A Quick Primer on TNT

Alexander Fraebel

April 14, 2021

1 Introduction

TNT crate for the Rust programming language that creates simple runtime validated proofs in Number Theory. This primer will explain the basics of using the crate.

The sections on Terms and Formulas explain how valid statements of TNT are formed. They mostly can be summarized by the Backus-Naur Form below. This precisely specifies Numbers, Variables, and Expressions and open Formulas. However quantified Formulas have additional restrictions that cannot be expressed as a context free grammar.

```
<num> ::= { "0" | "S" <num> }

<var> ::= { <lowercase_letter> | <var> "'" }

<arith> ::= { "+" | "*" }

<expr> ::= { <num> |
             <var> |
             "(" <expr> <arith> <expr> ")" |
             "S" expr }

<quant> ::= { "A" <var> ":" |
              "E" <var> ":" }

<logical> ::= { "&" | "|" | ">" }
```

```
<formula> ::= { <expr> "=" <expr> |
                "[" <formula> <logical> <formula> "]" |
                <quant> <formula> |
                "~" <formula> }
```

2 Terms

The grammar of TNT starts with three types: Variables, Numbers, and Expressions. Each of these implements the Term trait. The structs for these types all contain a String representing the underlying statement of TNT, the Variable struct also includes some additional information for speed of parsing. For brevity we will just refer to the contents of the struct. All Terms can be combined arithmetically. A few examples appear at the end of this section.

Numbers A valid Number consists of `0` preceded by `S` any number of times. These represent the natural numbers and the `S` symbol may be interpreted as the successor function. For convenience the Number type implements the methods .zero() and .one() which create `0` and `S0` automatically. The .random() method creates a random geometrically distributed Number.

Variables A valid Variable consists of any lowercase ASCII letter followed by any number of apostrophes. For instance `a` is a Variable as is `u''`. These stand for some unspecified natural number. The .random() method creates a Variable with a uniformly random letter and a geometrically distributed number of apostrophes.

Expression A valid Expression is a Number, Variable, an Expression preceded by `S`, or a arithmetic combination of two Expressions. The valid arithmetic operations are `+` and `*` and they must be parenthesized. Expressions may be arbitrarily complicated. For example `S(Sa'+0)`, `SSSSq`, `(SS0*(S(h'*S0)+j))`. The .random() method creates a random Expression of unbounded size.

3 Formulas

Formulas are well-formed formulas of the TNT language. The Formula enum has two variants: Simple and Complex. A Formula::Simple must be two Expressions with `=` between them such as `SS0=(a*b)`. In general a Formula can combine two Formulas with any of three binary logical operations, `&`, `|`, `>`. They may also be preceded with a tilde or with a quantification. Valid quantifications take the following form: start with `A` or `E`, then a Variable that does not appear within any quantification in the Formula, then `:`.

4 Deductions

Deductions are the centerpiece of the crate as they are required for building a full proof and applying all the allowed rules of inference.

add_axiom Takes a &Formula and adds it to the list if it is in the provided axioms.

specification Takes an index, a &Variable, and a &Term. Adds a new theorem that eliminates the universal quantification of the Variable then replaces every occurrence with the Term.

generalization Takes an index and a &Variable. Adds a new theorem that starts with a universal quantification of the Variable.

existence Takes an index and a &Variable. Adds a new theorem that starts with an existential quantification of the Variable.

successor Takes an index. Adds a new theorem that puts an `S` in front of each side of the equality.

predecessor Takes an index. Adds a new theorem that removes an `S` from each side of the equality.

interchange_ea Takes an index, a &Variable, and a position.

interchange_ae Takes an index, a &Variable, and a position.

symmetry Takes an index. Adds a new theorem that switches the two sides of the equality.

transitivity Takes two indices.

supposition Takes a Formula. Increases the depth by one step, creating a supposition block, and adds the Formula to the list.

implication Takes no arguments. Decreases the depth by one step then checks the previous supposition block and adds a theorem to the list that the first theorem implies the last theorem.

induction Takes a &Variable and two indices.