ML Pattern-Matching, Recursion, and Rewriting: from FoCaLiZe to Dedukti

Raphaël Cauderlier, Catherine Dubois

October 26, 2016

Deduction Modulo

- Proof search technique in first-order logic
  - Implicit computation (rewriting)
  - Explicit reasoning
- Confluent terminating rewriting $\Rightarrow$ decidable proof checking
- Concise and efficient

Example

$$
\begin{array}{lcl}
x + 0 & \hookrightarrow & x \\
0 + y & \hookrightarrow & y \\
\mathsf{S}(x) + y & \hookrightarrow & \mathsf{S}(x + y) \\
x + \mathsf{S}(y) & \hookrightarrow & \mathsf{S}(x + y)
\end{array}
$$

$$
\begin{array}{lcl}
\mathsf{twice}(0) & \hookrightarrow & 0 \\
\mathsf{twice}(\mathsf{S}(x)) & \hookrightarrow & \mathsf{S}(\mathsf{S}(\mathsf{twice}(x)))
\end{array}
$$

$$\vdash \mathsf{twice}(0) = 0 + 0$$

Example

$$
\begin{array}{lcl}
x + 0 & \hookrightarrow & x \\
0 + y & \hookrightarrow & y \\
\mathsf{S}(x) + y & \hookrightarrow & \mathsf{S}(x + y) \\
x + \mathsf{S}(y) & \hookrightarrow & \mathsf{S}(x + y)
\end{array}
$$

$$
\begin{array}{lcl}
\mathsf{twice}(0) & \hookrightarrow & 0 \\
\mathsf{twice}(\mathsf{S}(x)) & \hookrightarrow & \mathsf{S}(\mathsf{S}(\mathsf{twice}(x)))
\end{array}
$$

$$
\frac{}{\vdash \quad 0 = 0}\ \mathsf{Refl}(0)
$$

Example

$$
\begin{array}{lcl}
x + 0 & \hookrightarrow & x \\
0 + y & \hookrightarrow & y \\
\mathsf{S}(x) + y & \hookrightarrow & \mathsf{S}(x + y) \\
x + \mathsf{S}(y) & \hookrightarrow & \mathsf{S}(x + y)
\end{array}
$$

$$
\begin{array}{lcl}
\mathsf{twice}(0) & \hookrightarrow & 0 \\
\mathsf{twice}(\mathsf{S}(x)) & \hookrightarrow & \mathsf{S}(\mathsf{S}(\mathsf{twice}(x)))
\end{array}
$$

$$\mathsf{twice}(x) = x + x \vdash \qquad \mathsf{twice}(\mathsf{S}(x)) = \mathsf{S}(x) + \mathsf{S}(x)$$

Example

$$
\begin{array}{lcl}
x + 0 & \hookrightarrow & x \\
0 + y & \hookrightarrow & y \\
\mathsf{S}(x) + y & \hookrightarrow & \mathsf{S}(x + y) \\
x + \mathsf{S}(y) & \hookrightarrow & \mathsf{S}(x + y) \\
 & & \\
\mathsf{twice}(0) & \hookrightarrow & 0 \\
\mathsf{twice}(\mathsf{S}(x)) & \hookrightarrow & \mathsf{S}(\mathsf{S}(\mathsf{twice}(x)))
\end{array}
$$

$$
\dfrac{\dfrac{\dfrac{}{\mathsf{twice}(x) = x + x \vdash \mathsf{twice}(x) = x + x}\,\text{Axiom}}{\mathsf{twice}(x) = x + x \vdash \mathsf{S}(\mathsf{twice}(x)) = \mathsf{S}(x + x)}\,\text{Congr(S)}}{\mathsf{twice}(x) = x + x \vdash \mathsf{S}(\mathsf{S}(\mathsf{twice}(x))) = \mathsf{S}(\mathsf{S}(x + x))}\,\text{Congr(S)}
$$

Zenon Modulo

- Zenon: a theorem prover proving theorems
  - First-order logic
  - Tableaux method
  - Coq proofs
- Zenon Modulo
  - = Zenon
  - + Typing
  - + Deduction Modulo

Zenon Modulo

- Zenon: a theorem prover proving theorems
  - First-order logic
  - Tableaux method
  - Coq proofs
- Zenon Modulo
  - = Zenon
  - + Typing
  - + Deduction Modulo
  - + Dedukti proofs

Dedukti

- Dependent types + rewriting

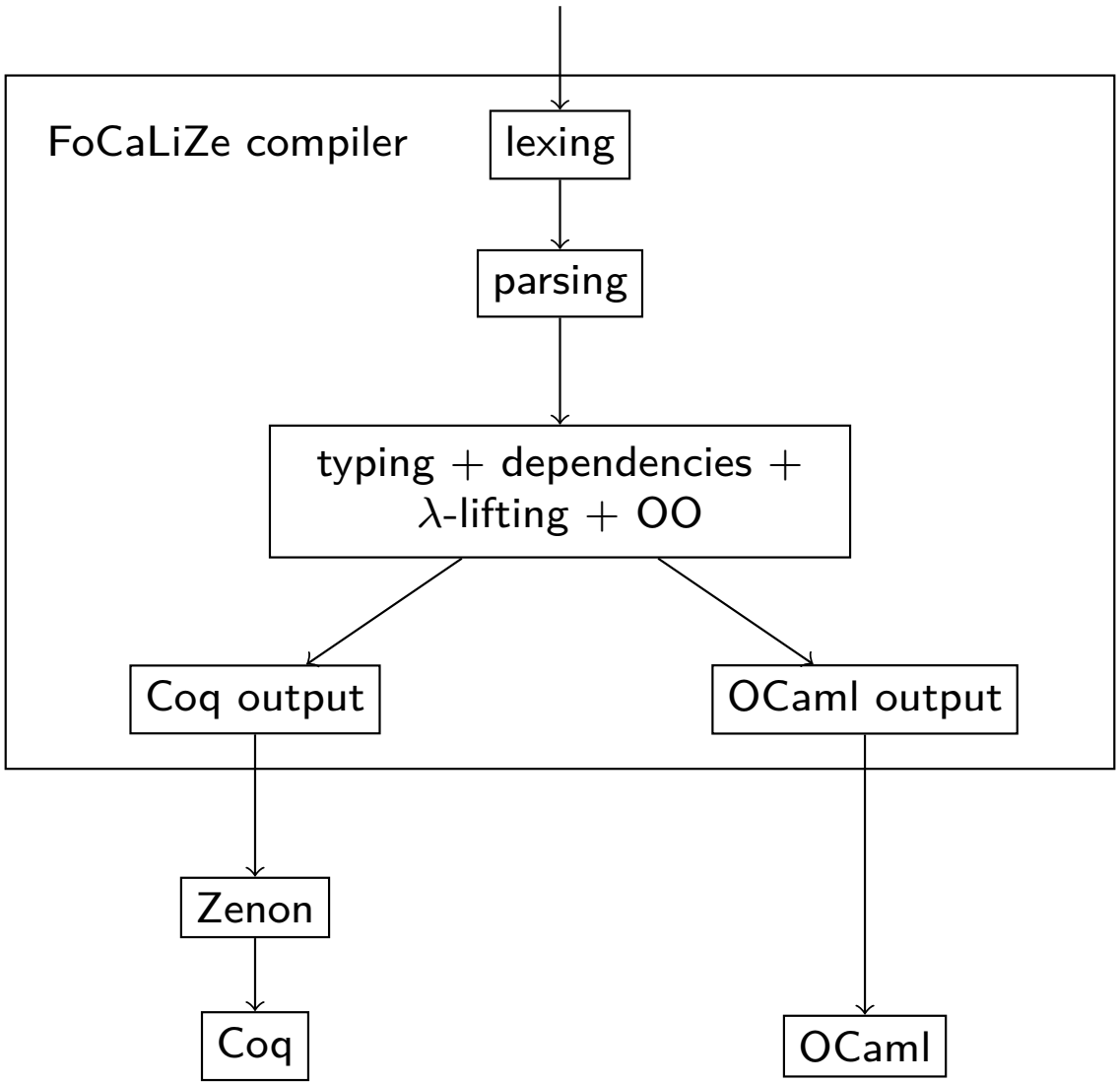
- A proof checker for Deduction modulo
- Logical Framework (HOL, Matita)
- Shallow translations (ς-calculus)

FoCaLiZe, a Formal IDE

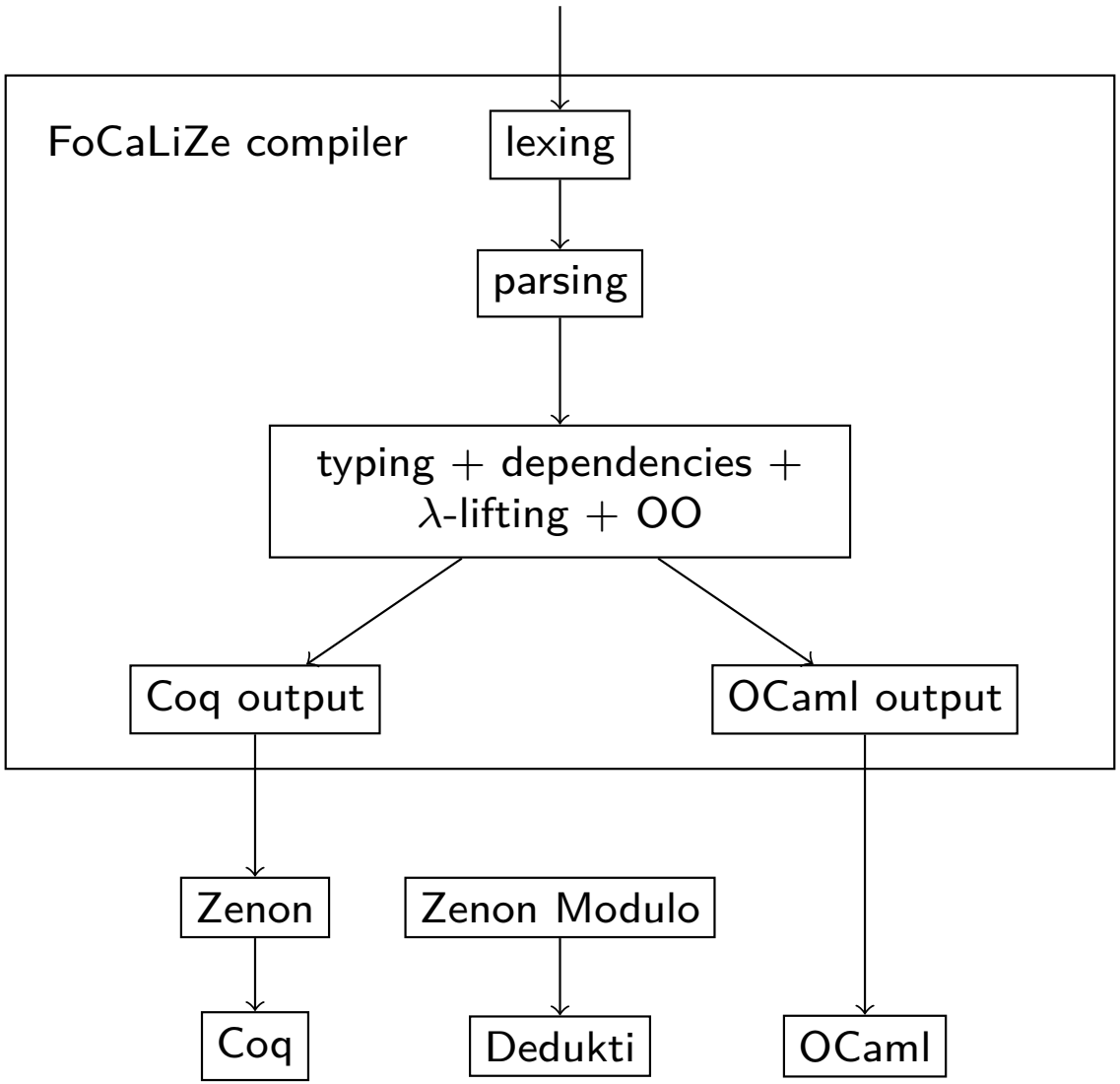
Efficient and formally verified programs

- Functional programming (ML)
- First-order specifications
- Automated proofs (Zenon)
- Modularity (inheritance and parameterization)

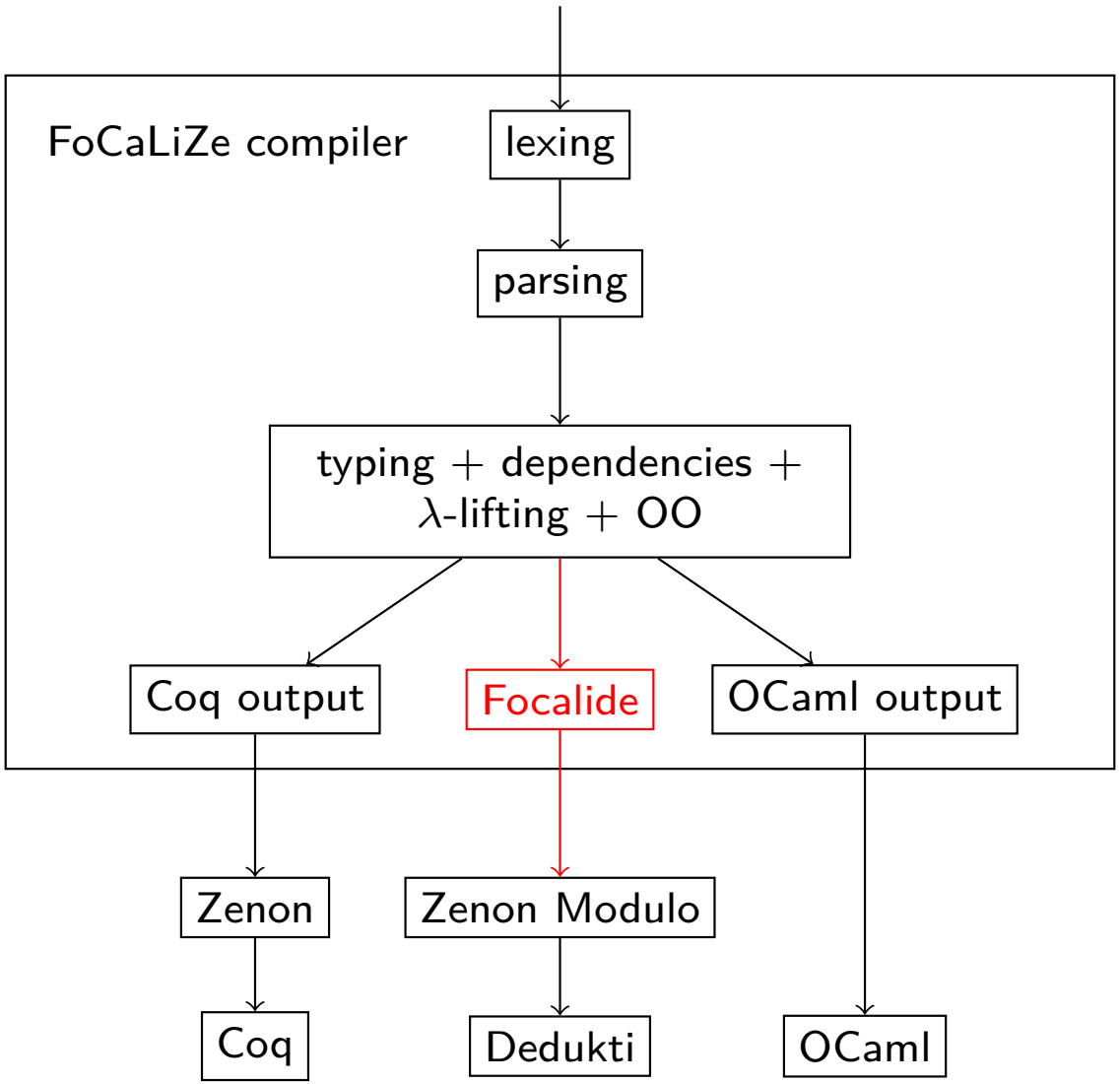
FoCaLiZe Compilation



FoCaLiZe Compilation



FoCaLiZe Compilation



Contribution

- Focalide: FoCaLiZe in Dedukti
- Integration of Zenon Modulo in FoCaLiZe
- Shallow translation of ML to Dedukti

Contribution

- Focalide: FoCaLiZe in Dedukti
- Integration of Zenon Modulo in FoCaLiZe
- Shallow translation of ML to Dedukti
  - Pattern-matching

Contribution

- Focalide: FoCaLiZe in Dedukti
- Integration of Zenon Modulo in FoCaLiZe
- Shallow translation of ML to Dedukti
  - Pattern-matching
  - Recursion

Contents

Contents

Pattern-Matching: Naive Translation

```
let f x =
  match x with
  | 0 -> 0
  | n -> n-1;
```

Pattern-Matching: Naive Translation

```
let f x =
  match x with
  | 0 -> 0
  | n -> n-1;
```

$$\begin{aligned} \mathsf{f} &: \mathbb{Z} \to \mathbb{Z} \\ \\ \mathsf{f}\,0 &\hookrightarrow 0 \\ \mathsf{f}\,n &\hookrightarrow n-1 \end{aligned}$$

Pattern-Matching: Naive Translation

```
let f x =
  match x with
  | 0 -> 0
  | n -> n-1;
```

$$\begin{aligned} \mathsf{f} &: \mathbb{Z} \to \mathbb{Z} \\ \mathsf{f}\,0 &\hookrightarrow 0 \\ \mathsf{f}\,n &\hookrightarrow n-1 \end{aligned}$$

- Not confluent: $\mathsf{f}\,0$ reduces to both 0 and -1.

Pattern-Matching: Naive Translation

```
let f x =
  match x with
  | 0 -> 0
  | n -> n-1;
```

$$\begin{aligned} \mathsf{f} &: \mathbb{Z} \to \mathbb{Z} \\ \mathsf{f}\ 0 &\hookrightarrow 0 \\ \mathsf{f}\ n &\hookrightarrow n - 1 \end{aligned}$$

- Not confluent: f 0 reduces to both 0 and -1.
- Inconsistency!: we can prove $0 = -1$

Pattern-Matching vs. Rewriting

	Pattern-Matching	Rewriting
Conflict resolution	First match	Confluent
Locality	Local	Toplevel

Pattern-Matching: Destructors

Destructor associated to the constructor C:

```
match a with
| C(x1, x2, x3, x4) -> b
| _ -> c
```

Pattern-Matching: Serialization

```
match a with
| p1 -> t1
| p2 -> t2
| p3 -> t3
```

$\rightsquigarrow$

```
match a with
| p1 -> t1
| _ ->
  (match a with
   | p2 -> t2
   | _ ->
     (match a with
      | p3 -> t3
      | _ -> ERROR
     )
  )
```

Pattern-Matching

```
match a with
| x -> b          ⇝    let x = a in b
| _ -> c
```

```
match a with
| _ -> b          ⇝    b
| _ -> c
```

```
match a with
| cst -> b        ⇝    if a = cst then b else c
| _ -> c
```

```
match a with                match a with
| p as x -> b     ⇝        | p -> let x = a in b
| _ -> c                    | _ -> c
```

Pattern-Matching: Flattening

```
match a with
| C(p1, ..., pn) -> b
| _ -> c
```

$\rightsquigarrow$

```
match a with
| C(x1, ..., xn) ->
  (match x1 with
  | p1 ->
    (match x2 with
    | p2 ->
    ...
      (match xn with
      | pn -> b
      | _ -> c)
    | _ -> c)
  | _ -> c)
| _ -> c
```

Pattern-Matching: Examples

```
let f x =
  match x with
  | 0 -> 0
  | n -> n-1;;
```

$\rightsquigarrow$

```
let f x =
  if x = 0 then 0 else
    let n = x in n - 1;;
```

Pattern-Matching: Examples

```
let f x =
  match x with
  | 0 -> 0
  | n -> n-1;;
```

$\rightsquigarrow$

```
let f x =
  if x = 0 then 0 else
    let n = x in n - 1;;
```

```
type nat =
  | O | S of nat;;

let pred x =
  match x with
  | O -> O
  | S(n) -> n;;
```

$\rightsquigarrow$

```
type nat =
  | O | S of nat;;

let pred x =
  match x with
  | O -> O
  | _ -> match x with
         | S(n) -> n
         | _ -> ERROR;;
```

Contents

Recursion: Locality

Local recursion is definable from toplevel recursion

```
let rec fix f x = f (fix f) x
```

Recursion: Naive Translation

```
let rec fact x =
  if x <= 1
   then 1
   else x * fact (x - 1)
```

Recursion: Naive Translation

```
let rec fact x =
  if x <= 1
   then 1
   else x * fact (x - 1)
```

$$\text{fact} : \mathbb{N} \to \mathbb{N}$$

$$\text{fact}\ x \hookrightarrow \texttt{if}\ (x \leq 1)\ \texttt{then}\ 1\ \texttt{else}\ (x \times \text{fact}\ (x - 1))$$

Recursion: Naive Translation

$$\forall x.\ \mathsf{fact}(x) > 0 \hookrightarrow$$

Recursion: Naive Translation

$$\forall x.\ \mathsf{fact}(x) > 0 \hookrightarrow$$
$$\forall x.\ (\mathsf{if}\ x \leq 1\ \mathsf{then}\ 1\ \mathsf{else}\ x \times \mathsf{fact}(x-1)) > 0 \hookrightarrow$$

Recursion: Naive Translation

$$\forall x.\ \mathsf{fact}(x) > 0 \hookrightarrow$$
$$\forall x.\ (\mathsf{if}\ x \leq 1\ \mathsf{then}\ 1\ \mathsf{else}\ x \times \mathsf{fact}(x-1)) > 0 \hookrightarrow$$
$$\forall x.\ (\ldots \mathsf{fact}(x-2) \ldots) > 0 \hookrightarrow$$
$$\ldots$$

Recursion

```
let rec fact x =
  if x <= 1
   then 1
   else x * fact (x - 1)
```

$\mathsf{fact} : \mathbb{N} \to \mathbb{N}.$
$\mathsf{fact'} : \mathbb{N} \to \mathbb{N}.$

$\mathsf{fact'}\ 0 \hookrightarrow \mathsf{fact}\ 0$
$\mathsf{fact'}\ (\mathsf{S}\ n) \hookrightarrow \mathsf{fact}\ (\mathsf{S}\ n)$

$\mathsf{fact}\ x \hookrightarrow$
$\quad \mathtt{if}\ (x \leq 1)$
$\quad\quad \mathtt{then}\ 1$
$\quad\quad \mathtt{else}\ (x \times \mathsf{fact'}\ (x - 1))$

Contents

Evaluation: Focalide Coverage

Library	Size	Coverage (%)
`stdlib`	160kB	99.42
`extlib`	155kB	100
`contribs`	124kB	99.54
`iterators`	78.4kB	88.33
`term-proof`	24.4kB	99.62
`ejcp`	13.7kB	95.16

Evaluation: Time (s)

Library	Zen	ZM	Coq	Dk	Zen + Coq	ZM + Dk
`stdlib`	11.73	32.87	17.41	1.46	29.14	34.33
`extlib`	9.48	26.50	19.45	1.64	28.93	28.14
`contribs`	5.38	9.96	26.92	1.17	32.30	11.13
`iterators`	2.58	3.85	6.59	0.27	9.17	4.12
`term-proof`	1.10	0.55	24.54	0.02	25.64	0.57
`ejcp`	0.44	0.86	11.13	0.06	11.57	0.92

Evaluation: Time (s)

Library	Zen	ZM	Coq	Dk	Zen + Coq	ZM + Dk
`stdlib`	11.73	32.87	17.41	1.46	29.14	34.33
`extlib`	9.48	26.50	19.45	1.64	28.93	28.14
`contribs`	5.38	9.96	26.92	1.17	32.30	11.13
`iterators`	2.58	3.85	6.59	0.27	9.17	4.12
`term-proof`	1.10	0.55	24.54	0.02	25.64	0.57
`ejcp`	0.44	0.86	11.13	0.06	11.57	0.92

- Zenon Modulo is slower (?)

Evaluation: Time (s)

Library	Zen	ZM	Coq	Dk	Zen + Coq	ZM + Dk
`stdlib`	11.73	32.87	17.41	1.46	29.14	34.33
`extlib`	9.48	26.50	19.45	1.64	28.93	28.14
`contribs`	5.38	9.96	26.92	1.17	32.30	11.13
`iterators`	2.58	3.85	6.59	0.27	9.17	4.12
`term-proof`	1.10	0.55	24.54	0.02	25.64	0.57
`ejcp`	0.44	0.86	11.13	0.06	11.57	0.92

- Zenon Modulo is slower (?)
- Dedukti is faster

Evaluation: Time (s) – Computation-intensive proof

n	Zenon	Coq	Zenon Modulo	Dedukti
10	31.48	4.63	0.04	0.00
11	63.05	11.04	0.04	0.00
12	99.55	7.55	0.05	0.00
13	197.80	10.97	0.04	0.00
14	348.87	1020.67	0.04	0.00
15	492.72	1087.13	0.04	0.00
16	724.46	$>$ 2h	0.04	0.00
17	1111.10	1433.76	0.04	0.00
18	1589.10	$>$2h	0.07	0.00
19	2310.48	$>$2h	0.04	0.00
...	...	...	...	...
2000	$>$2h	$>$2h	0.87	1.82

Conclusion

In FoCaLiZe	In Dedukti
Deduction modulo	A new logical system (FoCaLiZe)
Proof exchange	A new programming language (ML)
	Object-oriented modularity

Perspectives

- Zenon Modulo: optimize higher-order functions
- Pattern-matching: remove run-time errors
- Mutual recursion