Delimited overloading

Christophe Troestler

Institut de Mathématique
Université de Mons-Hainaut
Mons, Belgium

OCaml users meeting
February 4, 2009
Grenoble
Acknowledgments

This project was started thanks to the support of Jane Street Capital who sponsored

- Dany Maslowsky and
- Julie De Pril

during their OCaml summer projet 2008.
Outline

1. Standard Overloadings
2. Defining overloadings
3. Priority & associativity
4. Macros
5. Some technical details
Basic use

Float.(1 + x * f 4)
Float.(4 * u**2 / sqrt(abs alpha))
Hashtbl.(h.(key) <- x)

Liters, functions, and "constant constructions" substitution.
Better readability.

Big_int.(if x > 0 then x else 0)

Can use the usual comparison operators.

Int32.(4 + a.(Int.(1 + x)))

Embedded overloadings.
Basic use

- \( \text{Float.}(1 + x \times f\ 4) \)
- \( \text{Float.}(4 \times u^{2} / \sqrt{\text{abs\ alpha}}) \)
- \( \text{Hashtbl.}(h.(\text{key}) <- x) \)

- Literals, functions, and “constant constructions” substitution.
- Better readability.

- \( \text{Big\ int.}(\text{if}\ x > 0\ \text{then}\ x\ \text{else}\ 0) \)

- Can use the usual comparison operators.

- \( \text{Int32.}(4 + a.(\text{Int.}(1 + x))) \)

- Embedded overloadings.
Basic use

Float.(1 + x * f 4)
Float.(4 * u**2 / sqrt(abs alpha))
Hashtbl.(h.(key) <- x)

❖ Literals, functions, and “constant constructions” substitution.
❖ Better readability.

Big_int.(if x > 0 then x else 0)

❖ Can use the usual comparison operators.

Int32.(4 + a.(Int.(1 + x)))

❖ Embedded overloading.
Static checks & simple optimizations

Num.("12345678" + x)

Compile time check.
If one writes Num.("a12"), when compiling, the following error is issued

Parse error: The string "a12" does not represent a valid Num.
Preprocessing error on file foo.ml

Float.((x+1)**2)

Simple optimization. The whole expression is substituted by (binding introduced only if needed):

let tmp = x +. 1.0 in tmp *. tmp
Static checks & simple optimizations

\[\text{Num.}("12345678" + x)\]

Compile time check.
If one writes \text{Num.}("a12"), when compiling, the following error is issued

Parse error: The string "a12" does not represent a valid Num.
Preprocessing error on file foo.ml

\[\text{Float.}((x+1)**2)\]

Simple optimization. The whole expression is substituted by (binding introduced only if needed):

\[
\text{let } tmp = x +. 1.0 \text{ in } tmp \ast. tmp
\]
Complex numbers

Complex.(let z = 3 + 2 I in \texttt{sin}(z * z))

- "I" notation.
- Let binding are allowed.
- Complex functions like \texttt{sin}, \texttt{cos},... are inlined.

For example,

Complex.((2 + 3 I) * f x)

is turned into

let \texttt{tmp} = f x in
\{ Complex.re = (\texttt{tmp}.Complex.re *. 2.0) -. (\texttt{tmp}.Complex.im *. 3.0); \\
Complex.im = (\texttt{tmp}.Complex.re *. 3.0) +. (\texttt{tmp}.Complex.im *. 2.0); \}
Complex numbers

Complex.(let z = 3 + 2 I in sin(z * z))

“i” notation.
Let binding are allowed.
Complex functions like sin, cos, ... are inlined.
For example,

Complex.((2 + 3 I) * f x)

is turned into

let tmp = f x in
{ Complex.re = (tmp.Complex.re *. 2.0) -. (tmp.Complex.im *. 3.0);
  Complex.im = (tmp.Complex.re *. 3.0) +. (tmp.Complex.im *. 2.0); }
Summary

pa_do.cmo provides overloadings for

Int    Float    Hashtbl
Int32   Complex  String
Int64
Nativeint

pa_do_nums.cmo provides overloadings for

Num
Ratio
Big_int

Requires nums.cmo to be loaded by camlp4 for static checks.
Summary

pa_do.cmo provides overloadings for

Int     Float     Hashtbl
Int32   Complex   String
Int64   
Nativeint

pa_do_nums.cmo provides overloadings for

Num
Ratio
Big_int

Requires nums.cmo to be loaded by camlp4 for static checks.
Outline

1. Standard Overloadings
2. Defining overloadings
3. Priority & associativity
4. Macros
5. Some technical details
## Concrete syntax v.s. API

<table>
<thead>
<tr>
<th>Concrete syntax</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>In the source file</td>
<td>In a separate file</td>
</tr>
<tr>
<td>Must be repeated in each file</td>
<td>Can be bundled with a library</td>
</tr>
<tr>
<td>No possibility of overloading general expressions</td>
<td>One can overload general expressions and perform some optimizations</td>
</tr>
</tbody>
</table>
## Concrete syntax v.s. API

<table>
<thead>
<tr>
<th>Concrete syntax</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>In the source file</td>
<td>In a separate file</td>
</tr>
<tr>
<td>Must be repeated in each file</td>
<td>Can be bundled with a library</td>
</tr>
<tr>
<td>No possibility of overloading general expressions</td>
<td>One can overload general expressions and perform some optimizations</td>
</tr>
</tbody>
</table>

**Demo**
"Set of overloadings"

Constructions that can be overloaded:

- literals
  - ‘x
  - a.(i)
  - a.(i) <- x
  - a.[i]
  - a.[i] <- x
  - a.{i}
  - a.{i} <- x
  - a := x
  - a <- x
  - a.p

- functions & operators overloadings
- general substitutions
module Foo :

sig
    type t
    val of_int :  int -> t
    val compare :  t -> t -> int
    val add :  t -> t -> t
    val mul :  t -> t -> t
end

Overloading with the concrete syntax:

int literals: OVERLOAD_INT Foo (of_int)
comparison: OVERLOAD_COMPARISON Foo (compare)
functions: OVERLOAD Foo ( (+ ) -> add; ( * ) -> mul )
module Foo :
  sig
    type t
    val of_int : int -> t
    val compare : t -> t -> int
    val add : t -> t -> t
    val mul : t -> t -> t
  end

Overloading with the concrete syntax:

- int literals: OVERLOAD_INT Foo (of_int)
- comparison: OVERLOAD_COMPARISON Foo (compare)
- functions:
  OVERLOAD Foo (( + ) -> add; ( * ) -> mul)
module Foo :
  sig
    type t
    val of_int :  int -> t
    val compare :  t -> t -> int
    val add :  t -> t -> t
    val mul :  t -> t -> t
  end

Overloading with the concrete syntax:

- int literals: OVERLOAD_INT Foo (of_int)
- comparison: OVERLOAD_COMPARISON Foo (compare)

functions:
OVERLOAD Foo (( + ) -> add; ( * ) -> mul)
How to overload one’s own module (1/2)

module Foo :
  sig
    type t
    val of_int : int -> t
    val compare : t -> t -> int
    val add : t -> t -> t
    val mul : t -> t -> t
  end

Overloading with the concrete syntax:

- int literals: OVERLOAD_INT Foo (of_int)
- comparison: OVERLOAD_COMPARISON Foo (compare)
- functions:
  OVERLOAD Foo ( ( + ) -> add; ( * ) -> mul )
How to overload one’s own module (2/2)

Remarks

- If Foo implements all standard functions add, sub, mul, div and neg (unary negation):
  
  ![OVERLOAD_ARITHMETIC Foo]

- If a new module is implemented:

```ocaml
module Special_foo : sig
  include Foo
  val sub : t -> t -> t
end

OVERLOAD Special_foo inherit Foo
```

```ocaml
OVERLOAD Special_foo ( ( - ) -> sub )
```
Remarks

- If Foo implements all standard functions add, sub, mul, div and neg (unary negation):
  
  `OVERLOAD_ARITHMETIC Foo`.

- If a new module is implemented:

```ocaml
module Special_foo :
sig
  include Foo
  val sub : t -> t -> t
end
```

`OVERLOAD Special_foo inherit Foo`

`OVERLOAD Special_foo ( ( - ) -> sub )`
Some more examples

Basin of attraction of Newton’s method for $z^3 = 1$.

```ocaml
Complex.(  
  let z = ref z0 in  
  for i = Int.(1) to niter do  
    z := (2 * !z + 1 / !z**2) / 3  
  done;  
  if abs(!z - root0) <= r then Some color0  
  ... )
```

Let $D = 1.7$. If $p = [p_0; \ldots; p_n]$ represents the polynomial $\sum_{i=0}^{n} p_i z^i$, its norm is (here) defined by $\|p\| := \sum_{i=0}^{n} |p_i| D^i$.

```ocaml
let domain = Interval.(17 / 10)  
let norm p = Interval.(List.fold_right (fun c n ->  
  abs c + domain * n) p 0)
```
Some more examples

Basin of attraction of Newton’s method for $z^3 = 1$.

Complex.(  
    let z = ref z0 in  
    for i = Int.(1) to niter do  
        z := (2 * !z + 1 / !z**2) / 3  
        done;  
    if abs(!z - root0) <= r then Some color0  
    ... )

Let $D = 1.7$. If $p = [p_0; ... ; p_n]$ represents the polynomial $\sum_{i=0}^{n} p_i z^i$, its norm is (here) defined by $\|p\| := \sum_{i=0}^{n} |p_i| D^i$

let domain = Interval.(17 / 10)  
let norm p = Interval.(List.fold_right (fun c n ->  
    abs c + domain * n) p 0)
Outline

1. Standard Overloadings
2. Defining overloadings
3. Priority & associativity
4. Macros
5. Some technical details
Priority and associativity of operators

pa_infix.cmo

Concrete syntax:

INFIX ( %+ ) RIGHTA HIGHER (+)
INFIX ( ^* ) LEVEL (+)
PREFIX ( /+/ )
POSTFIX ( /// ) LEVEL ( ! )

API: treat $a = b \ |> \ c$ as $a = (b \ |> \ c)$ and replace $x \ |> \ f$ by $f \ x$:

open Pa_infix
module L = Level
let l = L.binary (L.Higher L.comparison) \sim\ assoc:L.LeftA in
let expr x y _loc = <:expr< $y$ $x$ >> in
infix "\|>" \sim\ expr l
Outline

1. Standard Overloadings
2. Defining overloadings
3. Priority & associativity
4. Macros
5. Some technical details
Collaboration with Delimited Overloading

\[
\text{DEFINE NEWTON}(M, x) = M.( (2 * x + 1 / x^{**2}) / 3 )
\]

Use it as

\[
\begin{align*}
\text{NEWTON}(\text{Float}, r) \\
\text{NEWTON}(\text{Complex}, z)
\end{align*}
\]

- Poor man defunctorizer;
- Contrarily to functors, requalifies constants.
Better error reporting

DEFINE A(x) =
    let s = ref 0.0 in
    for i = 1 to x do
        s := !s + float i
    done;
    s

let () = print_float(A(100))

With the standard macros:

File "...", line 8, characters 21-27:
This expression has type float but is here used with type int
Better error reporting

DEFINE A(x) =
    let s = ref 0.0 in
    for i = 1 to x do
        s := !s + float i
    done;
    s

let () = print_float(A(100))

With Delimited Overloading macros:

File "...", line 8, characters 21-27:
Expanding of the macro "A" at the previous location yields the error:
File "...", line 4, characters 11-13:
This expression has type float but is here used with type int
Outline

1. Standard Overloadings
2. Defining overloadings
3. Priority & associativity
4. Macros
5. Some technical details
   - Optimization for complex operators
   - Embedded overloading
   - General substitution of expressions
Optimization for complex operators

Complex.((2 + 3 I) * f x)

1. Classify subexpressions according to

```ocaml
type t =
| Zero
| Re of Ast.expr
| Im of Ast.expr
| Cplx of Ast.expr * Ast.expr
| Unknown of Ast.expr
```

2. Specialize complex functions, introducing bindings as needed.
Embedded overloading (1/3)

\[ X.(e) \mapsto \sigma_X(e) \]

where \( \sigma_X : expression \rightarrow expression \)

Problem encountered:

\[
\text{Int32.}(a.(\text{Int.}(0)) \leftarrow 7 + x) \\
\downarrow \text{apply } \sigma_{\text{Int}}; \text{ here } \sigma_{\text{Int}}(0) = 0 \\
\text{Int32.}(a.(0) \leftarrow 7 + x) \\
\downarrow \text{apply } \sigma_{\text{Int32}} \\
a.(01) \leftarrow \text{Int32.add } 71 \ x
\]

Problem!

Protection of already overloaded expressions
Embedded overloading (1/3)

\[ X.(e) \mapsto \sigma_X(e) \]

where \( \sigma_X : expression \rightarrow expression \)

Problem encountered:

\[ \text{Int32.}(a.(\text{Int.}(0)) \leftarrow 7 + x) \]

apply \( \sigma_{\text{Int}} \); here \( \sigma_{\text{Int}}(0) = 0 \)

\[ \text{Int32.}(a.(0) \leftarrow 7 + x) \]

apply \( \sigma_{\text{Int32}} \)

\[ a.(0l) \leftarrow \text{Int32.add 7l x} \]

Problem!

Protection of already overloaded expressions
Embedded overloading (1/3)

\[ X.(e) \mapsto \sigma_X(e) \]

where \( \sigma_X : expression \rightarrow expression \)

Problem encountered:

\[
\begin{align*}
\text{Int32.}(a.(\text{Int.}(0)) & \leftarrow 7 + x) \\
\downarrow & \text{apply } \sigma_{\text{Int}}; \text{ here } \sigma_{\text{Int}}(0) = 0 \\
\text{Int32.}(a.(0) & \leftarrow 7 + x) \\
\downarrow & \text{apply } \sigma_{\text{Int32}} \\
a.(0l) & \leftarrow \text{Int32.add} 7l x
\end{align*}
\]

Problem!

Protection of already overloaded expressions
### Embedded overloading (1/3)

\[ X.(e) \mapsto \sigma_X(e) \]

where \( \sigma_X : \text{expression} \rightarrow \text{expression} \)

#### Problem encountered:

\[
\begin{align*}
\text{Int32.}(a.&(\text{Int.}(0)) \leftarrow 7 + x) \\
\text{apply } \sigma_{\text{Int}}; \text{ here } \sigma_{\text{Int}}(0) = 0 \\
\text{Int32.}(a.(0) \leftarrow 7 + x) \\
\text{apply } \sigma_{\text{Int32}} \\
a.(01) \leftarrow \text{Int32.add } 71 \ x
\end{align*}
\]

Problem!

Protection of already overloaded expressions
Embedded overloading (1/3)

\[ X.(e) \leftrightarrow \sigma_X(e) \]

where \( \sigma_X : \text{expression} \rightarrow \text{expression} \)

**Problem encountered:**

\[
\begin{align*}
\text{Int32.(a.(\text{Int.(0)}) \leftarrow 7 + x)} & \quad \downarrow \text{apply } \sigma_{\text{Int}}; \text{ here } \sigma_{\text{Int}}(0) = 0 \\
\text{Int32.(a.(0) \leftarrow 7 + x)} & \quad \downarrow \text{apply } \sigma_{\text{Int32}} \\
a.(01) & \leftarrow \text{Int32.add 71 x}
\end{align*}
\]

Problem!

Protection of already overloaded expressions
Embedded overloading (1/3)

\[ X.(e) \mapsto \sigma_X(e) \]

where \( \sigma_X : expression \rightarrow expression \)

Problem encountered:

\[
\text{Int32.}(a.\left(\text{Int.}(0)\right) \leftarrow 7 + x) \\
\downarrow \text{apply } \sigma_{\text{Int}}; \text{ here } \sigma_{\text{Int}}(0) = 0 \\
\text{Int32.}(a.(0) \leftarrow 7 + x) \\
\downarrow \text{apply } \sigma_{\text{Int32}} \\
a.(0l) \leftarrow \text{Int32.add} \ 7l \ x
\]

Problem!

Protection of already overloaded expressions
Embedded overloading (1/3)

\[ X.(e) \mapsto \sigma_X(e) \]

where \( \sigma_X : \text{expression} \to \text{expression} \)

Problem encountered:

\[
\begin{align*}
\text{Int32.}(a.(\text{Int.}(0)) \leftarrow 7 + x) & \quad \text{apply } \sigma_{\text{Int}}; \text{ here } \sigma_{\text{Int}}(0) = 0 \\
\text{Int32.}(a.(0) \leftarrow 7 + x) & \quad \text{apply } \sigma_{\text{Int32}} \\
a.(0l) \leftarrow \text{Int32.add 7l x} & \quad \text{Problem!}
\end{align*}
\]
Embedded overloading (1/3)

\[ X.(e) \mapsto \sigma_X(e) \]

where \( \sigma_X : \text{expression} \rightarrow \text{expression} \)

Problem encountered:

\[
\begin{align*}
\text{Int32.}(a.(\text{Int.}(0)) & \leftarrow 7 + x) \\
\downarrow \text{apply } \sigma_{\text{Int}}; \text{ here } \sigma_{\text{Int}}(0) = 0 \\
\text{Int32.}(a.(0) & \leftarrow 7 + x) \\
\downarrow \text{apply } \sigma_{\text{Int32}} \\
a.(0\text{l}) & \leftarrow \text{Int32.add} 7\text{l} \ x
\end{align*}
\]

Problem!
Embedded overloading (1/3)

\[ X \rightarrow (e) \mapsto \sigma_X(e) \]

where \( \sigma_X : expression \rightarrow expression \)

Problem encountered:

\[
\begin{align*}
\text{Int32.}(a.(\text{Int.}(0)) \leftarrow 7 + x) & \\
& \downarrow \text{apply } \sigma_{\text{Int}}; \text{ here } \sigma_{\text{Int}}(0) = 0 \\
\text{Int32.}(a.(0) \leftarrow 7 + x) & \\
& \downarrow \text{apply } \sigma_{\text{Int32}} \\
a.(01) & \leftarrow \text{Int32.add 7}1 \ x
\end{align*}
\]

Problem!

Protection of already overloaded expressions
Embedded overloading (2/3)

Requirements:

- $\sigma_{\text{Int}}(0)$ must be a valid expression;
- it must not change the meaning of the program nor its performance;
- locations must not be affected (for correct error reporting).

Solution:

$$X.(e) \mapsto p(\sigma_X(e))$$

where $p$ is an undeclared function name!

- $p$ is removed by the surrounding overloading ⇒ global flag to know whether to insert $\pi$.
- external $p : \alpha \rightarrow \alpha = "\%identity"$ forbid some optimizations to take place!
Embedded overloading (2/3)

Requirements:

- $\sigma_{\text{Int}}(0)$ must be a valid expression;
- it must not change the meaning of the program nor its performance;
- locations must not be affected (for correct error reporting).

Solution:

$$X.(e) \mapsto p(\sigma_X(e))$$

where $p$ is an undeclared function name!

- $p$ is removed by the surrounding overloading $\Rightarrow$ global flag to know whether to insert $\pi$.
- external $p : \alpha \rightarrow \alpha = "\%identity"$ forbid some optimizations to take place!
Embedded overloading (2/3)

Requirements:

- $\sigma_{\text{Int}}(0)$ must be a valid expression;
- it must not change the meaning of the program nor its performance;
- locations must not be affected (for correct error reporting).

Solution:

$$X.(e) \mapsto p(\sigma_X(e))$$

where $p$ is an undeclared function name!

$p$ is removed by the surrounding overloading $\Rightarrow$ global flag to know whether to insert $\pi$.

external $p : \alpha \rightarrow \alpha = "$identity" forbid some optimizations to take place!
Embedded overloading (2/3)

Requirements:

- $\sigma_{\text{Int}}(0)$ must be a valid expression;
- it must not change the meaning of the program nor its performance;
- locations must not be affected (for correct error reporting).

Solution:

$$X.(e) \mapsto p(\sigma_X(e))$$

where $p$ is an undeclared function name!

- $p$ is removed by the surrounding overloading $\Rightarrow$ global flag to know whether to insert $\pi$.
- external $p : \alpha \rightarrow \alpha = "\%identity"$ forbid some optimizations to take place!
Embedded overloading (3/3)

\[
\text{Int32.}(a.(\text{Int.}(0)) \leftarrow 7 + x)
\]

\[
\downarrow \text{Int.}(0) = p(\sigma_{\text{Int}(0)}) = p(0)
\]

\[
\text{Int32.}(a.(p(0)) \leftarrow 7 + x)
\]

\[
\downarrow \text{apply } \sigma_{\text{Int32}}
\]

\[
p(a.(0) \leftarrow \text{Int32.add } 7l x)
\]

OK!
**Embedded overloading (3/3)**

\[
\text{Int32.}(a.\text{(Int.(0))} \leftarrow 7 + x)
\]

\[
\downarrow
\text{Int.}(0) = p(\sigma_{\text{Int}(0)}) = p(0)
\]

\[
\text{Int32.}(a.(p(0)) \leftarrow 7 + x)
\]

\[
\downarrow \text{apply } \sigma_{\text{Int32}}
\]

\[
p(a.(0) \leftarrow \text{Int32.add 7l x})
\]

OK!
Embedded overloading (3/3)

\[
\text{Int32.}(a. (\text{Int.}(0)) \leftarrow 7 + x) \\
\downarrow \text{Int.}(0) = p(\sigma_{\text{Int}}(0)) = p(0) \\
\text{Int32.}(a. (p(0)) \leftarrow 7 + x) \\
\downarrow \text{apply } \sigma_{\text{Int32}} \\
p(a.(0) \leftarrow \text{Int32.add} \ 7 \ 1 \ x)
\]

OK!
Embedded overloading (3/3)

\[
\begin{align*}
\text{Int32.}(a.(\text{Int}.(0)) & \leftarrow 7 + x) \\
\quad \downarrow \text{Int}.(0) &= p(\sigma_{\text{Int}}(0)) = p(0) \\
\text{Int32.}(a.(p(0)) & \leftarrow 7 + x) \\
\quad \downarrow \text{apply } \sigma_{\text{Int32}} \\
p(a.(0) & \leftarrow \text{Int32.add} \ 7 \ 1 \ x)
\end{align*}
\]

\text{OK!}
Embedded overloading (3/3)

\[
\text{Int32.}(a.(\text{Int}_.(0)) \leftarrow 7 + x) \\
\quad \downarrow \text{Int}_.(0) = p(\sigma_{\text{Int}_.}(0)) = p(0) \\
\text{Int32.}(a.(p(0)) \leftarrow 7 + x) \\
\quad \downarrow \text{apply } \sigma_{\text{Int32}} \\
\quad p(a.(0) \leftarrow \text{Int32.add } 7l x) \\
\quad \text{OK!}
\]
Embedded overloading (3/3)

\[
\begin{align*}
\operatorname{Int32}.(a.\left(\operatorname{Int}.(0)\right) & \leftarrow 7 + x) \\
\downarrow & \\
\operatorname{Int}.(0) & = p(\sigma_{\operatorname{Int}(0)}) = p(0) \\
\operatorname{Int32}.(a.\left(p(0)\right) & \leftarrow 7 + x) \\
\downarrow & \\
\text{apply } \sigma_{\operatorname{Int32}} & \\
\operatorname{p}(a.(0) & \leftarrow \operatorname{Int32}.\operatorname{add} 7l x)
\end{align*}
\]

OK!
Embedded overloading (3/3)

\[
\text{Int32.} (a. (\text{Int.} (0)) \leftarrow 7 + x)
\]

\[
\downarrow \text{Int.} (0) = p(\sigma_{\text{Int}} (0)) = p(0)
\]

\[
\text{Int32.} (a. (p(0)) \leftarrow 7 + x)
\]

\[
\downarrow \text{apply } \sigma_{\text{Int32}}
\]

\[
p (a. (0) \leftarrow \text{Int32.add 7l} \ x)
\]

\text{OK!}
General substitution of expressions $X.(e)$

Original expression $e$ ➔ $\pi$ ➔ basic overloadings ➔ $\text{Ast.map#expr}$ ➔ Substituted expression
General substitution of expressions $X.(e)$

Original expression $e$ → $\pi$ → basic overloadings → $\text{Ast.map#expr}$ → substituted expression
General substitution of expressions $X.(e)$
General substitution of expressions $X.(e)$

Original expression $e$ -> substituted expression

- $\pi$
- $f_1$
- basic overloading
- $\text{Ast.map#expr}$

Some technical details
General substitution of expressions $X.(e)$

Original expression $e$ → $\pi$ → $f_2$ → $f_1$ → basic overloadings → Ast.map#expr → Substituted expression
General substitution of expressions $X.(e)$

Original expression $e$

Substituted expression

Basic overloadings

Ast.map#expr

Credits Outline Standard Overloadings Defining overloadings Priority & associativity Macros

Some technical details
Thank you for your attention.