Present & future of the Moca c

Pierre Weis

Ssurf – May 13 2008

The new Moca release

The *0.6* release of the Moca compiler was delivered
the 13^{th} of 2008.

The main new feature is the *automatic test generatio*
generated files. This has been done by Laura Lowe
its internship at Loria (extremely successful interns
ment).

From each relational data type definition a set of te
the invariants is generated.

The new Moca enhancemen

In addition, we did:

- Numerous bug fixes (in particular thanks to Laura tests).
- Completion has been enhanced.
- Distributivity has been widely generalized.
- Vary-ary generation completely revisited.
- Some bug additions (for instance `Division_by_a` moval).

Embedded user's Caml coo

An important new feature is the possibility for the u
arbitrary Caml code within the definition of the rela
This was mandatory to define a specific comparis
when the regular Caml polymorphic `Pervasives.co`
semantically sound for the relation type. The us
output as is, *after* the definition of the relational type
the beginning of the definition of the construction f

As usual, the documentation has been improved,
by the addition of the ESOP article, the JFLA talk,
other talks given about moca.

The next Moca release(s)

We will split the future development of *Moca* into t

- the implementation part,
- the research part.

The implementation part is shorter term and practic
well understood.

The research part is long term and may be imprac
unclear, half backed and not understood at all.

The implementation part pl

We split the implementation plan into:

- revisit the algebraic keywords specification,
- enhance the internal test bed,
- enhance the automatic test generation procedur
- write the manual for Moca.

Implementation: specificati

For each algebraic keyword, we must precisely defin
iors for each of its variations. In particular, concer
generators, we must fix the vocabulary:

- constant generator (or constary ?),
- unary generator,
- binary generator (two arguments),
- listary generator (a.k.a. vary-ary or vary-adic),
- multary generator (a.k.s. multi-ary i.e. any arity

Implementation: vocabular

By definition:

- a *constary* generator has *no argument*,
- a *unary* generator (has *one* argument which is *r*
- a *binary* generator (has *two* arguments),
- a *listary* generator (has *one* argument which is *a*
- a *multary* generator (has any number of argume

Implementation: specificati

For each keyword and each arity, give:

- the applicability to each arity, the applicability v
  or required property,
- the rule(s) generated (match clause),
- the priority w.r.t. other rules (?),
- the "no values of the relational type matches" s
- systematically specify the `Left`, `Right`, and `Both`

No macro expansion in the `parser.mly` syntax !

Implementation: the test dire

We should enrich the test bed cases for *Moca*:

- enhance the internal test bed to check as much
  the *combination* of algebraic rules,
- complete the set of test files to handle the usual
  cal structures (fields, vectorial spaces, etc.) and
  usual generators/relations presentations of grou
  in Coxeter and alii),
- move some test files to the examples for the m

Implementation: the automatic test

Augment the test generation procedure, such that:

- for each keyword and each case of the keyword s
  add a specific test bed to check the behavior of
  w.r.t. this specification,
- enhance the automatic test generation procedu
  polymorphic relational data types,
- more generally, enhance the test procedure to
  exhaustive set of examples given in the test dire

The research part

We will split the future research development of *Mo*

- the Test,
- the Completion,
- the Focalize Library,
- the Proofs,
- Moca for Focalize,
- Moca for the Caml programmer,

Research: the Test generati

Try to understand the generality of the test gener
dure:

- how to generalize the procedure to user's define
  (relations) ?
- how to generalize the procedure to the full Cam

Research: Completion algorit

- Generalize usage of automatic completion,
- AC completion ?
- How to generate complex rules via completion ?

Research: generation of comple

An easy algebraic reasoning proves that the rules

`Absorbent` + `Inverse`

induce the rule

`Division_by_absorbent`.

Hence, the generation function for `Inverse` needs th

`A -> failwith "Division by absorbent"`

UNLESS $A = E$ holds.

Is it possible to generate such a complex completio

Research: the Focalize libra

- Take the Focalize library and "implement" it u algebraic rules.
- Implement the associated algorithm of the Foca

Research: Proofs

Write proofs, proofs, and proofs!!!

- write by hand a proof of a simple example of moc
code,
- understand how to generalize the preceding proof
a proof with the generated code (file `file_coq.v`

Gather and carefully state the generic properties ar
the Moca generated code, to be able to go on next

Research: Moca for Focali

Interface Moca to Focalize:

- add a `private` type facility to Focalize data type
- interface Moca to Focalize to allow relational da initions in Focalize,
- add the relevant lemmas and properties for the functions to the Focalize code (free proofs to Zenon and Coq).

Research: Moca for Cam

Use Moca to generate `.mli` files that we do not war

- from a `.mlm` file with additional annotations:
  - `export` clauses to export relevant identifiers,
  - `abstract` annotations for abstract data types,
  - `test` annotations to specify test equations or d
    or ad hoc algebraic rules.

A data types impedimenta gen

- Other generations (similar to `-test`) ?
  - `set` clause to generate set universes,
  - `data base` annotation to generate data bases,
  - `make_string` annotation to generate a `make_str`
  - `read_string` annotation to generate a `read_str`
- Generalization to general printing function and p
tion, (which printing annotations? which parsing
...).

Development guidelines

Program with peace in mind, since

No confusion can ever arise

- except for the value of some quantities, unknow
  time, hence impossible for us to check,
- since anyway all the *Moca* generated programs
  again by the Caml compiler (including the com
  exhaustive and fragile matches in pattern match