Automath

an 'interactive' theorem checker

Andreas Kochesser

Institute of Computer Science
University of Innsbruck

June 2013

History

The Automath Language

- AUT-68: around 1967 by N.G. de Bruijn († Feb. 2012)
- AUT-QE: quasi expressions
- AUT-Π (and AUT-ΔΛ)

Versions of Automath as Program

- first version 1968 by I. Zandleven
- current version (4.2) 1999 by F. Wiedijk

History

The Automath Language

- AUT-68: around 1967 by N.G. de Bruijn († Feb. 2012)
- AUT-QE: quasi expressions
- AUT-Π (and AUT-$\Delta\Lambda$)

Versions of Automath as Program

- first version 1968 by I. Zandleven
- current version (4.2) 1999 by F. Wiedijk

History

General Things About Automath

- the first *interactive* theorem proof checker
- type theoretical proof checker
- only one large 'book' available
  - *Grundlagen der Analysis* by E. Landau
  - translated by L.S. Jutting
- Influenced Coq, Lego, ...

Book and Line

Book

- a correct piece of Automath text
- . . . consisting of *lines*

Line

- *context indicator part*
- *identifier part*
- *definition part*
- *category part*
- some separators

Book and Line

Book

- a correct piece of Automath text
- . . . consisting of *lines*

Line

- *context indicator part*
- *identifier part*
- *definition part*
- *category part*
- some separators

Line 1

Definition Part

- the symbol —: variable definition
- the symbol **PN**: primitive notion
- an expression

Identifier Part

- an unused symbol
- called *variable* if the definition part is —
- otherwise called *constant*

Line 2 – Context indication

Example

	0	* w	:= —	; ⋆⋆⋆
	w	* a	:= ···	; ⋆⋆⋆
|	w	* x	:= —	; ⋆⋆⋆
|	x	* b	:= ···	; ⋆⋆⋆
|	w	* y	:= —	; ⋆⋆⋆
||	y	* z	:= —	; ⋆⋆⋆

- empty context symbol **0**
- indicator string (last line: 'w,y,z')

Line 2 – Context indication

Example

	0	* w	:= —	; ⋆⋆⋆
	w	* a	:= ···	; ⋆⋆⋆
|	w	* x	:= —	; ⋆⋆⋆
|	x	* b	:= ···	; ⋆⋆⋆
|	w	* y	:= —	; ⋆⋆⋆
||	y	* z	:= —	; ⋆⋆⋆

- empty context symbol **0**
- indicator string (last line: 'w,y,z')

Line 2 – Context indication

Example

	0	* w	:= —	; ⋆⋆⋆
	w	* a	:= ···	; ⋆⋆⋆
|	w	* x	:= —	; ⋆⋆⋆
|	x	* b	:= ···	; ⋆⋆⋆
|	w	* y	:= —	; ⋆⋆⋆
||	y	* z	:= —	; ⋆⋆⋆

- empty context symbol **0**
- indicator string (last line: 'w,y,z')

Line 3 – Categories

- every identifier must be assigned to a category (or **'type'**)
- rules:
  - we can only speak about categories and about objects belonging to categories.
  - any object can only belong to *one* type
- categories have the symbol **'type'**in the *category part*
- objects have an expression defining a category as *category part*

Example

```
0    * N    := —    ; 'type'
N    * x    := —    ; N
```

Line 3 – Categories

- every identifier must be assigned to a category (or **'type'**)
- rules:
  - we can only speak about categories and about objects belonging to categories.
  - any object can only belong to *one* type
- categories have the symbol **'type'**in the *category part*
- objects have an expression defining a category as *category part*

Example

```
0    * N    := —    ; 'type'
N    * x    := —    ; N
```

Line 3 – Categories

- every identifier must be assigned to a category (or **'type'**)
- rules:
  - we can only speak about categories and about objects belonging to categories.
  - any object can only belong to *one* type
- categories have the symbol **'type'**in the *category part*
- objects have an expression defining a category as *category part*

Example

```
0    * N      := —        ; 'type'
N    * x      := —        ; N
```

Line 3 – Categories

- every identifier must be assigned to a category (or **'type'**)
- rules:
  - we can only speak about categories and about objects belonging to categories.
  - any object can only belong to *one* type
- categories have the symbol **'type'**in the *category part*
- objects have an expression defining a category as *category part*

Example

```
0    * N        := PN          ; 'type'
0    * x        := PN          ; N
```

Line 4 – Expressions

- only contain already known *constants* or *variables*
- substitution, abstraction, application

Example

```
    0    * Prop      := PN              ; 'type'
    0    * con       := PN              ; Prop
|   0    * p         := —               ; Prop
||  p    * q         := —               ; Prop
||  q    * imp       := PN              ; Prop
|   p    * not       := imp(p,con)      ; Prop
```

Substitution

- changes context of an constant.
- replaces variables x_i for expressions Σ_i (definition/category part)
- category part must be compatible

Example

```
||  q    * imp      := PN              ; Prop
|   p    * not      := imp(p,con)      ; Prop
```

- hidden variable depencies
- depending lines must only be *admissible* (no objection to add them)

Substitution

- changes context of an constant.
- replaces variables x_i for expressions Σ_i (definition/category part)
- category part must be compatible

Example

```
||  q    * imp      := PN              ; Prop
|   p    * not      := imp(p,con)      ; Prop
```

- hidden variable depencies
- depending lines must only be *admissible* (no objection to add them)

Abbreviation

- goal: shorten written line length
- automatically insert missing context variables

Example

instead of:

```
|    p     * not          := imp(p,con)          ; Prop
```

we could write:

```
|    p     * not          := imp(con)            ; Prop
```

Abbreviation

- goal: shorten written line length
- automatically insert missing context variables

Example

instead of:

```
    p     * not        := imp(p,con)        ; Prop
```

we could write:

```
    p     * not        := imp(con)          ; Prop
```

Functional Abstraction 1

Example

```
0    * p        := —          ; Prop
p    * c        := Prop       ; 'type'
p    * d        := not(p)     ; Prop
```

- attaches a variable p to a constant d which has the type *Prop* (or c)
- this somehow resembles functional abstraction
- it would be nicer to write $\lambda_{x \in Prop} d(x)$

Functional Abstraction 1

Example

```
0    * p        := —          ; Prop
p    * c        := Prop       ; 'type'
p    * d        := not(p)     ; Prop
```

- attaches a variable p to a constant d which has the type *Prop* (or c)
- this somehow resembles functional abstraction
- it would be nicer to write $\lambda_{x \in Prop} d(x)$

Functional Abstraction 2

Example

0 * d := [x:Prop]not(x) ; [x:Prop]Prop

- here it is: our beloved λ
- the expression [x:Σ_1]Σ_2 means $\lambda_{x \in \Sigma_1} \Sigma_2$
- AUT-68 allows only abstractions on object/category level
- AUT-QE also allows abstractions on category/**'type'**level
- example: [x:*Category*]**'type'**

Functional Abstraction 2

Example

```
0    * d          := [x:Prop]not(x)    ; [x:Prop]Prop
```

- here it is: our beloved λ
- the expression [x:Σ_1]Σ_2 means $\lambda_{x \in \Sigma_1} \Sigma_2$
- AUT-68 allows only abstractions on object/category level
- AUT-QE also allows abstractions on category/'**type**'level
- example: [x:*Category*]'**type**'

Application

- opposite to application
- remove an abstraction

Example

```
0   * A     := —          ; 'type'
A   * B     := —          ; 'type'
B   * imp   := [x:A]B     ; [x:A]'type'
B   * a     := —          ; A
a   * app   := {a}imp     ; 'type'
```

Example Proof

Comparison – Automath / Mizar / HOL-Light

Automath

- no automation for proofs
- every little detail has to be written
- no kernel providing a base to build upon
- books start out with axiom definitions (unproved)
- but it is fast
- complete C source code below 200kB

Comparison – Automath / Mizar / HOL-Light

compared to Mizar and HOL-Light

- no tactics
- inductive types must be defined by hand
- far away from natural language
- minimalist syntax (maybe close to mathematical notation)
- no extensive libraries

What we have seen . . .

- history of Automath
- syntax
- how to define basic logic in Automath
- short overview what Automath does not have

Literature & Websites

For anyone who likes old papers

- `http://www.win.tue.nl/automath/`
  - N.G. de Bruijn
    AUT001: AUTOMATH, a language for mathematics (1968)
    AUT002: The mathematical language AUTOMATH (1968)
    XAUT001: Description of the language AUTOMATH (1967)
  - R.P. Nederpelt
    AUT019: AUTOMATH, a language for checking mathematics with a computer

About the latest Automath version

- `http://www.cs.ru.nl/~freek/aut`
  - Freek Wiedijk
    A new implementation of Automath
    A nice and accurate Checker for the Mathematical Language Automath

Last slide before the backup slides

Thank λou!

Questions?

Syntax in *aut*

- loose syntax, allows more styles to write a book
  - separators @, =, :
  - TYPE, PROP, PRIM
  - application: <x>f
  - style with category and definition swapped
  - leave out obvious context
  - define variables with [x:T]

AUT-Π and λ-AUT

AUT-Π

- extension of AUT-QE
- allows quantification of Objects/Abstractions

λ-AUT

- AUT in the style of the lambda calculus

- both have no implementation