translation to IR)

This **IR** was designed to be highly general, to allow for potentially any different language frontends, and as such supports full pointer arithmetic, function calls in expressions, void calls, and all possible control flows (as a block graph). It can also optionally define an integer overflow handler.

ThreeCode

Compiler Extensions

Project Management