# Intuition v.0.3 — User Manual*

Marcin Benke and Jacek Chrząszcz and Aleksy Schubert and Maciej Zielenkiewicz
{ben,chrzaszcz,alx,maciekz}@mimuw.edu.pl

Institute of Informatics, University of Warsaw, Poland

December 2015

**Abstract**

We present here the information on how to use the Intuition typechecker and proover.

## 1 Preliminaries

Intuition consists of two independent programs: typechecker and prover. The typechecker takes a first-order formula in TPTP format with its potential proof written in own syntax and checks if the proof indeed proves the formula. The prover generates proofs of first-order formulae. The typechecker is writte in C programming language while the prover is written in Haskell.

This document is constructed as follows. We present the way the typechecker can be used in Section 2 and in Section 3.

## 2 How to Use the Typechecker?

The typechecker `intuitiontc` can be called in two ways

```
> ./intuitiontc -F <filename>
```

where `<filename>` is the name of a file in TPTP format with a FOF formula. In this case the program just parses the formula and exits. Alternatively the program can be invoked as

```
> ./intuitiontc -P <filename>
```

where `<filename>` is the name of a file in TPTP format with a FOF formula, and its proof in natural deduction form. In this case the program checks if the proof contained in the file is indeed a proof of the formula present there.

For example one can parse the following formula written in the TPTP format

*This work has been supported by the Polish NCN grant NCN 2012/07/B/ST6/01532.

```
fof(example_07,conjecture, (! [X] : vP(X)) => (? [X] : vP(X))).
```

that represents the formula

$$(\forall X.vP(X)) \implies (\exists X.vP(X)).$$

More examples of the formulas can be found in the .tptp files located in `tests/examples-formulas/` directory of the typechecker source code distribution file.

One can also check the correctness of a proof. For instance for the extended TPTP code

```
fof(example_04,conjecture,
    (vA | vB) => (vB | vA),
inference(
lambda(x,fof(vA | vB),
       case(var(x),
         y,fof(vA),
            in2(var(y),fof(vB | vA)),
         z,fof(vB),
            in1(var(z),fof(vB | vA))
)))).
```

Note that the portion of the expresion enclosed in `inference(...)` block represents the proof written in the format presented below in Section 2.1. The code above represents the assignment to check correctness of the judgement

$$\begin{aligned} \vdash \lambda x : vA \vee vB.\, \texttt{case}\; x\; \texttt{in} \\ & \texttt{left}\; y.in_{2,vB\vee vA}(y) \\ & \texttt{right}\; z.in_{1,vB\vee vA}(z) \quad : vA \vee vB \implies vB \vee vA \end{aligned}$$

More examples of formulas and their proofs can be fount in `tests/examples-proofs/` directory of the typechecker source code distribution file.

## 2.1 The Grammar of Proofs

The proof terms analysed by `intuitiontc` should be located in the formula annotations section of the `fof` TPTP expression. They are enclosed in `inference(...)` expression there. The grammar of proof terms themselves is as presented in Figure 1. The grammar uses TPTP productions for first-order formulas (fof_formula ) and atomic words, i.e. words that start with a small letter (atomic_word).

# 3 How to Use the Prover?

The prover program can be used interactively from the Haskell interactive environment. One has to construct a formula to generate proof for and then invoke the convertType

```
proof_term ::=
   exfalso ( proof_term, term_fof_formula )
  | lambda ( atomic_word, term_fof_formula, proof_term )
  | app ( proof_term, proof_term )
  | abstract ( atomic_word, term_fof_formula,
             atomic_word, term_fof_formula,
             proof_term, proof_term )
  | existI ( proof_term, proof_term, term_fof_formula )
  | case ( proof_term, case_term , case_term )
  | in1 ( proof_term, term_fof_formula )
  | in2 ( proof_term, term_fof_formula )
  | proj1 ( proof_term )
  | proj2 ( proof_term )
  | tuple ( proof_term, proof_term, term_fof_formula )
  | var ( atomic_word )
term_fof_formula ::= fof ( fof_formula )
case_term ::= atomic_word, term_fof_formula, proof_term
```

Figure 1: The grammar of proof terms.

function. For example to construct a proof for the formula

$$\forall x.(a \implies b \implies c) \implies a \implies c$$

one can in Haskell build the formula

```haskell
let
  tested = Tall "x" (Tall "_" a (Tall "_" b c))
                    (Tall "_" a c)
  a  = Tvar "a"
  b  = Tvar "b"
  c  = Tvar "c"
in
```

and then invoke convertType as follows

```haskell
  print $ convertType tested
```

More examples are available in the directory tests of the prover source code distribution file.