OCaml - 2013

A New Implementation of Formats based on GADTs

Benoît Vaugon
Ensta-ParisTech

Introduction

Formats in OCaml

- Used for Printing and Scanning.
- Stdlib modules: Printf, Scanf and Format.
- Advantage: separate structure from data.

Basic Examples

- `Printf.printf "%d/%d/%d" m d y`
- `Scanf.scanf "%d/%d/%d" (fun m d y -> (m, d, y))`

Advanced Examples

- `Printf.sprintf "%#-0*.3X" 6 42` $(\rightarrow$ `"0x02A␣"`$)$
- `Printf.printf "today=%a%!" print_date (m, d, y)`
- `Printf.printf "version=%(%d%d%s%)" "%d.%d(%S)" 4 0 "alpha"`
- `Format.printf "@[<hov␣2>%d@,%d@]" 42 43`
- `Scanf.sscanf "OCaml|2013" "%s@|%[0-9]%!" callback`
- `Scanf.sscanf "today=09/24/2013" "today=%r" scan_date callback`

Summary

Format Types

The OCaml type-checker:

```
match expression, expected_type with
| String_literal s, ty when equiv ty format6_ty -> [...]
| [...]
```

Inferred type:

```
type ('a, 'b, 'c, 'd, 'e, 'f) format6
```

'a: the type of the parameters of the format

'b: the type of the first argument given to [%a] and [%t] printing functions

'c: the type of the result of the [%a] and [%t] functions

'd: the result type for the scanf-style functions,

'e: the type of the receiver function for the scanf-style functions

'f: the result type for the printf-style function

Format Types (Examples)

Standard library functions:

```
Printf.printf :
   ('a, out_channel, unit, unit, unit, unit) format6 -> 'a
Scanf.scanf :
   ('a, in_channel, 'c, 'd, 'a -> 'f, 'f) format6 -> 'd
```

Inferred types of formats:

```
format_of_string "%d" :
   (int -> 'a, 'b, 'c, 'd, 'e, 'f) format6
format_of_string "%a" :
   (('b -> 'x -> 'c) -> 'x -> 'f, 'b, 'c, 'e, 'e, 'f) format6
format_of_string "%r" :
   ('a -> 'f, 'b, 'c, ('b -> 'a) -> 'e, 'e, 'f) format6
```

The Current Implementation

Type-checking:

- Parsing of the literal string
- Manual inference of the `format6` type parameters

Memory representation:

- At runtime, formats are represented by **strings**

Printing function steps:

1. Parse the format and count parameters
2. Accumulate parameters
3. Extract and patch sub-formats
4. Call the C sprintf function on each sub-formats

Scanning function steps:

1. Count the number of `"%r"` in the format
2. Accumulate the readers and the callback function
3. Scan the channel and accumulate parameters
4. Call the callback function all at once

Problems

Safety

- Multiple format parsers ($\Rightarrow$ risk of incompatibilities)
  ex: `Printf.printf "%1.1s" "hello"`

Problems

Safety

- Multiple format parsers ($\Rightarrow$ risk of incompatibilities)
  ex: `Printf.printf "%1.1s" "hello"`
  $\rightarrow$ `Invalid_argument "Printf:␣bad␣conversion␣%s..."`

Problems
Safety

- Multiple format parsers ($\Rightarrow$ risk of incompatibilities)
  ex: `Printf.printf "%1.1s" "hello"`
  $\rightarrow$ `Invalid_argument "Printf:␣bad␣conversion␣%s..."`
- Weakness of the type-checker:
  ex: `Printf.sprintf "%2.+f" 3.14`

Problems

Safety

- Multiple format parsers (⇒ risk of incompatibilities)
  ex: `Printf.printf "%1.1s" "hello"`
  → `Invalid_argument "Printf:␣bad␣conversion␣%s..."`
- Weakness of the type-checker:
  ex: `Printf.sprintf "%2.+f" 3.14`
  → `"%2.+0f"`

Problems

Safety

- Multiple format parsers (⇒ risk of incompatibilities)
  ex: `Printf.printf "%1.1s" "hello"`
  → `Invalid_argument "Printf:␣bad␣conversion␣%s..."`
- Weakness of the type-checker:
  ex: `Printf.sprintf "%2.+f" 3.14`
  → `"%2.+0f"`
- Use of `Obj.magic` in printing and scanning functions
  ex: `Format.printf "@%d%s" 42 "hello"`

Problems

Safety

- Multiple format parsers (⇒ risk of incompatibilities)
  ex: `Printf.printf "%1.1s" "hello"`
  → `Invalid_argument "Printf:␣bad␣conversion␣%s..."`
- Weakness of the type-checker:
  ex: `Printf.sprintf "%2.+f" 3.14`
  → `"%2.+0f"`
- Use of `Obj.magic` in printing and scanning functions
  ex: `Format.printf "@%d%s" 42 "hello"`
  → Segmentation fault

Problems

Safety

- Multiple format parsers ($\Rightarrow$ risk of incompatibilities)
  ex: `Printf.printf "%1.1s" "hello"`
  $\rightarrow$ `Invalid_argument "Printf:␣bad␣conversion␣%s..."`
- Weakness of the type-checker:
  ex: `Printf.sprintf "%2.+f" 3.14`
  $\rightarrow$ `"%2.+0f"`
- Use of `Obj.magic` in printing and scanning functions
  ex: `Format.printf "@%d%s" 42 "hello"`
  $\rightarrow$ Segmentation fault

Speed

- Parsing of the format at runtime
- Re-parsing by C (slow) printing functions
- Lots of memory allocations

Memory allocations

- Sub-formats extractions (substrings)
- Lots of partial calls $\Rightarrow$ closure allocations
- Ex: `Printf.printf "Hello␣world\n"` $\rightsquigarrow$ allocates 738 bytes
  `Printf.printf "%s|%d\n" "OCaml" 2013` $\rightsquigarrow$ allocates 1512 bytes

The New Implementation

The Idea:

- Implement the `format6` type by a GADT
  - ⇒ The `format6` type is now concrete (not predefined)

Examples

- `"Hello"` ⇝ `String_literal ("Hello", End_of_format)`
- `"n␣=␣%02d\n%!"` ⇝

```
String_literal ("n␣=␣",
  Int (Conv_d, Lit_pad (Zero_pad, 2), No_prec,
    Char_literal ('\n',
      Flush End_of_format)))
```

Remark:

- Formats are **statically** allocated
  (not dynamically multiple times allocated)

The New Implementation

```
type ('a, 'b, 'c, 'd, 'e, 'f) format6 =
| Flush : ('a, 'b, 'c, 'd, 'e, 'f) format6 ->
    ('a, 'b, 'c, 'd, 'e, 'f) format6

| String_literal : string * ('a, 'b, 'c, 'd, 'e, 'f) format6 ->
    ('a, 'b, 'c, 'd, 'e, 'f) format6

| Bool : ('a, 'b, 'c, 'd, 'e, 'f) format6 ->
    (bool -> 'a, 'b, 'c, 'd, 'e, 'f) format6

| Int : conv * ('x, 'y) pad * ('y, int -> 'a) prec *
  ('a, 'b, 'c, 'd, 'e, 'f) format6 ->
    ('x, 'b, 'c, 'd, 'e, 'f) format6

| Alpha : ('a, 'b, 'c, 'd, 'e, 'f) format6 ->
    (('b -> 'x -> 'c) -> 'x -> 'a, 'b, 'c, 'd, 'e, 'f) format6

| [...]

| End_of_format : ('f, 'b, 'c, 'e, 'e, 'f) format6
```

Issues

Evaluation order

- For **printing** functions:
  - Accumulate parameters before printing
- For **scanning** functions:
  - Accumulate readers and the callback function before scanning

The `string_of_format` function

- In the current implementation: implemented by `%identity`
- In the new implementation, 2 possibilities:
  - Re-generate the string from the `GADT`
  - Implement formats by a tuple `(GADT, "original␣string")`

Only one format parser

- for the standard library and the OCaml type-checker

```
type ('b, 'c, 'e, 'f) fmt_ebb = Fmt_EBB :
  ('a, 'b, 'c, 'd, 'e, 'f) CamlinternalFormatBasics.fmt ->
    ('b, 'c, 'e, 'f) fmt_ebb
val fmt_ebb_of_string : string -> ('b, 'c, 'e, 'f) fmt_ebb
val type_format : ('x, 'b, 'c, 't, 'u, 'v) format6 ->
  ('a, 'b, 'c, 'd, 'e, 'f) fmtty ->
  ('a, 'b, 'c, 'd, 'e, 'f) format6
```

Issues

The "%(..%r..%)" construction

- Need to include a proof term of the number of "%r"

```
type ('d1, 'e1, 'd2, 'e2) reader_nb_unifier =
| Zero_reader :
    ('d1, 'd1, 'd2, 'd2) reader_nb_unifier
| Succ_reader :
    ('d1, 'e1, 'd2, 'e2) reader_nb_unifier ->
    ('x -> 'd1, 'e1, 'x -> 'd2, 'e2) reader_nb_unifier

type format6 =
| [...]
| Format_subst :
    int option * ('d1, 'q1, 'd2, 'q2) reader_nb_unifier *
    ('x, 'b, 'c, 'd1, 'q1, 'u) fmtty *
    ('u, 'b, 'c, 'q1, 'e1, 'f) format6 ->
    (('x, 'b, 'c, 'd2, 'q2, 'u) format6 -> 'x,
      'b, 'c, 'd1, 'e1, 'f) format6
```

Performances

P1 : `printf "Hello␣world\n"`

P2 : `printf "%s" "Hello␣world\n"`

P3 : `printf "%s|%d\n" "OCaml" 2013`

P4 : `printf "%d|%d|%d|%d|%d|%d|%d|%d" 1 2 3 4 5 6 7 8`

S1 : `sscanf "Hello␣world\n" "Hello␣world\n" ()`

S2 : `sscanf "Hello␣world\n" "%s" (fun _ -> ())`

S3 : `sscanf "OCaml|2013" "%s@|%[0-9]" (fun _ _ -> ())`

S4 : `sscanf "1|2|3|4|5|6|7|8" "%d|%d|%d|%d|%d|%d|%d|%d" ignore8`

Test	Allocs (bytes)	Time (ns)
P1	732 $\rightsquigarrow$ 24	230 $\rightsquigarrow$ 55
P2	1048 $\rightsquigarrow$ 96	230 $\rightsquigarrow$ 62
P3	1512 $\rightsquigarrow$ 264	590 $\rightsquigarrow$ 280
P4	5112 $\rightsquigarrow$ 1128	2700 $\rightsquigarrow$ 1600
S1	1976 $\rightsquigarrow$ 1392	380 $\rightsquigarrow$ 320
S2	2296 $\rightsquigarrow$ 1448	330 $\rightsquigarrow$ 200
S3	3632 $\rightsquigarrow$ 1768	830 $\rightsquigarrow$ 430
S4	4304 $\rightsquigarrow$ 2600	1480 $\rightsquigarrow$ 1070

Conclusion

Choices / Other Implementations

- With GADTs
  - The `string_of_format` problem
  - Optimisations on small formats to remove all allocations
  - ...
- Without GADTs
  - Ex: implement formats by a 4-tuple:
    - Printing function for channel
    - Printing function for buffer
    - Scanning function
    - Original format string

Improvements

- Safety
  - Only one format parser
  - No use of `Obj.magic`
- Performances