Extension points for OCaml

Leo White

September 24, 2013

Camlp4

A preprocessor for OCaml:

- Supports arbitrary extensions to the language's grammar
- Powerful system that has enabled some great extensions for OCaml

Camlp4 Examples

Type-conv extensions:

```
type t = {
  foo: int with default(42);
  bar: float
} with sexp

let x = sexp_of_t { foo = 3; bar = 5.0 }
```

Camlp4 Examples

ULex:

```
let lex = lexer
  [ 'a'-'z' 1000-1500 ] 65 -> Foo
| [ 'a' - 'z' ]* -> Bar
```

Camlp4 Examples

COW:

```
let world = "world" in
let html =
  <:html< <h1>Hello $str:world$!</h1> >>
```

Camlp4

Camlp4 is very complex. This complexity comes from its support for arbitrary extensions to the OCaml syntax. However, most of its common uses don't require this ability.

Can we provide a simpler alternative?

AST transformers

Rather than write extensions as preprocessors, write them as AST transformers.

The extension is given an AST by the compiler, transforms it, and returns the new AST to the compiler.

This can be done in OCaml 4.01 using the `-ppx` option.

Extension points: Attributes

sexplib

```
type t = {
  foo: int; [@default 42]
  bar: float
} [@@sexp]
```

Extension points: Attributes

$$
\begin{array}{lcl}
\mathit{attribute} & ::= & [@\ \mathit{id}\ \mathit{struct\_item}] \\
& | & [@\ \mathit{id}\ ?\ \mathit{pattern}] \\
& | & [@\ \mathit{id}\ :\ \mathit{type\_expr}]
\end{array}
$$

Extension points: Extensions

sedlex

```
[%lexer
  match foo with
    (Range('a','z')|Range(1000, 1500)), 65 -> Foo
  | Star (Range('a','z')) -> Bar
]
```

Alternative string syntax

ulex

```
[%lexer
   match foo with
      {| [ 'a'-'z' 1000-1500 ] 65 |} -> Foo
   | {| [ 'a' - 'z' ]* |} -> Bar
]
```

COW

```
let world = "world" in
let html = [%html {|<h1>Hello $str:world$!</h1>|}]
```

Short-cut syntax

```
match%lexer foo with
   {| [ 'a'-'z' 1000-1500 ] 65 |} -> Foo
 | {| [ 'a' - 'z' ]* |} -> Bar
```

Short-cut syntax

Without short-cut

```
[%lwt let x = f () in
[%lwt let y = g () in
  expr ]]
```

With short-cut

```
let%lwt x = f () in
let%lwt y = g () in
  expr
```

Status

These additional syntaxes have been implemented by Alain Frisch and merged onto trunk.

Thoughts and feedback very welcome (wg-camlp4@lists.ocaml.org).