# Typechecking

## How to complete an exercise successfully?

Follow the rules as described in the Lecture!

### How to get additional information?

You are encouraged to discuss past and present exercise sheets with the teaching assistants. Either approach the teaching assistant during the exercise session, or visit us during the weekly office hours. We are also available through e-mail or on the StudIP forum. We try to reply as quickly as possible and in general, you should get a reply the next weekday, but we cannot guarantee this.

# Type Checker

Static type systems are some of the most successful and widely accepted formal tools to help write correct programs. The common interpretation of types in programming languages is a combination of the following:

a) A type is a set of values.

b) The values of a type share an implementation at run-time.

Further, type systems are often categorized in two dimensions: **static** vs. **dynamic**, and **strong** vs. **weak**. A **static** type system is executed at **compile time**, i.e., before the program is run, e.g., during compilation. A **dynamic** type system on the other hand is executed at **run time**, i.e., when the program is executed. One advantage of **static** typing is that it checks that a program is **type safe**, i.e., will not generate any type errors during execution, where a type error is a data flow between a type-wise incompatible value source and sink (`bool x = 'a';`). Once a program is proven to be type safe, certain checks can be eliminated, thereby gaining performance. The advantage of **dynamic** typing is often said to be that incorrectly typed can still be executed during development, and that programs do not have to be annotated with types, thereby being less obscured. However, modern programming languages (Haskell, OCaml,...) with static typing use type inference and therefore do not require many type annotations anymore. **Strong** versus **weak** generally refers to how easy it is to claim a value is of a different type or, similarly, how close the type of a source has to match the type of the sink. No generally accepted formal definitions of the terms exist.

## Task

Your task is to write a type checker for a simple, C-like language. Download the corresponding "CPP.cf" file from StudIP and read through the course book's website on type checking[1]. Do **not** work with your own grammar from the previous task!

**Hints:** Because the language requires variables and functions to be declared before use, we often know which type a source, e.g. an expression, should have. This knowledge greatly simplifies type checking as we do not need type variables and unification. However, there are operators where there is no functional dependency between the result type of the operator and the types of the operands.

*Prepare for interview:* Which operators are that? How could you deal with them?

[1] http://www1.digitalgrammars.com/ipl-book/assignments/assignment2/assignment2.html

## Deadlines of open tasks

a) Grammar: 26.05.2016, 13:14 (Upload to StudIP and demonstrate)

b) Type Checker: 09.06.2016, 13:14 (Upload to StudIP and demonstrate)