Logic and lambda calculus in TEX

Victor Eijkhout

Notes for CS 594 – Fall 2004

- Expansion is very powerful
- Implement logic, numbers, lambda calculus

auxiliaries

```
\def\Ignore#1{}
\def\Identity#1{#1}
\def\First#1#2{#1}
\def\Second#1#2{#2}
```

Truth

```
\let\True=\First
\let\False=\Second
```

and logical operators:

```
\def\And#1#2{#1{#2}\False}
\def\Or#1#2{#1\True{#2}}
\def\Twiddle#1#2#3{#1{#3}{#2}}
\let\Not=\Twiddle
```

Truth test

True takes first of TF:

input : `\True`

output : T

False takes second of TF:

input : `\False`

output : F

Not true is false:

input : `\Not \True`

output : F

And truth table TrueTrue:

input : `\And \True \True`

output : T

And truth table TrueFalse:

input : `\And \True \False`

output : F

And truth table FalseTrue:

Lists

Definition

We implement a list as an operator with two arguments:

- If the list is not empty, the first argument is applied to the head, and the tail is evaluated;
- If the list is empty, the second argument is evaluated.

In other words

$$L\, a_1\, a_2 = \begin{cases} a_2 & \text{if } L = () \\ a_1(x)\, Y & \text{if } L = (x, Y) \end{cases}$$

Construction

```
\let\Nil=\Second
% \Cons <head> <tail> <arg1> <arg2>
\def\Cons#1#2#3#4{#3{#1}{#2}}
\def\Error{{ERROR}}
\def\Head#1{#1\First\Error}
\def\Tail#1{#1\Second\Error}
```

List examples

```
\def\Singleton#1{\Cons{#1}\Nil}
```

Head of a singleton:

```
input :  \Head {\Singleton \True }
output :  T
```

Head of a tail of a 2-elt list:

```
input :  \Head {\Tail {\Cons \True
{\Singleton \False }}}
output :  F
```

Visualization

```
\def\gobbletwo#1#2{}
\def\Transcribe#1{#1\TranscribeHT\gobbletwo}
\def\TranscribeHT#1#2{1\Transcribe{#2}}
```

Functions on lists

Given function f, initial argument e, and list X, then

$$\texttt{Apply}\, f\, e\, X \Rightarrow f\, x_1\, (f\, x_2\, (\ldots (f\, x_n\, e)\ldots))$$

```
% #1=function #2=initial arg #3=list
\def\ListApply#1#2#3{#3{\ListApplyp{#1}{#2}}{#2}}
\def\ListApplyp#1#2#3#4{#1{#3}{\ListApply{#1}{#2}{#4}}}
```

Concatenate

```
\def\Cat#1#2{\ListApply\Cons{#2}{#1}}
```

For example:

Cat two lists:

input : `\Transcribe {\Cat {\Singleton \Nil }{\Cons \Nil {\Singleton \Nil }}}`

output : *111*

Numbers

Adding one

```
\let\Zero\Nil
\def\AddOne#1{\Cons\Nil{#1}}
```

Examples:

Transcribe zero:

input : \Zero

output :

Transcribe one:

input : \AddOne \Zero

output : 1

Transcribe three:

input : \AddOne {\AddOne {\AddOne \Zero }}

output : 111

Subtracting one

```
\def\SubOne#1{#1\Second\Error}
```

Predecessor of two:

```
input :  \SubOne {\AddOne {\AddOne \Zero }}
output :  1
```

Arithmetic

Addition

```
\let\Add=\Cat
```

Adding numbers:

```
input :  \Add {\Three }{\Five }
output :  11111111
```

Comparison

```
\def\GreaterThan#1#2{#2{\GreaterThanp{#1}}\False}
\def\GreaterThanp#1#2#3{#1{\GreaterThanx{#3}}\True}
\def\GreaterThanx#1#2#3{\GreaterThan{#1}{#3}}
```

Greater (true result):

input : \GreaterThan \Two \Five

output : T

Greater (false result):

input : \GreaterThan \Three \Two

output : F

Greater (equal case):

input : \GreaterThan \Two \Two

output : F

Greater than zero:

input : \GreaterThan \Two \Zero

output : F

Use true result:

```
input :  \GreaterThan \Two \Five \Three \One
output :  111
```

Use false result:

```
input :  \GreaterThan \Three \Two \Three \One
output :  1
```

$3 < (5 - 1)$*:*

```
input :  \GreaterThan \Three {\Sub \One \Five
}
output :  F
```

$3 < (5 - 4)$*:*

```
input :  \GreaterThan \Three {\Sub \Four
\Five }
output :  T
```

Integers

```
% \StreamOp <operator> <initial value>
\def\StreamOp#1#2{\Cons{#2}{\StreamOp{#1}{#1{#2}}}}
\def\Integers{\StreamOp\AddOne\Zero}
```

Integers:
input : \Head {\Tail {\Integers }}
output : 1
Integers:
input : \Head {\Tail {\Tail {\Tail {\Tail {\Tail {\Integers }}}}}}
output : 11111