This tutorial explains how to use the Stdm[1] software for checking the validity of proofs.

1 Installation

Install the Glasgow Haskell Compiler (version 7.4.1):

- On a Fedora Linux machine, run the following command as root:
  `yum install ghc`
- On an Ubuntu-based Linux machine (Ubuntu, Xubuntu, Linux Mint), run:
  `sudo apt-get install ghc`
- For other systems, check the main site:
  `http://www.haskell.org/ghc/download`

Download the `Stdm.lhs` file from the course files section on Ninova:
`http://ninova.itu.edu.tr/Ders/142/Dosyalar`

Using the file manager, right-click on the `Stdm.lhs` file and open it with GHCi. If that is not available on your system, start a terminal emulator and type the command: `ghci Stdm.lhs`. You should see an output like this:

```
GHCi, version 7.4.1: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
[1 of 1] Compiling Stdm             ( Stdm.lhs, interpreted )
Ok, modules loaded: Stdm.
*Stdm>
```

Here, `*Stdm>` is a *prompt*, indicating that the system is expecting input.

You can exit the program by pressing `Ctrl-D`.

2 Syntax

The names of proposition variables are uppercase letters. The truth values are `TRUE` and `FALSE`. The propositional operators are: `Not`, `And`, `Or`, and `Imp`. These operators are written before their operands. For example, the proposition $P \wedge Q$ is written as `And P Q`.

A theorem consists of a list of assumptions and a conclusion. The assumptions are a comma-separated list of propositions written in square brackets. The conclusion is a proposition.

```
Theorem [assumption1, assumption2, ...] conclusion
```

Examples:

```
Theorem [P, Imp P Q] Q    -- P /\ P -> Q |-  Q
Theorem [P, Q] (And P Q)  -- P, Q |- P /\ Q
```

[1] http://www.dcs.gla.ac.uk/ jtod/discrete-mathematics/

3 Proofs

Proofs consist of assumptions and rules of inference. A proposition can be converted into a proof by assuming it. For example:

```
Assume P
Assume (And P Q)
```

Whether a proof really proves a theorem or not can be checked as follows:
`check_proof theorem proof`

```
*Stdm> let thPP = Theorem [P] P
*Stdm> let prAP = Assume P
*Stdm> check_proof thPP prAP
The proof is valid
```

Note the difference between a proposition and a proof. For example, `And P Q` is a proposition but `Assume (And P Q)` is a proof (or a stage in a proof). Assuming a proposition converts it into a proof but of course that proof is not necessarily valid for the theorem at hand. The following proof is invalid because it does not reach the conclusion of the theorem:

```
*Stdm> let thPQ = Theorem [P] Q
*Stdm> check_proof thPQ prAP
*** The proof is NOT valid ***
The proof does not match the sequent.
 .what is actually proved is:
      P  |-  P
```

And this proof is invalid because it assumes a proposition which was not supplied:

```
*Stdm> let prAQ = Assume Q
*Stdm> check_proof thPQ prAQ
*** The proof is NOT valid ***
The proof does not match the sequent.
 .what is actually proved is:
      Q  |-  Q
 .these assumptions are used but not part of the sequent:
      Q
```

The rules of inference take a number of already proven propositions and a conclusion and check whether the conclusion really follows from those propositions. They are explained below.

Identity.

```
*Stdm> let prID = ID prAP P
*Stdm> check_proof thPP prID
The proof is valid
```

Note that the identity rule has to be applied to a proof, not a proposition:

```
*Stdm> let prIDX = ID P P

<interactive>:3:16:
    Couldn’t match expected type ‘Proof’ with actual type ‘Prop’
    In the first argument of ‘ID’, namely ‘P’
    In the expression: ID P P
    In an equation for ‘prIDX’: prIDX = ID P P
```

Contradiction.

```
*Stdm> let thFP = Theorem [FALSE] P
*Stdm> let prCV1 = Assume FALSE
*Stdm> let prCV = CTR prCTR1 P
*Stdm> check_proof thFP prCV
The proof is valid
```

And here are some incorrect applications of the rule:

```
*Stdm> let thTP = Theorem [TRUE] P
*Stdm> let prCX1 = Assume TRUE
*Stdm> let prCX = CTR prCX1 P
*Stdm> check_proof thTP prCX
*** The proof is NOT valid ***
Reported errors:
 .CTR: the antecedent (TRUE) is not FALSE
*Stdm> let thEP = Theorem [] P
*Stdm> check_proof thEP prCV
*** The proof is NOT valid ***
The proof does not match the sequent.
 .what is actually proved is:
      FALSE  |-  P
 .these assumptions are used but not part of the sequent:
      FALSE
```

Or Introduction. This rule has two variants, `OrIL` and `OrIR`, which add new operands to the left and right operands, respectively.

```
*Stdm> let thOI = Theorem [P] (Or P Q)
*Stdm> let prOI1 = Assume P
*Stdm> let prOI = OrIL prOI1 (Or P Q)
*Stdm> check_proof thOI prOI
The proof is valid
```

Examples of incorrect applications:

```
*Stdm> let thOIR = Theorem [P] (Or Q P)
*Stdm> check_proof thOIR prOI
*** The proof is NOT valid ***
The proof does not match the sequent.
 .what is actually proved is:
      P  |-  Or P Q
```

```
*Stdm> let prOIR1 = Assume P
*Stdm> let prOIR = OrIR prOIR1 (Or P Q)
*Stdm> check_proof thOIR prOIR
*** The proof is NOT valid ***
Reported errors:
 .OrIL: the right term of OR conclusion (Or P Q) doesn't match the assumption (P)
```

And Elimination. This rule has two variants, `AndEL` and `AndER`, which eliminate to the left and right operands, respectively.

```
*Stdm> let thAE = Theorem [And P Q] P
*Stdm> let prAE1 = Assume (And P Q)
*Stdm> let prAE = AndEL prAE1 P
*Stdm> check_proof thAE prAE
The proof is valid
```

Example of incorrect application:

```
*Stdm> let prAER = AndER prAE1 P
*Stdm> check_proof thAE prAER
*** The proof is NOT valid ***
Reported errors:
 .AndER: the right term of And assumption (And P Q) doesn't match the conclusion (P)
```

Here is another invalid proof and a possible correction:

```
*Stdm> let thPQP = Theorem [P, Q] P
*Stdm> let prPQP1 = Assume P
*Stdm> let prPQP = AndEL prPQP1 P
*Stdm> check_proof thPQP prPQP
*** The proof is NOT valid ***
Reported errors:
 .AndEL: the assumption (P) is not an And expression
*Stdm> let prPQP2 = Assume (And P Q)
*Stdm> let prPQPX = AndEL prPQP2 P
*Stdm> check_proof thPQP prPQPX
*** The proof is NOT valid ***
The proof does not match the sequent.
 .what is actually proved is:
      And P Q  |-  P
 .these assumptions are used but not part of the sequent:
      And P Q
*Stdm> check_proof thPQP prAP
The proof is valid
notice: these assumptions are useless: Q
```

And Introduction.

```
*Stdm> let thAI = Theorem [P, Q] (And P Q)
*Stdm> let prAI1 = Assume P
*Stdm> let prAI2 = Assume Q
```

```
*Stdm> let prAI = AndI (prAI1, prAI2) (And P Q)
*Stdm> check_proof thAI prAI
The proof is valid
*Stdm> let prPQPV = AndEL prAI P
*Stdm> check_proof thPQP prPQPV
The proof is valid
```

Implication Elimination.

```
*Stdm> let thMP = Theorem [P, Imp P Q] Q
*Stdm> let prMP1 = Assume P
*Stdm> let prMP2 = Assume (Imp P Q)
*Stdm> let prMP = ImpE (prMP1, prMP2) Q
*Stdm> check_proof thMP prMP
The proof is valid
```

Implication Introduction. Note that the equivalence `Not P` ⇔ `Imp P FALSE` is already established.

```
*Stdm> let thMT = Theorem [Imp P Q, Not Q] (Not P)
*Stdm> let prMT1 = Assume P
*Stdm> let prMT2 = Assume (Imp P Q)
*Stdm> let prMT3 = ImpE (prMT1, prMT2) Q
*Stdm> let prMT4 = Assume (Not Q)
*Stdm> let prMT5 = ID prMT4 (Imp Q FALSE)
*Stdm> let prMT6 = ImpE (prMT3, prMT5) FALSE
*Stdm> let prMT7 = ImpI prMT6 (Imp P FALSE)
*Stdm> let prMT = ID prMT7 (Not P)
*Stdm> check_proof thMT prMT
The proof is valid
```

Note that all the provisional assumptions have to be discharged. So, the following proof is not valid:

```
*Stdm> let prMTX = CTR prMT6 (Not Q)
*Stdm> check_proof thMT prMTX
*** The proof is NOT valid ***
The proof does not match the sequent.
 .what is actually proved is:
      P, Imp P Q, Imp Q FALSE  |-  Not Q
 .these assumptions are used but not part of the sequent:
      P
```

Or Elimination. Example: proof of disjunctive syllogism

```
*Stdm> let thDS = Theorem [Or P Q, Not P] Q
*Stdm> let prDS1 = Assume (Or P Q)
*Stdm> let prDS2 = Assume (Not P)
*Stdm> let prDS3 = ID prDS2 (Imp P FALSE)
*Stdm> let prDS4a1 = Assume P
*Stdm> let prDS4a2 = ImpE (prDS4a1, prDS3) FALSE
```

```
*Stdm> let prDS4a = CTR prDS3a2 Q
*Stdm> let prDS4b1 = Assume Q
*Stdm> let prDS4b = ID prDS3b1 Q
*Stdm> let prDS = OrE (prDS1, prDS3a, prDS3b) Q
*Stdm> check_proof thDS prDS
The proof is valid
```

Example: proof of hypothetical syllogism

```
*Stdm> let thHS = Theorem [Imp P Q, Imp Q R] (Imp P R)
*Stdm> let prHS1 = Assume P
*Stdm> let prHS2 = Assume (Imp P Q)
*Stdm> let prHS3 = ImpE (prHS1, prHS2) Q
*Stdm> let prHS4 = Assume (Imp Q R)
*Stdm> let prHS5 = ImpE (prHS3, prHS4) R
*Stdm> let prHS = ImpI prHS5 (Imp P R)
*Stdm> check_proof thHS prHS
The proof is valid
```

Example: the use of a theorem

```
*Stdm> let thSpock = Theorem [P, Imp P Q, Imp Q R, Imp R S] S
*Stdm> let prSpock1 = Assume P
*Stdm> let prSpock1 = Assume (Imp P Q)
*Stdm> let prSpock2 = Assume (Imp Q R)
*Stdm> let prSpock3 = Use thHS [prSpock1, prSpock2] (Imp P R)
*Stdm> let prSpock4 = Assume (Imp R S)
*Stdm> let prSpock5 = Use thHS [prSpock3, prSpock4] (Imp P S)
*Stdm> let prSpock6 = Assume P
*Stdm> let prSpock = ImpE (prSpock6, prSpock5) S
*Stdm> check_proof thSpock prSpock
The proof is valid
```