The Makam metalanguage

Reducing the cost of experimentation in PL research

Antonis Stampoulis and Adam Chlipala

MIT CSAIL

2nd CRSX and HACS User Meetup
June 25th, 2014

What do languages of the future look like?

Refining language design ideas until they are “good enough” takes time

Refining language design ideas until they are “good enough” takes time

$\rightarrow$ Need better tooling for experimentation

Makam is a metalanguage for rapid PL prototyping

Makam is a metalanguage for rapid PL prototyping

- Main focus is expressivity
- Can handle modern research programming languages
- Small, conceptually clear core framework
- Close correspondence between definitions on paper and in Makam
- Prototyping in days instead of months!

Contributions

- The Makam metalanguage design, a refinement of λProlog
- Entirely new implementation
- Set of design patterns addressing common challenges
- Various large examples: type systems of OCaml, HOL, VeriML, Ur/Web; part of compilation from System F to TAL

Overview

Makam is a typed language

Makam is a typed language

- Need to declare object-level sorts and constructors before use
- Similar to describing abstract syntax on paper

Makam is a typed language

- Need to declare object-level sorts and constructors before use
- Similar to describing abstract syntax on paper

```
term : type.
typ : type.

intconst : int -> term.
plus : term -> term -> term.
app : term -> term -> term.

tint : typ.
arrow : typ -> typ -> typ.
```

Based on logic programming

Based on logic programming

- Predicates for different operations (e.g. typing, semantics, translations, etc.)
- Declarative and executable rules

Based on logic programming

- Predicates for different operations (e.g. typing, semantics, translations, etc.)
- Declarative and executable rules

```
typeof : term -> typ -> prop.
eval : term -> term -> prop.

typeof (app E1 E2) T' <-
  typeof E1 (arrow T T'), typeof E2 T.
```

Representing binding

Representing binding

$$\frac{\Gamma,\ x : \tau \vdash e : \tau'}{\Gamma \vdash \lambda x.e : \tau \to \tau'} \qquad \frac{e_1 \Downarrow \lambda x.e \qquad e_2 \Downarrow v_2 \qquad e[v_2/x] \Downarrow v'}{e_1\ e_2 \Downarrow v'}$$

Representing binding

$$\frac{\Gamma,\ x : \tau \vdash e : \tau'}{\Gamma \vdash \lambda x.e : \tau \to \tau'} \qquad \frac{e_1 \Downarrow \lambda x.e \qquad e_2 \Downarrow v_2 \qquad e[v_2/x] \Downarrow v'}{e_1\ e_2 \Downarrow v'}$$

- A common and significant challenge in language implementation
- λProlog idea: implement once and for all in the metalanguage; reuse it in the object languages

Higher-order abstract syntax

```
lam : (term -> term) -> term.

typeof (lam E) (arrow T T') <-
  (x:term -> typeof x T -> typeof (E x) T').

eval (app E1 E2) V' <-
  eval E1 (lam E), eval E2 V2, eval (E V2) V'.
```

Querying

Querying

- Querying for `typeof` gives us a prototype type checker
- Querying for `eval` gives us a prototype interpreter

Querying

- Querying for `typeof` gives us a prototype type checker
- Querying for `eval` gives us a prototype interpreter

```
typeof (lam (fun x => plus x x)) T ?
>> T := arrow tint tint
```

Polymorphism
& higher-order predicates

Polymorphism & higher-order predicates

```
map : (A -> B -> prop) -> list A -> list B -> prop.
map P (cons HD TL) (cons HD' TL') <-
  P HD HD', map P TL TL'.
map P nil nil.

tuple : list term -> term.
prod  : list typ -> typ.
typeof (tuple ES) (prod TS) <- map typeof ES TS.
```

Polymorphic binding structures

- Examples: multiple binding, mutual recursion, linear variables, etc.
- Definable within the language

Polymorphic binding structures

- Examples: multiple binding, mutual recursion, linear variables, etc.
- Definable within the language

```
lammany : (term -> (term -> .... -> term)) -> term.
```

Polymorphic binding structures

- Examples: multiple binding, mutual recursion, linear variables, etc.
- Definable within the language

```
lammany : bindmany term term -> term.
```

Polymorphic binding structures

- Examples: multiple binding, mutual recursion, linear variables, etc.
- Definable within the language

```
lammany : bindmany term term -> term.

typeof (lammany E) (arrowmany TS T) <-
   newvars_many E (fun xs body =>
   assume_many typeof xs TS
   (typeof body T)).
```

Unification in Makam

Unification in Makam

- Based on the higher-order pattern matching algorithm
- Means that unification is aware of the HOAS binding structure
- Subsumes the core operations of many type inferencing mechanisms

Unification in Makam

```
polylam : (typ -> term) -> term.
polyinst : term -> typ -> term.
forall : (typ -> typ) -> typ.

typeof (polylam E) (forall T) <-
  (a:typ -> typeof (E a) (T a)).

typeof (polyinst E T) (T' T) <-
  typeof E (pi T').
```

Structural recursion

Structural recursion

- Many operations are defined through structural recursion save for a few essential cases
- Usually need lots of boilerplate

Structural recursion

- Many operations are defined through structural recursion save for a few essential cases
- Usually need lots of boilerplate

```
expandsugar : term -> term -> prop.
expandsugar (lammany E) E' <- ...
expandsugar (app E1 E2) (app E1' E2') <-
  expandsugar E1 E1', expandsugar E2 E2'.
expandsugar (lam E) (lam E') <-
 (x:term -> expandsugar x x -> expandsugar (E x) (E' x)).
```

Structural recursion

Structural recursion

- We can define a fully generic structural recursion operation in Makam
- Relies on dynamic typing unification
- Works even with auxiliary data structures like `list` and `bindmany`

Structural recursion

- We can define a fully generic structural recursion operation in Makam
- Relies on dynamic typing unification
- Works even with auxiliary data structures like `list` and `bindmany`

```
expandsugar : term -> term -> prop.
expandsugar E E' <-
  ifte (eq E (lammany _))
       (...)
       (structural expandsugar E E').
```

Staging

Staging

- Predicates that compute other predicates, top-level commands, etc.
- Allows metalanguage extensions to be defined within the metalanguage
- Examples: parser and pretty-printer generation, mode declarations, etc.

Staging

- Predicates that compute other predicates, top-level commands, etc.
- Allows metalanguage extensions to be defined within the metalanguage
- Examples: parser and pretty-printer generation, mode declarations, etc.

```
‘( parse term
    ( "λ" x:string "." e:term  { lam (nu x e) } ) ).
```

Examples

OCaml type system	550 lines
Type classes	100 lines
Higher-order logic	250 lines
VeriML constructs	150 lines
Featherweight Ur	500 lines
System F to TAL	850 lines
PEG parser gen	350 lines
LF	350 lines

Summary

- Makam reduces PL prototyping time from months to days
- Small core yet surprisingly expressive
- Reusable design patterns to handle common challenges
- Can already handle sophisticated type systems and translation procedures
- Will release publicly mid-July

Thanks!

Backup slides

Staging

Staging

```
and : prop -> prop -> prop.
newvar : (A -> prop) -> prop.
assume : prop -> prop -> prop.

cmd_newconst : string -> A -> cmd.
cmd_newrule : prop -> prop -> cmd.
cmd_stage : (cmd -> prop) -> cmd.
```