Summary of "Typed Arithmetic Expressions"

CHRISTOPH MENDE, NIKOLAJ-JENS SCHWARTZ

ACM Reference format:
Christoph Mende, Nikolaj-Jens Schwartz. 2017. Summary of "Typed Arithmetic Expressions". 1, 1, Article 1 (April 2017), 1 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The chapter "Typed Arithmetic Expressions" from the Book "Types and Programming languages"[1] is about proofs about a typed programming language. The programming language from the chapter "Untyped Arithmetic Expressions" is reused and extended with a type system that differantiates between natural numbers (Nat) and boolean values (Bool).

Nat is defined as 0, $succ\ t_1$, and $pred\ t_1$ if t_1 is a Nat. Bool is defined as $true$, $false$, and $iszero\ t_1$ if t_1 is a Nat. Additionally, the construct $if\ t_1\ then\ t_2\ else\ t_3$ is defined to return a value of the type of t_3 if t_1 is a Bool.

If a term is typeable, it has at most one type and there is always just one derivation tree for it.

The chapter "Safety = Progress + Preservation" is about proving that a term cannot reach a "stuck state" where the evaluation rules don't tell us, what to do next. This can be done using the progess and preservation theorems. Both Theorems together prove that a well-typed term cannot reach a stuck state.

Progress Theorem: For the case t=succ t1 t1:Nat there either is a t'1 or t1 is a value. If t1 is a nv t has to be too. If t1 -> t'1 by E-SUCC succ t1 -> succ t'1.

REFERENCES

[1] Benjamin C. Pierce. 2002. *Types and Programming Languages* (1st ed.). The MIT Press.

2017. XXXX-XXXX/2017/4-ART1 $15.00
DOI: 10.1145/nnnnnnn.nnnnnnn