Using Camlp4 for Presenting Dynamic Mathematics on the Web

DynaMoW, an OCaml Language Extension for the Run-Time Generation of Mathematical Contents and their Presentation on the Web

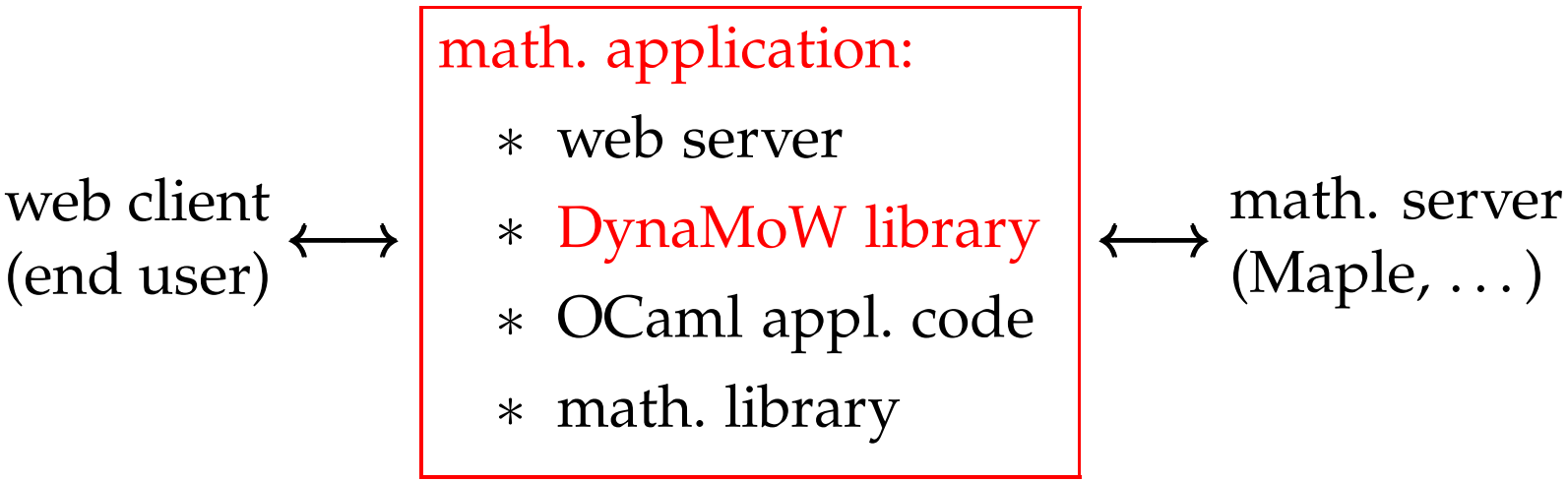
(An experience report)

Frédéric Chyzak and Alexis Darrasse

ICFP'11 (Tokyo), September 20, 2011

DynaMoW: The Project and its Constraints

- Interactive and programmable mathematical web sites
- Incremental mathematical calculations at click-time



- Metaprogramming: guest (very) different from host
- As computer-algebra-system agnostic as possible
- Not a goal: a computer-algebra system online

DynaMoW Syntax: Quotations and Antiquotations

Symbolic result, converted to LaTeX, put into a paragraph

```
let res = <:symb< symbolic expression >> in
  <:par< some text <:imath< some latex $(symb:res) >> >>
```

Using OCaml values in symbolic computations

```
let n = 23 and s = "foo" in
  <:symb< f($(int:n), $(str:s)) >>
```

Symbolic objects cast to OCaml types

```
let n = 23 + <:int< symbolic expression >> in ...
if <:bool< symbolic expression >> then ... else ...
<:unit< f := symbolic expression >>
```

DynaMoW Syntax: Three Keywords

Declare a computer-algebra system and a type for its values

```
use_cas MyCAS
type_symb my_cas = ... MyCAS.t symb.t
```

Change treatment of subsequent quotations by Camlp4.

Declare a web service (expanded as a module)

```
let_service MyService
  (v : τ) ...
  (x : σ = default_value) ... :
  returned_document_type * returned_symbolic_type =
  ...
```

Function-like declaration in view of function-like calls.

Toy Application: A Trigonometric Service

```
use_cas Maple
type_symb maple = unit Maple.t DynaMoW.symb
module DSC = DynaMoW.Services.Content

let_service Trigo (n : int = 2) : DSC.sec_entities * unit =
  let m_expr = <:symb< cos($(int:n) * x) >> in
  let s = Expand.obj (m_expr, ()) in
  let par_intro =
    <:par<The expansion of <:imath<\cos nx>> when <:imath<n = $(int:n)>>
      contains <:imath<$(int:s)>> terms:>>
  and par_math =
    DSC.inline_service (Expand.descr m_expr None) in
  let par = DSC.(@@@) par_intro par_math in
  (DSC.section <:text<My Trigonometric Service>> par, ())

let_service Expand (expr : maple) : DSC.sec_entities * int =
  let s = <:symb< expand($(expr)) >> in
  (<:par<<:dmath<$(symb:expr) = $(symb:s)>>>>, <:int<nops($(s)) >>)
```

Comments on the Implementation

- Mathematical application is a FastCGI
- Computer-algebra systems as plugins
- Automatic mathematical presentation

- Automatic serialisation and deserialisation of arguments
- Parameters with default values imply interaction
- Function-like calls, but not intended for recursion
- One computer-algebra session per service instance
- Statelessness for asynchronous and parallel (re-)computations of fragments of a web page

- $\simeq$ 4.2 kloc of OCaml ($\simeq$ 0.5 kloc for quotations/keywords)
- $\simeq$ 0.5 kloc for a glue to a computer-algebra system
- $\simeq$ 0.1 kloc of JavaScript (Ajax)

Developing Mathematical Applications

- Two specialised mathematical encyclopedias:
  - DDMF (“special functions” of mathematics)
  - ECS (combinatorial structures)
- Ancestors: Maple-centric + some Perl (CGI)

- Instrumenting some part of our existing Maple library
- OCaml list of symbolic refs vs Ref to a symbolic list
- Debugging: calling the computer-algebra system from an OCaml toplevel
- Special effort on modularity for efficiency (statelessness)

Conclusions

- Download: `http://ddmf.msr-inria.inria.fr/DynaMoW/`
- Non-reflexive quotations are important!
- Types in antiquotations: unwanted need to annotate

```
let a = 3 in <:symb< f($(int:a)) >>
```

- Services declarations: functors are not enough

- Mathematical rendering: MathJax still too slow?
- (Computer-algebra) code extraction?

- Education, another type of applications?