Contents

1 ALGT

This document gives an overview of the *ALGT*-tool. First, a general overview of what the tool does is given and why it was developped. Second, a hands-on tutorial develops a simply typed lambda calculus. Thirdly, the reference manual gives an in-depth table of what options and flags can be used. Fourth, some concepts and algorithms are explained more thoroughly, together with properties they use. And as last, the 'dynamize' and 'gradualize' options are explained in depth, as these are what the master disertation is about.

This documentation is about version *0.1.13.1* (`Total Property Check: no types from builtins`), generated on 2017-2-9.

2 What is ALGT?

ALGT, of *Automated Language Generation Tool* [1] is a tool to formally specify any programming language. This is done by first declaring a syntax, using BNF, and then declaring semantics by introducing logical deduction rules, such

[1] Note the similarity of *ALGT* and *AGT* . The tool started it's life as *AGT*, based on the paper *Abstracting Gradual Typing.* When it became more general, another name and acronym was used. TODO reference to paper!

as evaluation or typing rules. Eventually, properties can be introduced which can be tested. Of course, these can be run.

The tool is kept as general as possible, so any language can be modelled.

3 Tutorial: developing a simple programming language

We will develop a programming language which can work with booleans and integers. Apart from doing basic arithmetic, we can apply anonymous functions on them.

Example programs, and the value they become, are

Table 1: Example programs and their outcome

Program	Evaluates to
`True`	`True`
`1`	`1`
`24 + 18`	`42`
‘If True Then 1 Else	2‘1
`1 + (2 + 3)`	`6`
‘(\ x : Int . x + 1	) 41‘42

The expression `(\x : Int . x + 1)` is a *lambda expression*. This is an anonymous function, taking one argument - named x- of type `Int`. When applied (e.g. `(\x : Int . x + 1) 41`, the expression right of the `.` is returned, with the variable `x` substituted by the argument, becoming `41 + 1`.

3.1 Setting up a .language

A language is declared inside a `.language` file [2]. Create `STFL.language`, and put a title in it:

‘

3.2 Declaring the syntax

For a full reference on syntax, see the reference manual on syntax

A program is nothing more then a string of a specific form. To describe strings of an arbitrary structure, *BNF* [3] can be used.

Consider all possible boolean forms: `True` and `False`.

[2]Actually, the extension doesn’t matter at all.

[3]Backus-Naur-form, as introduced by John Backus in the ALGOL60-report. TODO reference