Kipper Bericht 11. Juli

Allgemeines

Mithilfe des erstklassigen fachlichen Wissens von Prof. Dr. Mössenböck machen wir große Fortschritte in unserem Projekt. Bei seinen täglichen Meetings und Vorträgen zum internen Aufbau eines Kompilers konnten wir unter anderem unsere Symboltabelle massiv verbessern. Die Einführung eines Universums half uns, sich das Scopekonzept besser vorstellen zu können.

Unsere neuen Features bis jetzt

Luna Klatzer

Bis jetzt war der Hauptfokus auf die internen Kompilerstrukturen, welche unter anderem aus der Symboltabelle für Variablen, Funktionen und Types besteht. Diese neuen Strukturen erlauben es auch OOP Syntax & Code Generation zu implementieren. Zusätzlich bedeutet das, dass Type-Checks jetzt auch über richtige Referenzen laufen und man auch komplexe Typen miteinandere vergleichen kann, welche dann recursively auf einzelne Properties sich auch abgleichen können.

Lorenz Holzbauer

Es wurde zuerst String Multiplication implementiert. Dies diente zum Einarbeiten in den Compiler. Nun kann mithilfe des * Operators ein String n-mal wiederholt werden.

Des Weiteren wurden die Bitwise Operationen implementiert. Bei diesen musste die Order of Precedence richtig eingehalten werden, wofür ein paar Parser-Kniffe nötig waren.

Außerdem wurde der Lambda-Milestone fertiggestellt. Kipper kann nun einer Variable eine Lambda Expression assignen. Diese hat eine seperate Scope und wird Typüberprüft. Aktuell kann man Lambdas leider noch nicht ausführen da Generics, die Luna implementiert hat, gerade noch nicht gemerged sind. Dennoch können die generierten Lambda Expressions in der Target Language bereits aufgerufen werden.

Fabian Baitura

Zur Einarbeitung in den Compiler wurde die Do-While-Schleife implementiert. Danach wurden File-Scoped Pragmas hinzugefügt, um die Optimierung ein- oder auszuschalten. Außerdem wurden Interfaces sowie ihre Member (Properties und Methoden) implementiert.

Next Steps

Luna Klatzer

Nach der Fertigstellung der Symbol Table und Custom Type Logik werden mal alle AST Nodes mit der Library verbunden und die Logik miteinander vereint, sodass auch dann Objekte und Interfaces, die geparsed wurden auch jetzt korrekt repräsentiert werden und dann auch für komplexe Typchecks verwendet werden können. Zusätzlich wird dann auch die Logik für Typ-Inference implementiert, welche dann für Objekte den Typen automatisch auslesen und diesen dann als Return-Typ für die Expression zurückgeben sollen. Damit

kann man dann auch überprüfen ob der Variablentyp mit dem Objektliteral, welcher als Wert angegeben wird, zusammenpasst.

Lorenz Holzbauer

Sobald das neue Typsystem gemerged ist, wird die Typrepresentation im Kompilierten Code implementiert. Mithilfe dieser sollte der typeof-Operator keinen String mehr zurückgeben, sondern ein Runtime-Objekt an dem auf Typgleichheit geprüft werden kann. Aufgrund der Komplexität dieser Aufgabe ist dies der Hauptteil von Lorenz's Forschungsfrage.

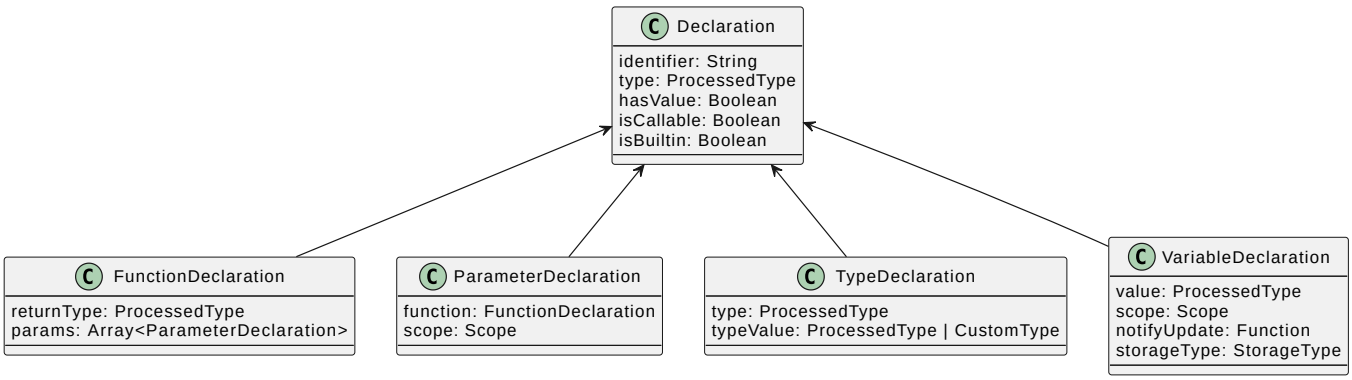
Des Weiteren wird das Try-Catch Statement implementiert werden, sobald der Exception Type fertiggestellt ist. Die nötige Vorarbeit zur Implementation im Parser und Lexer wurde schon erledigt, es fehlt also noch die korrekte interne Handhabung der Fehler.

Fabian Baitura

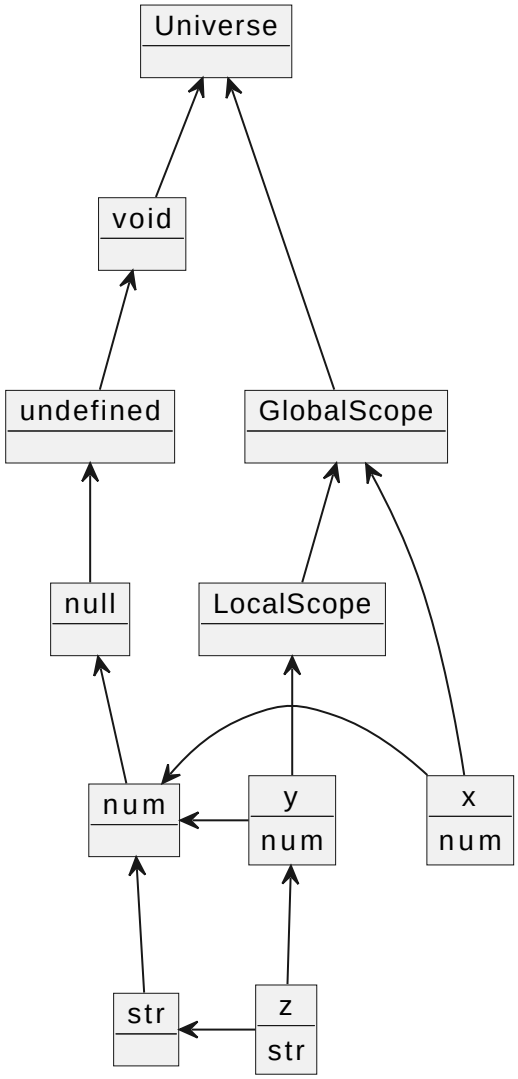
Zunächst werden Objekte in die Sprache hinzugefügt. Danach werden die semantischen Daten der Objekte genutzt, um interne Typrepräsentationen zu erstellen, die dann in die Zielsprache übersetzt werden können und mit denen Runtime-Typechecks durchgeführt werden können.

Deep Dive in unser Typsystem

Der Kern unsere Typsystems ist die Symbolliste. Diese wird mithilfe einer Map implementiert, und enthält Objekte mit folgenden Properties:



Wir haben uns für Vererbung entschieden, um die gemeinsamen Properties von Declaration zu abstrahieren. Wenn zwei Typen miteinander verglichen werden, wird in der Symboltabelle nachgeschaut, die Typen auf den selben Node zeigen. Dies wird rekursiv durchgeführt. Wir verwenden also Ducktyping, um Typen zu vergleichen. Dies ist notwendig, um eine die Kompabilität mit JavaScript zu gewährleisten.



Dies ist der Aufbau unserer Symboltabelle. Globale Typen befinden sich im Universum. Hier kann man schön sehen, wie jedes Objekt aus Unterobjekten bestehen kann, die jedoch alle auf den selben Referenztypen zeigen, um die Typen durch Referenzgleichheit zu vergleichen.