Formal proof of SCHUR conjugate function

Franck Butelle[1] Florent Hivert[2] Micaela Mayero[1]
Frédéric Toumazet[3]

[1] Univ. Paris 13, LIPN UMR 7030

[2] Univ. Paris 11, LRI UMR 8623

[4] Univ. Marne-la-Vallée, LIGM UMR 8049

MAP 2012 — Konstanz

- Objectives and tools
- Some combinatorial objects
- The C function
- Formal Proof
- Conclusion

Objectives and tools

- Proof of concept:
  - Algebraic combinatorics area: combinatorial explosion
  - SCHUR software, now under GNU GPL
  - Prove an old C program, uncommented and tricky, not designed to be proved !
    - extract one key function, simple but quite representative
    - prove it
    - try to deduce some methodology
- Tools and Means
  - Frama-C, plug-in Jessie
  - First-order logic annotations
  - Automatic provers...
  - And if it is not enough, use interactive provers.

SCHUR

- Interactive software (more than 240 commands)
- calculate properties of Lie groups and symmetric functions
- 20 years of research in algebraic combinatorics, physics, etc.
- Tool for computations, conjectures, teaching,...
- Over 45 000 lines of C without comments
- Originally written by B.G. Wybourne.
- Now maintained by F. Butelle, R. King and F. Toumazet.
- Nowadays under GPL (sourceforge.net).

Frama-C / Jessie

- platform for source-code analysis of C software (successor of Caduceus)
- Jessie plug-in: generate verification conditions from first-order logic annotations (ACSL, based on Why, Hoare logic).
- Call external automatic provers (SMT) (Simplify, Alt-Ergo, Z3, CVC3,...)
- Many output formats available for interactive provers (Coq, PVS, Isabelle/HOL,...)
- Graphical interface
- (Now Why3)

Some combinatorial objects

- Objectives and tools
- Some combinatorial objects
  - Integer Partition
  - Young Tableaux
  - Symmetric Functions
  - Schur Functions
  - The conjugate
- The C function
- Formal Proof
- Conclusion

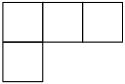
Integer Partition and Ferrers diagrams

Definition: Integer Partition

write n has a sum of non increasing integers

Example : $4 = 3 + 1 = 2 + 2 = 2 + 1 + 1 = 1 + 1 + 1 + 1$

Definition: Ferrers Diagram

The integer partition $\lambda = (3, 1)$ can be represented by the following diagram $F^{\lambda} =$ 

Important role:

- group representation theory
- symmetric polynomials and the symmetric group
- Frobenius (1849–1917): irreducible representations of symmetric groups are indexed by integer partitions...

Young Tableaux

Definition: a semi-standard Young tableau

of shape λ is a numbering of the boxes of F^λ with entries from $\{1, 2, ..., n\}$, weakly increasing across rows and strictly increasing down columns.

Example : $\lambda = (4, 2, 2, 1)$:

1	2	2	5
2	4		
3	6		
5			

Symmetric Functions

Definition: a symmetric function

$f(x_1, x_2, \dots)$ is invariant under any permutation of its variables:
$f(x_1, x_2, \dots) = f(x_2, x_1, \dots) = \dots$

Usually restricted to polynomials functions.

Schur Functions

Definition: a Schur function

For a semi-standard Young tableau T of shape λ,
if X^T is the product of all x_i, for all i appearing in T, then $s_\lambda = \sum_{T \in \mathrm{Tab}(\lambda)} x^T$ where $\mathrm{Tab}(\lambda)$ is the set of all tableaux of shape λ.

Example :

$\lambda = (2,1)$; when using alphabet $\{1,2,3\}$, $Tab(\lambda) =$

$$\begin{array}{|c|c|}\hline 1 & 1 \\ \hline 2 & \\ \cline{1-1}\end{array} \quad \begin{array}{|c|c|}\hline 1 & 1 \\ \hline 3 & \\ \cline{1-1}\end{array} \quad \begin{array}{|c|c|}\hline 2 & 2 \\ \hline 3 & \\ \cline{1-1}\end{array} \quad \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 & \\ \cline{1-1}\end{array} \quad \begin{array}{|c|c|}\hline 1 & 3 \\ \hline 2 & \\ \cline{1-1}\end{array} \quad \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 2 & \\ \cline{1-1}\end{array} \quad \begin{array}{|c|c|}\hline 1 & 3 \\ \hline 3 & \\ \cline{1-1}\end{array} \quad \begin{array}{|c|c|}\hline 2 & 3 \\ \hline 3 & \\ \cline{1-1}\end{array}$$

$$s_{(21)}(x_1, x_2, x_3) = x_1^2 x_2 + x_1^2 x_3 + x_2^2 x_3 + 2 x_1 x_2 x_3 + x_1 x_2^2 + x_1 x_3^2 + x_2 x_3^2$$

Schur functions are the most important linear basis of symmetric function's algebra.

Computation in algebraic combinatorics

Architecture of a software for computing in algebraic combinatorics:

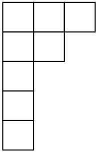
- a computer algebra kernel
- a very large bunch of small combinatorial functions which enumerate and manipulate the combinatorial data structures.
  - surgery on lists of integers or lists of lists of integers
  - computing the conjugate of a partition is a very good example...
    - used by more than 100 commands in Schur.

Conjugate partition

The conjugate of an integer partition is the partition associated to the diagonal symmetric of its shape.

Example

$\lambda = (3, 2, 1, 1, 1)$

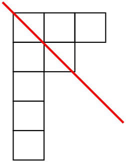


Conjugate partition

The conjugate of an integer partition is the partition associated to the diagonal symmetric of its shape.

Example

$\lambda = (3, 2, 1, 1, 1)$

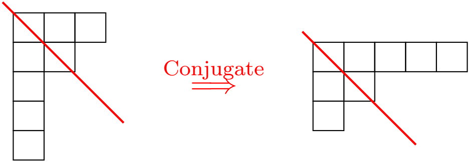


Conjugate partition

The conjugate of an integer partition is the partition associated to the diagonal symmetric of its shape.

Example

$\lambda = (3, 2, 1, 1, 1)$



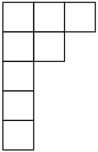
The conjugate of $(3, 2, 1, 1, 1)$ is therefore $(5, 2, 1)$.

Computation of the conjugate

A partition $\longrightarrow$ an array of integers.

$$\lambda = (3, 2, 1, 1, 1) \quad \longrightarrow \quad t[1] = 3, t[2] = 2, \ldots t[l(\lambda)] = 1.$$

Computation of the conjugate: count boxes.



Conjugate $\Longrightarrow$

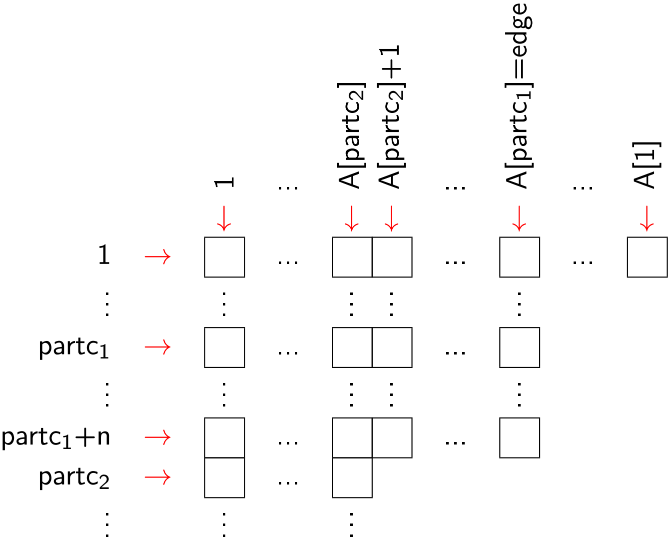
$$5, 2, 1 \begin{cases} 5 = & \#\text{ lines of length } \geq 1 \\ 2 = & \#\text{ lines of length } \geq 2 \\ 1 = & \#\text{ lines of length } \geq 3 \end{cases}$$

$$t_c[j] \;=\; |\{i \mid 1 \leq i \leq l(\lambda) \wedge t[i] \geq j\}|$$

The conjugate function in SCHUR

```c
#define MAX 100
void conjgte (int A[MAX], int B[MAX])
{
  int i, partc = 1, edge = 0;
  while (A[partc] != 0) {
     edge = A[partc];
     do
        partc = partc + 1;
     while (A[partc] == edge);

     for (i = A[partc] + 1; i <= edge; i++)
        B[i] = partc - 1;
  }
}
```

Function behavior



Annotations for the formal proof

no integer overflow is allowed...

```
#pragma JessieIntegerModel(strict)
```

Predicate declaration (based on actual data structure in SCHUR)

```
/*@ predicate is_partition{L}(int t[]) =
    ( \forall integer i; 1 <= i < MAX ==>
        0 <= t[i] < (MAX-1)) &&
    ( \forall integer i,j; 1 <= i <=j < MAX ==>
        t[j] <= t[i]) &&
    t[MAX-1]==0;
*/
```

Some more predicates...

countIfSup(t,j,k,z) is true iff z equals the number of lines of t, whose indexes are in $\{1, .., j-1\}$ and of length $\geq k$...

```
/*@ predicate countIfSup{L}(
      int t[],integer j,integer k,integer z) =
    is_partition{L}(t) &&
    1<= j<= MAX && 1<=k<MAX &&
    ((1<=z<j && \forall integer i ; 1<=i<=z ==> t[i]>=k)
     || (z==0 && \forall integer i ; 1<=i<j ==> t[i]<k));
 */
```

We deduce the postcondition to verify: t2 is the conjugate of t1 iff

```
/*@ predicate is_conjugate{L}(int t1[], int t2[]) =
    \forall integer k ; 1<=k<MAX
      ==> countIfSup(t1,MAX,k,t2[k]);
 */
```

Pre and Post conditions

```
/*@ requires \valid(A+ (1..(MAX-1)));
    requires \valid(B+ (1..(MAX-1)));
    requires is_partition(A);
    requires \forall integer k; 1<=k<MAX ==> B[k]== 0;
    assigns B[1..A[1]];
    ensures is_conjugate(A,B);
    */
void conjgte (int A[MAX], int B[MAX])
{
  int i, partc=1, edge = 0 ;
```

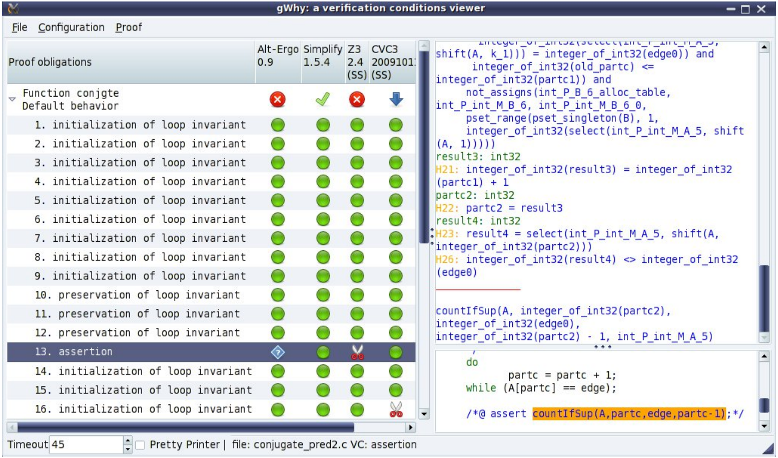
Loop invariant 1

```
/*@ loop variant MAX-partc;
    loop invariant 1<=partc<MAX;
    loop assigns B[1..A[1]];
    loop invariant \forall integer k;
       A[partc]+1<=k<=A[1] ==> countIfSup(A,MAX,k,B[k]);
  */
while (A[partc] != 0) {
   edge = A[partc];

/*@ ghost int old_partc = partc; */

/*@ loop variant MAX-partc;
    loop invariant old_partc<=partc ;
    loop invariant \forall integer k;
       old_partc<=k<=partc ==> A[k]==edge;
    loop invariant partc<MAX-1;
  */
do
   partc = partc + 1;
while (A[partc] == edge);
```

Loop invariant 2

```
/*@ assert countIfSup(A, partc , edge , partc -1);*/

/*@ loop variant edge-i;
    loop invariant i >= A[partc]+1 && edge+1>=i ;
    loop invariant \forall integer k;
        A[partc]+1 <=k <i ==> countIfSup(A,MAX,k,B[k]);
    loop assigns B[ (A[partc]+1)..edge];
  */
for (i = A[partc] + 1; i <= edge; i++)
   B[i] = partc - 1;
}
```

gWhy interface



Coq proof assistant

- When no automatic prover is able to prove a verification condition:
  try to achieve an assisted proof

Two cases:

- Either we are able to identify "errors" in annotations
  $\hookrightarrow$ correction and back to automatic SMT provers
- Or the property is "too complex" for SMT provers
  $\hookrightarrow$ prove it with Coq help

Coq for the conjugate

Identifying problems in annotations

Small causes, great consequences...

- Replacing initial axiomatic definition by a predicate for `countIfSup`
- postcondition proof.
  - one more precondition required
  - Definition of `countIfSup` was incomplete (the second part of || was missing)
- Loop Invariant.
  mistake in definition of `countIfSup` : $j < MAX$ instead of $j \leq MAX$

Difficulties

- Isolate piece of code to prove.
- Make "good" annotations without changing original code.
- Prove all verification conditions
- What confidence in automatic provers ?
  - no trace of how things are proved
  - some CVC3 versions have bugs...

Conclusion

- Proof of a key/typical function of SCHUR.
- Beginning of a methodology to prove a "big" computing software.
- Interactions between two communities
- For combinatorists, increase confidence in computations
- For formal proof people, proof of concept/feasibility

Future works

- Proving enumerative computations:
  - Littlewood-Richardson coefficients
  - Kostkas numbers
  - Kostkas matrices
  - ...
- Library formally proved