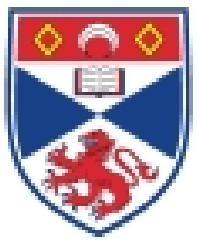
Interfacing Coq+SSReflect with with GAP

Vladimir Komendantsky Alexander Konovalov
Steve Linton

University of St Andrews, UK



?

UITP @ FLoC'10

Motivation

What can interactive TP and CAS do for each other

Related interfaces

Brief background

Term levels

The SSReflect hierarchy

Towards a concrete development

Towards a development regarding the SSReflect hierarchy

Possible development tracks of a Coq–CAS interface

1. CAS as a hint engine for Coq
2. CAS as a proof engine for Coq
3. prove in Coq correctness of CAS algorithms

The particular CAS we are working with:

GAP – Groups, Algorithms, Programs

`http://www.gap-system.org/`

Motivation 1: Hint engine

Mechanisms, introduced in proof assistants to improve the type inference algorithm:

- canonical structures: $\dfrac{?_x := t}{\pi_i ?_x \equiv u}$
- type classes,
- pullbacks,
- unification hints (Asperti et al. '09): $\dfrac{\vec{?_x} := \vec{H}}{P \equiv Q}$

Problem

Provide suitable hints for the unification procedure underlying type inference. How useful would be to employ the CAS as a search tool to generate hints?

Motivation 2: Proof engine

- **Admit GAP results.** (Good scenario, although with limitations.)

Example

Automatically generate an axiom, e.g.,

```
Axiom isoG : G \iso (Zp 12).
```

if we asked GAP to find a group isomorphic to a given Coq group `G` and the result was the Coq group `Zp 12`.

- **Prove GAP results** (something Coq does not know in advance but can prove if a hint is received from GAP).

Example

As above, but ask Coq user to prove the generated statements, i.e.,

```
Lemma isoG : G \iso (Zp 12).
```

followed by user or automated proof and `Qed`, or proof is impossible.

Motivation 3: Correctness of GAP algorithms

Frequently, GAP standard library functions rely on side effects produced by other functions:

```
##############################################################################
##
#M  Size( <G> ) . . . . .  for cyclotomic matrix group not known to be finite
##
InstallMethod( Size,
    "cyclotomic matrix group not known to be finite",
    [ IsCyclotomicMatrixGroup ],
    function( G )
    if IsFinite( G ) then
        return Size( G );  # now G knows it is finite
    else
        return infinity;
    fi;
end );
```

Motivation 3: Correctness of GAP algorithms

Frequently, GAP standard library functions rely on side effects produced by other functions:

```
##############################################################################
##
#M  Size( <G> ) . . . . .  for cyclotomic matrix group not known to be finite
##
InstallMethod( Size,
    "cyclotomic matrix group not known to be finite",
    [ IsCyclotomicMatrixGroup ],
    function( G )
    if IsFinite( G ) then
        return Size( G );  # now G knows it is finite
    else
        return infinity;
    fi;
end );
```

Let's use side effects!

A formal model of algorithm implementation ***would*** *imply a formal model of its specification*

```
##
#M  IsFinite( G ) . . . . . . . . . . .  IsFinite for cyclotomic matrix group
##
InstallMethod( IsFinite, "cyclotomic matrix group", [ IsCyclotomicMatrixGroup ],
function( G )
    local lat, ilat, grp, mat;
    # if not rational, use the nice monomorphism into a rational matrix group
    if not IsRationalMatrixGroup( G ) then
        return IsFinite( Image( NiceMonomorphism( G ) ) );
    fi;
    # if not integer, choose basis in which it is integer
    if not IsIntegerMatrixGroup( G ) then
        lat := InvariantLattice( G );
        if lat = fail then return false; fi;
        ilat := lat^-1; grp := G^(ilat); IsFinite( grp );
        # IsFinite may have set the size, so we save it;
        # <code omitted>
        # IsFinite may have set an invariant quadratic form
        # <code omitted>
        return IsFinite( grp );
    else
        return IsFinite( G );  # now G knows it is integer
    fi;
end );
```

Related work

- J. Harrison and L. Théry: HOL–Maple
- H. Herbelin, M. Mayero, D. Delahaye: "Maple mode" for Coq
- S. Ould-Biha: prototype external tactic coq_gap in C

Term levels in Coq-like proof assistants

1. fully specified terms: CIC terms

$$\forall a{:}\mathbb{Z},\ eq\ \mathbb{Z}\ a\ (Zplus\ a\ Z0)$$

2. partially specified terms: CIC terms with omitted subterms

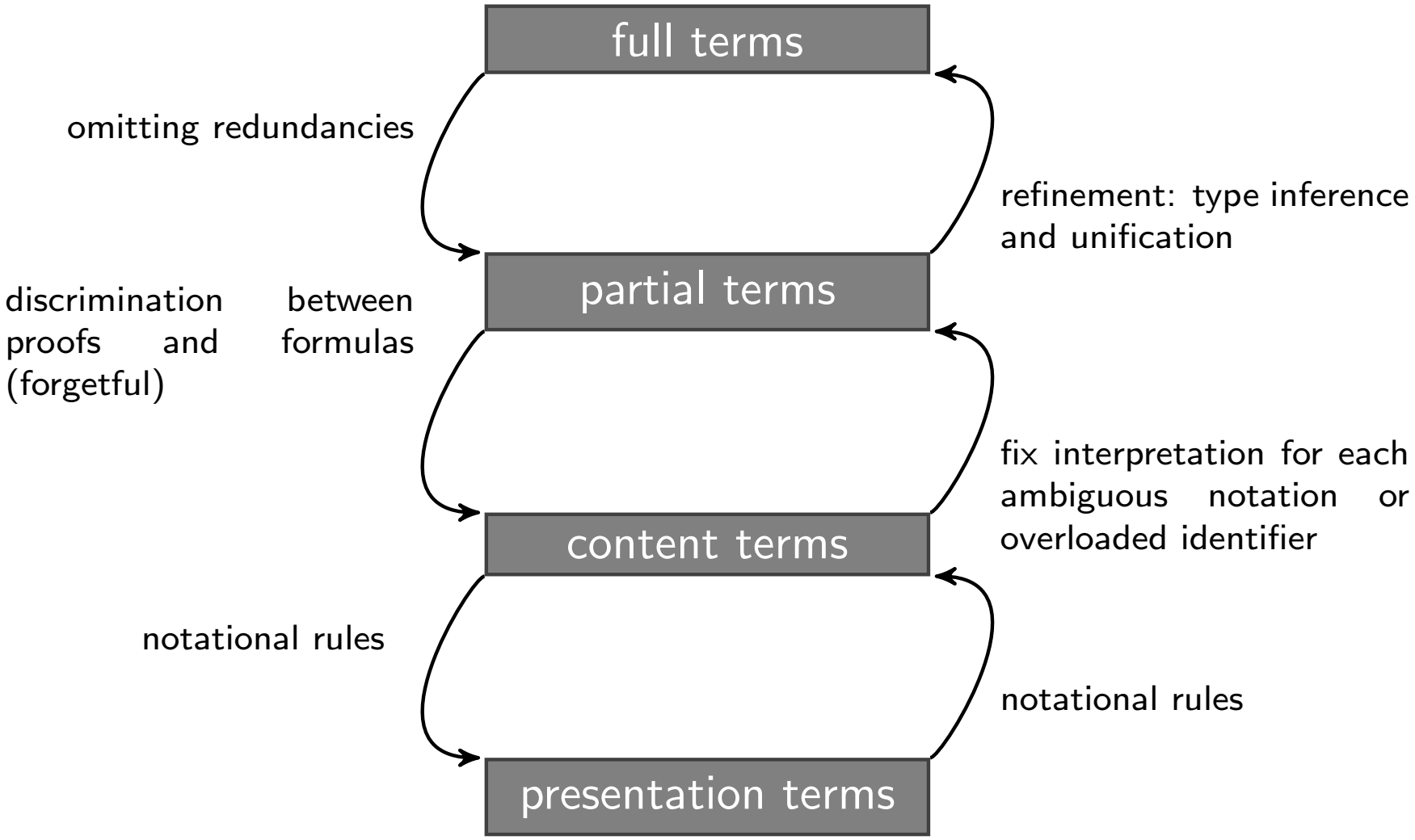
$$\forall a{:}\mathbb{Z},\ eq\ ?_1\ a\ (Zplus\ ?_2\ Z0)$$

3. content level terms: compact, overloaded human-oriented encoding with abstract syntactic structure

$$\forall a,\ a = \_ + 0$$

4. presentation level terms: formatting structure; finite, non-extendible language (e.g., MathML for printing, TeX-like for editing)

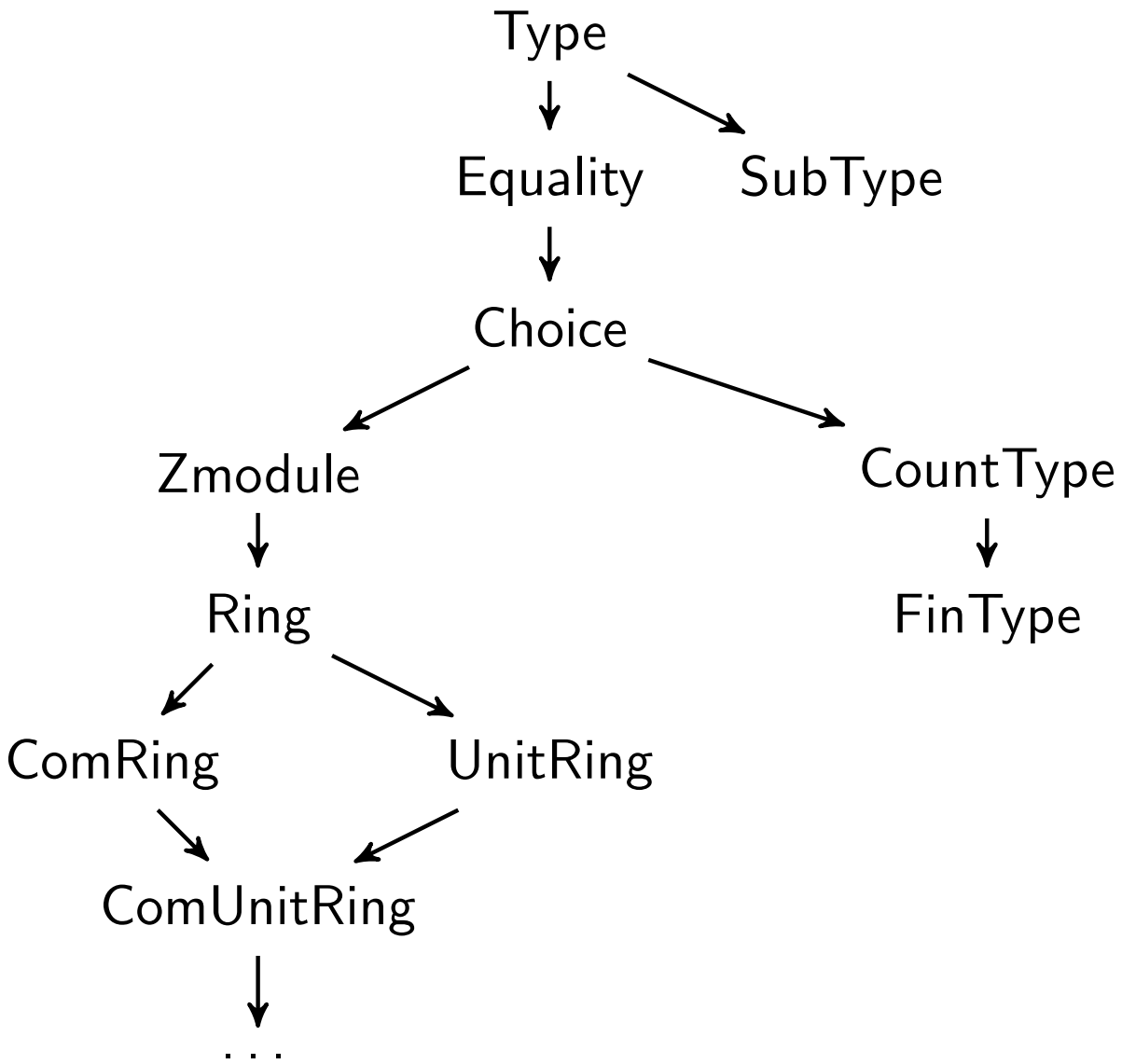
Term level translation



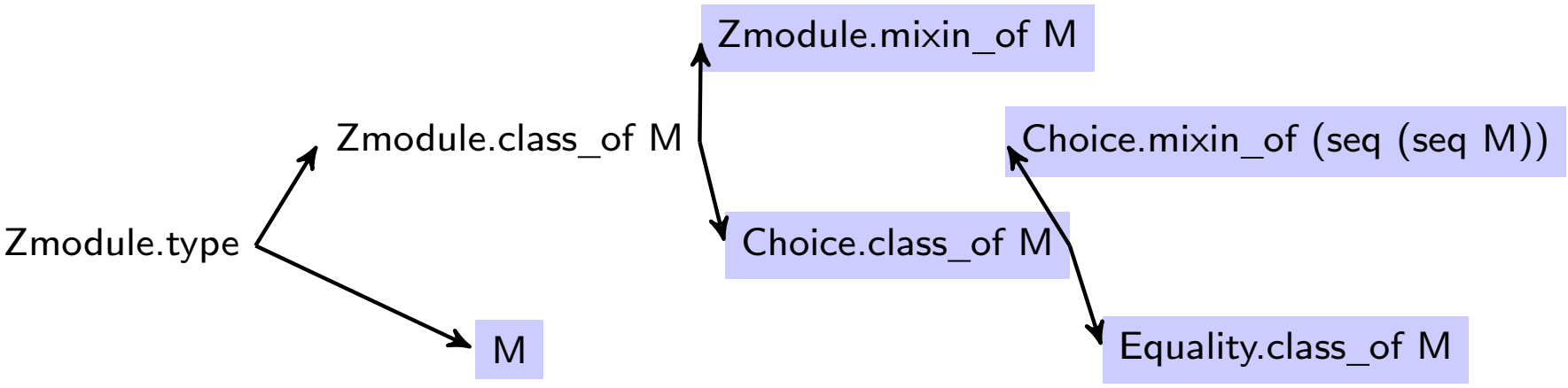
The choice of formalised mathematical structures: SSReflect's packed classes

- Formalisation of mathematical concepts in type theory is not straightforward!
- There may be several ways to formalise a concept, and one is required to choose the most appropriate way.
- *De Bruijn factor*: formalised proof / paper proof (in lines of text)
- Formalise informal maths without making explicit too much information.
- Structures with inheritance; shared carrier and unification.

Algebraic hierarchy in SSReflect



Packed classes and mixins (Zmodule example)



Legend

T_1 *inherits* from (*coerces* to) T_2

Representation of polynomials in OpenMath

$$x^4y^6 + 3y^5$$

```
DMP(poly_ring_d(ℤ, 2), SDMP(term(1, 4, 6), term(3, 0, 5)))
```

- `DMP`, constructor of distributed multivariate polynomials
- `poly_ring_d`, constructor of polynomial rings
- `SDMP`, constructor of formal sums
- `term`, constructor of monomials:

  $$\texttt{term}(c, e_1, \ldots, e_n)$$

  represents

  $$c \times v_1^{e_1} \times \cdots \times v_n^{e_n}$$

Univariate case of polynomial

Compute roots of $x^3 - 1$ in $\mathbb{Z}_3$.

1. Function call:

```
WS_RootsOfUPol(
  DMP(poly_ring_d_named(GFp(3), "x"),
    SDMP(
      term(power(primitive_element(3), 0), 3),
      term(power(primitive_element(3), 1)))))
```

2. Roots:

```
list(power(primitive_element(3), 0),
     power(primitive_element(3), 0),
     power(primitive_element(3), 0))
```

or, simply $[2^0; 2^0; 2^0]$.

Univariate polynomials in SSReflect

```
Record polynomial : Type :=
  Polynomial {polyseq :> seq R; _ : last 1 polyseq != 0}.

Definition Poly := foldr poly_cons (polyC 0).
Definition polyX := Poly [:: 0; 1].
```

```
Parameters (R : ringType) (p : {poly R}).
Print R.
  *** [ R : GRing.Ring.type ]
Print p.
  *** [ p : @poly_of R (Phant (GRing.Ring.sort R)) ]
```

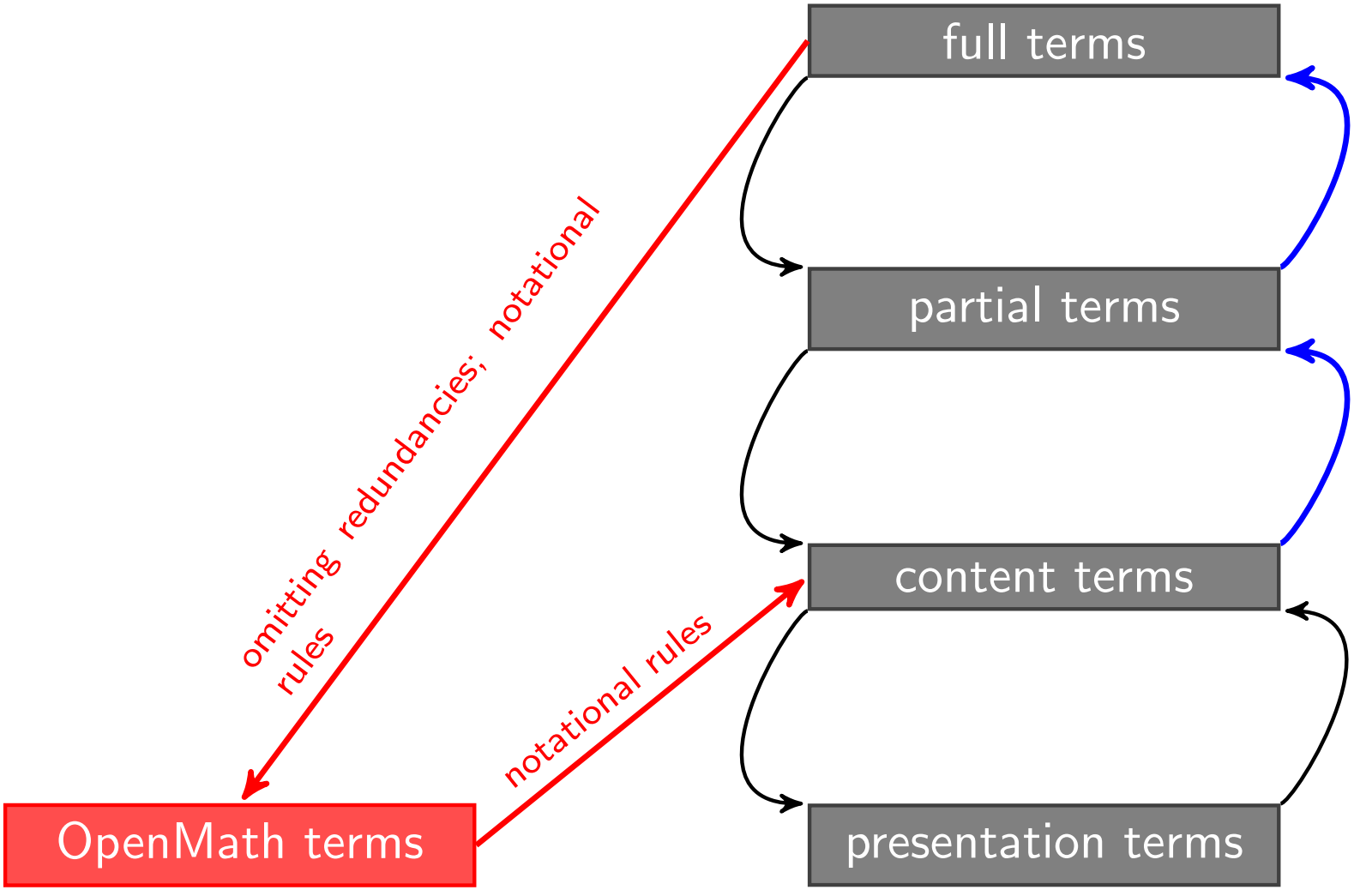
SSReflect polynomial example

$$x^3 - 1$$

```
[:: -1; 0; 0; 1]
```

```
[:: 2^0] = [:: expn (nat_of_ord 2) 0]
```

Term encoding of foreign data



Implementation: Use a communication protocol

SCSCP – Symbolic Computation Software Composability Protocol

`http://www.symbolic-computation.org/scscp`

- A computer algebra system may offer services over network and a client may employ them.
- All messages in the protocol are represented as OpenMath objects, using the new Content Dictionaries scscp1 and scscp2, developed for this purpose.

Summary

We are. . .

- working with a particular kind of mathematical structures in Type Theory: the packed classes of SSReflect (constructive Finite Group Theory library);
- currently implementing a prototype hint engine for Coq that employs GAP as a source of background knowledge:
  - higher-level (content-oriented) interaction with GAP;
  - support for packed classes is being developed.

Summary

We are. . .

- working with a particular kind of mathematical structures in Type Theory: the packed classes of SSReflect (constructive Finite Group Theory library);
- currently implementing a prototype hint engine for Coq that employs GAP as a source of background knowledge:
  - higher-level (content-oriented) interaction with GAP;
  - support for packed classes is being developed.

Thanks for listening. . .