A formal proof of the Littlewood-Richardson rule

Dedicated to the memory of Alain Lascoux

Florent Hivert

LRI / Université Paris Sud 11 / CNRS

Mars 2015

Outline

Algebraic Combinatorics

Going back and forth between

- algebraic identities
- algorithms

Today: Proving a multiplication rule of symmetric polynomials by executing the Robinson-Schensted algorithm on the concatenation of two words.

Works because of some associativity property: the plactic monoid.

Algebraic Combinatorics

Going back and forth between

- algebraic identities
- algorithms

Today: Proving a multiplication rule of symmetric polynomials by executing the Robinson-Schensted algorithm on the concatenation of two words.

Works because of some associativity property: the plactic monoid.

Algebraic Combinatorics

Going back and forth between

- algebraic identities
- algorithms

Today: Proving a multiplication rule of symmetric polynomials by executing the Robinson-Schensted algorithm on the concatenation of two words.

Works because of some associativity property: the plactic monoid.

Symmetric Polynomials

n-variables : $\mathbb{X}_n := \{x_0, x_1, \dots x_{n-1}\}$.
Polynomials in $\mathbb{X}$: $P(\mathbb{X}) = P(x_0, x_1, \dots, x_{n-1})$; ex: $3x_0^3x_2 + 5x_1x_2^4$.

Definition (Symmetric polynomial)

A polynomial is symmetric if it is invariant under any permutation of the variables: for all $\sigma \in \mathfrak{S}_n$,

$$P(x_0, x_1, \dots, x_{n-1}) = P(x_{\sigma(0)}, x_{\sigma(1)}, \dots, x_{\sigma(n-1)})$$

$$P(a, b, c) = a^2b + a^2c + b^2c + ab^2 + ac^2 + bc^2$$

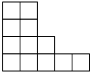
$$Q(a, b, c) = 5abc + 3a^2bc + 3ab^2c + 3abc^2$$

Integer Partitions

different ways of decomposing an integer $n \in \mathbb{N}$ as a sum:

$$5 = 5 = 4{+}1 = 3{+}2 = 3{+}1{+}1 = 2{+}2{+}1 = 2{+}1{+}1{+}1 = 1{+}1{+}1{+}1{+}1$$

Partition $\lambda := (\lambda_0 \geq \lambda_1 \geq \cdots \geq \lambda_l > 0)$.
$|\lambda| := \lambda_0 + \lambda_1 + \cdots + \lambda_l$ et $\ell(\lambda) := l$.

Ferrer's diagram of a partitions : $(5, 3, 2, 2) \leftrightarrow$ 

```
Fixpoint is_part sh := (* Predicate *)
  if sh is sh0 :: sh'
  then (sh0 >= head 1 sh') && (is_part sh')
  else true.

(* Boolean reflection lemma *)
Lemma is_partP sh : reflect
  (last 1 sh != 0 /\ forall i, (nth 0 sh i) >= (nth 0 sh i.+1))
  (is_part sh).
```

Schur symmetric polynomials

Definition (Schur symmetric polynomial)

Partition $\lambda := (\lambda_0 \geq \lambda_1 \geq \cdots \geq \lambda_{l-1})$ *with* $l \leq n$*; set* $\lambda_i := 0$ *for* $i \geq l$.

$$s_\lambda = \frac{\sum_{\sigma\in\mathfrak{S}_n} \mathrm{sign}(\sigma)\mathbb{X}_n^{\sigma(\lambda+\rho)}}{\prod_{0\leq i<j<n}(x_j - x_i)} = \frac{\begin{vmatrix} x_1^{\lambda_{n-1}+0} & x_2^{\lambda_{n-1}+0} & \dots & x_n^{\lambda_{n-1}+0} \\ x_1^{\lambda_{n-2}+1} & x_2^{\lambda_{n-2}+1} & \dots & x_n^{\lambda_{n-2}+1} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{\lambda_1+n-2} & x_2^{\lambda_1+n-2} & \dots & x_n^{\lambda_1+n-2} \\ x_1^{\lambda_0+n-1} & x_2^{\lambda_0+n-1} & \dots & x_n^{\lambda_0+n-1} \end{vmatrix}}{\begin{vmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \\ x_1^2 & x_2^2 & \dots & x_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{n-1} & x_2^{n-1} & \dots & x_n^{n-1} \end{vmatrix}}$$

$$s_{(2,1)}(a,b,c) = a^2b + ab^2 + a^2c + 2abc + b^2c + ac^2 + bc^2$$

Littlewood-Richardson coefficients

Proposition

The family $(s_\lambda(\mathbb{X}_n))_{\ell(\lambda)\leq n}$ *is a (linear) basis of the ring of symmetric polynomials on* $\mathbb{X}_n$*.*

Definition (Littlewood-Richardson coefficients)

Coefficients $c^\nu_{\lambda,\mu}$ *of the expansion of the product:*

$$s_\lambda s_\mu = \sum_\nu c^\nu_{\lambda,\mu} s_\nu \,.$$

Fact: $s_\lambda(\mathbb{X}_{n-1}, x_n := 0) = s_\lambda(\mathbb{X}_{n-1})$ if $\ell(\lambda) < n$ else 0.
Consequence: $c^\nu_{\lambda,\mu}$ are independant of the number of variables.

History

- stated (1934) by D. E. Littlewood and A. R. Richardson, wrong proof, wrong example.
- Robinson (1938), wrong completed proof.
- First correct proof: Schützenberger (1977).
- Dozens of thesis and paper about its proof (Zelevinsky 1981, Macdonald 1995, Gasharov 1998, Duchamp-H-Thibon 2001, van Leeuwen 2001, Stembridge 2002).

Wikipedia*: The Littlewood–Richardson rule is notorious for the number of errors that appeared prior to its complete, published proof. Several published attempts to prove it are incomplete, and it is particularly difficult to avoid errors when doing hand calculations with it: even the original example in D. E. Littlewood and A. R. Richardson (1934) contains an error.*

Applications

- $\#P$-hard problem (possibly, invariant theory, $P \neq NP$).
- multiplicity of induction or restriction of irreducible representations of the symmetric groups;
- multiplicity of the tensor product of the irreducible representations of linear groups;
- Geometry: mumber of intersection in a grassmanian variety, cup product of the cohomology;
- Horn problem: eigenvalues of the sum of two hermitian matrix;
- Extension of abelian groups (Hall algebra);
- Application in quantum physics (spectrum rays of the Hydrogen atoms);

Young Tableau

Definition

- *Filling of a partition shape;*
- *non decreasing along the rows;*
- *strictly increasing along the columns.*
- *row reading*

$$\begin{array}{|c|c|c|c|c|c|c|} \hline d & d & e \\ \cline{1-3} b & c & c & c & d \\ \cline{1-5} a & a & a & b & b & d & e \\ \hline \end{array} = ddebcccdaaabbde$$

$$\begin{array}{|c|c|c|c|c|} \hline 5 \\ \cline{1-1} 2 & 6 & 9 \\ \cline{1-3} 1 & 3 & 4 & 7 & 8 \\ \hline \end{array} = 526913478$$

Ordered types

I'm using SSReflect class/mixin/canonical paradigm.

```
Definition axiom T (r : rel T) :=
    [/\ reflexive r, antisymmetric r, transitive r &
        (forall m n : T, (r m n) || (r n m))].

Record mixin_of T := Mixin { r : rel T; x : T; _ : axiom r }.
Record class_of T := Class {base : Countable.class_of T; mixin : mixin_of T}.
Structure type := Pack {sort; _ : class_of sort; _ : Type}.
Notation ordType := type.

Definition leqX_op T := Order.r (Order.mixin (Order.class T)).

Delimit Scope ord_scope with Ord.
Open Scope ord_scope.
Notation "m <= n" := (leqX_op m n) : ord_scope.
```

Young Tableau: formal definition

```
Variable T : ordType.
Notation Z := (inhabitant T).
Notation is_row r := (sorted (@leqX_op T) r).

Definition dominate (u v : seq T) :=
  ((size u) <= (size v)) &&
   (all (fun i => (nth Z u i > nth Z v i)%Ord) (iota 0 (size u))).

Lemma dominateP u v :
  reflect ((size u) <= (size v) /\
           forall i, i < size u -> (nth Z u i > nth Z v i)%Ord)
          (dominate u v).

Fixpoint is_tableau (t : seq (seq T)) :=
  if t is t0 :: t' then
    [&& (t0 != [::]), is_row t0,
      dominate (head [::] t') t0 & is_tableau t']
  else true.

Definition to_word t := flatten (rev t).
```

Combinatorial definition of Schur functions

Definition

$$s_\lambda(\mathbb{X}) = \sum_{T \text{ tableaux of shape } \lambda} \mathbb{X}^T$$

where $\mathbb{X}^T$ is the product of the elements of T.

$$s_{(2,1)}(a,b,c) = a^2b + ab^2 + a^2c + \quad 2abc \quad + b^2c + ac^2 + bc^2$$

$$s_{(2,1)}(a,b,c) = \begin{array}{|c|c}\hline b & \\ \hline a & \multicolumn{1}{c|}{a} \\ \hline\end{array} + \begin{array}{|c|c}\hline b & \\ \hline a & \multicolumn{1}{c|}{b} \\ \hline\end{array} + \begin{array}{|c|c}\hline c & \\ \hline a & \multicolumn{1}{c|}{a} \\ \hline\end{array} + \begin{array}{|c|c}\hline b & \\ \hline a & \multicolumn{1}{c|}{c} \\ \hline\end{array} + \begin{array}{|c|c}\hline c & \\ \hline a & \multicolumn{1}{c|}{b} \\ \hline\end{array} + \begin{array}{|c|c}\hline c & \\ \hline b & \multicolumn{1}{c|}{b} \\ \hline\end{array} + \begin{array}{|c|c}\hline c & \\ \hline a & \multicolumn{1}{c|}{c} \\ \hline\end{array} + \begin{array}{|c|c}\hline c & \\ \hline b & \multicolumn{1}{c|}{c} \\ \hline\end{array}$$

```
Variable R : comRingType.
Fixpoint multpoly n :=
  if n is n'.+1 then poly_comRingType (multpoly n') else R.

(* 'I_n is the finite type {0, ..., n-1} *)
Definition vari n (i : 'I_n) : multpoly n. (* i-th variable *)
Proof.
  elim: n i => [//= | n IHn] i; first by apply 1.
  case (unliftP 0 i) => /= [j |] Hi.
  - by apply (polyC (IHn j)).
  - by apply 'X.
Defined.

(* set of row reading of a tableaux of a given shape *)
Definition tabwordshape (sh : seq nat) :=
  [set t : (sumn sh).-tuple 'I_n |
     (to_word (RS t) == t) && (shape (RS (t)) == sh)].

Definition commword (w : seq 'I_n) : multpoly n := \prod_(i <- w) vari i.
Definition polyset d (s : {set d.-tuple 'I_n}) := \sum_(w in s) commword w.

Definition Schur_pol (sh : seq nat) := polyset R (tabwordshape sh).
```

Yamanouchi Words

$|w|_x$: number of occurrence of x in w.

Definition

Sequence $w_0, \dots, w_{l-1}$ of integers such that for all k, i,

$$\left|w_i, \dots, w_{l-1}\right|_k \geq \left|w_i, \dots, w_{l-1}\right|_{k+1}$$

Equivalently $(|w|_i)_{i \leq \max(w)}$ is a partition and $w_1, \dots, w_{l-1}$ is also Yamanouchi.

$$(), 0, 00, 10, 000, 100, 010, 210,$$

$$0000, 1010, 1100, 0010, 0100, 1000, 0210, 2010, 2100, 3210$$

Yamanouchi words in Coq

```
(* incr_nth s i == the nat sequence s with item i incremented (s is *)
(*                 first padded with 0's to size i+1, if needed).   *)

  Fixpoint shape_rowseq s :=
    if s is s0 :: s'
    then incr_nth (shape_rowseq s') s0
    else [::].

  Definition shape_rowseq_count :=
    [fun s => [seq (count_mem i) s | i <- iota 0 (foldr maxn 0 (map S s))]].

  Lemma shape_rowseq_countE : shape_rowseq_count =1 shape_rowseq.

  Fixpoint is_yam s :=
    if s is s0 :: s'
    then is_part (shape_rowseq s) && is_yam s'
    else true.
```

The LR Rule at last !

Theorem (Littlewood-Richardson rule)

$c^{\nu}_{\lambda,\mu}$ is the number of (skew) tableaux of shape the difference ν/λ, whose row reading is a Yamanouchi word of evaluation μ.

$$C^{5432}_{331,421} = 3 \qquad \begin{array}{|c|c|c|c|c|} 1 & 2 & & & \\ & 0 & 0 & & \\ & & & 1 & \\ & & & 0 & 0 \end{array} \qquad \begin{array}{|c|c|c|c|c|} 0 & 2 & & & \\ & 0 & 1 & & \\ & & & 1 & \\ & & & 0 & 0 \end{array} \qquad \begin{array}{|c|c|c|c|c|} 0 & 1 & & & \\ & 0 & 2 & & \\ & & & 1 & \\ & & & 0 & 0 \end{array}$$

$$C^{7542}_{431,4321} = 4 \qquad \begin{array}{ccccccc} 2 & 3 & & & & & \\ & 1 & 1 & 2 & & & \\ & & & 0 & 1 & & \\ & & & & 0 & 0 & 0 \end{array} \qquad \begin{array}{ccccccc} 2 & 3 & & & & & \\ & 0 & 1 & 2 & & & \\ & & & 1 & 1 & & \\ & & & & 0 & 0 & 0 \end{array} \qquad \begin{array}{ccccccc} 1 & 3 & & & & & \\ & 0 & 2 & 2 & & & \\ & & & 1 & 1 & & \\ & & & & 0 & 0 & 0 \end{array} \qquad \begin{array}{ccccccc} 0 & 3 & & & & & \\ & 1 & 2 & 2 & & & \\ & & & 1 & 1 & & \\ & & & & 0 & 0 & 0 \end{array}$$

$$C^{7542}_{4321,431} = 4 \qquad \begin{array}{cccccc} 2 & & & & & \\ & 1 & 1 & & & \\ & & 0 & 1 & & \\ & & & 0 & 0 & 0 \end{array} \qquad \begin{array}{cccccc} 1 & & & & & \\ & 1 & 2 & & & \\ & & 0 & 1 & & \\ & & & 0 & 0 & 0 \end{array} \qquad \begin{array}{cccccc} 1 & & & & & \\ & 0 & 2 & & & \\ & & 1 & 1 & & \\ & & & 0 & 0 & 0 \end{array} \qquad \begin{array}{cccccc} 0 & & & & & \\ & 1 & 2 & & & \\ & & 1 & 1 & & \\ & & & 0 & 0 & 0 \end{array}$$

Outline of a proof

Lascoux, Leclerc and Thibon *The Plactic monoid*, *in* M. Lothaire, Algebraic combinatorics on words, Cambridge Univ. Press.

1. increasing subsequences and Schensted's algorithms;
2. Robinson-Schensted correspondance : a bijection;
3. Green's invariants: computing the maximum sum of the length of k disjoint non-decreassing subsequences;
4. Knuth relations, the plactic monoïd;
5. Green's invariants are plactic invariants: Equivalence between RS and plactic;
6. standardization; symmetry of RS;
7. Lifting to non commutative polynomials : Free quasi-symmetric function and shuffle product;
8. non-commutative lifting the LR-rule : The free/tableau LR-rule;
9. Back to Yamanouchi words: a final bijection.

The longest increasing subsequence problem

Some increasing subsequences:

$$ababcabbadbab$$
$$ababcabbadbab$$
$$ababcabbadbab$$

Problem (Schensted)

Given a finite sequence w, compute the maximum length of a increasing subsequence.

Schensted's algorithm

Algorithm

Start with an empty row r, insert the letters l of the word one by one from left to right by the following rule:

- *replace the first letter strictly larger that l by l;*
- *append l to r if there is no such letter.*

Insertion of $ababcabbadbab\ w$

$$\emptyset \xrightarrow{a} \boxed{a} \xrightarrow{b} \boxed{a|b} \xrightarrow{a} \boxed{a|a} \xrightarrow{b} \boxed{a|a|b} \xrightarrow{c} \boxed{a|a|b|c} \xrightarrow{a}$$

$$\boxed{a|a|a|c} \xrightarrow{b} \boxed{a|a|a|b} \xrightarrow{b} \boxed{a|a|a|b|b} \xrightarrow{a}$$

$$\boxed{a|a|a|a|b} \xrightarrow{d} \boxed{a|a|a|a|b|d} \xrightarrow{b} \boxed{a|a|a|a|b|b} \xrightarrow{a}$$

$$\boxed{a|a|a|a|a|b} \xrightarrow{b} \boxed{a|a|a|a|a|b|b}$$

Schensted's specification

Warning: list index start from 0.

Theorem (Schensted 1961)

The i-th entry $r[i+1]$ of the row r is the smallest letter which ends a increasing subsequence of length i.

$$\text{Schensted}(\ ababcabbadbab\ ) = \boxed{a}\boxed{a}\boxed{a}\boxed{a}\boxed{a}\boxed{b}\boxed{b}$$

```
Fixpoint insrow r l : seq T :=
  if r is l0 :: r then
    if (l < l0)%Ord then l :: r
    else l0 :: (insrow r l)
  else [:: l].

Fixpoint inspos r (l : T) : nat :=
  if r is l0 :: r' then
    if (l < l0)%Ord then 0
    else (inspos r' l).+1
  else 0.

Definition ins r l := set_nth l r (inspos r l) l.

Lemma insE r l : insmin r l = ins r l.
Lemma insrowE r l : insmin r l = insrow r l.
```

```
(* rev == list reversal,   rcons s x == the sequence s, followed by x *)
(* subseq s1 s2 == s1 is a subsequence of s2                          *)
Fixpoint Sch_rev w := if w is l0 :: w' then ins (Sch_rev w') l0 else [::].
Definition Sch w := Sch_rev (rev w).

Lemma is_row_Sch w : is_row (Sch w).

Definition subseqrow s w := subseq s w && is_row s.
Definition subseqrow_n s w n := [&& subseq s w , (size s == n) & is_row s].

Theorem Sch_exists w i :
  i < size (Sch w) ->
  exists s : seq T, (last Z s == nth Z (Sch w) i) && subseqrow_n s w i.+1.
(* Induction elimining the last letter : elim/last_ind: w *)
Theorem Sch_leq_last w s si :
  subseqrow (rcons s si) w ->
  size s < size (Sch w) /\ (nth Z (Sch w) (size s) <= si)%Ord.
(* Induction elimining the last letter : elim/last_ind: w *)

Theorem Sch_max_size w :
  size (Sch w) = \max_(s : subseqs w | is_row s) size s.
```

The "apply it recursively" rule

If you have a great idea,
apply it recursively
and you'll get
an even greater idea !

The "apply it recursively" rule

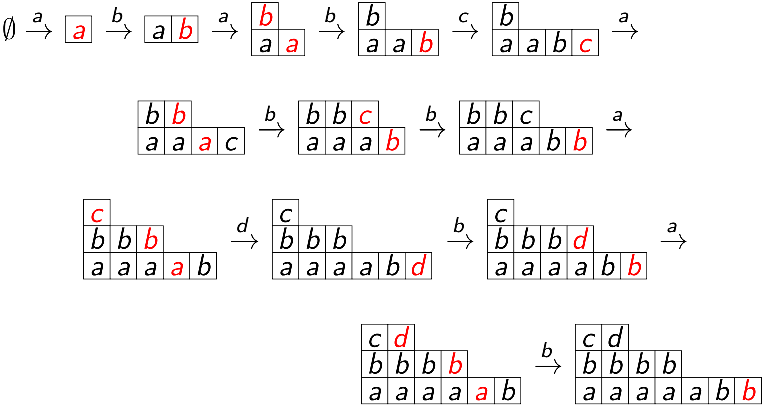
If you have a great idea,
apply it recursively
and you'll get
an even greater idea !

The “apply it recursively” rule

If you have a great idea,
apply it recursively
and you’ll get
an even greater idea !

The Robinson-Schensted's correspondence

Bumping the letters: when a letter is replaced in Schensted algorithm, **insert it in a next row** (placed on top in the drawing).



```
Definition bump r l := (l < (last l r))%Ord.

Fixpoint bumprow r l : (option T) * (seq T) :=
  if r is l0 :: r then
    if (l < l0)%Ord then (Some l0, l :: r)
    else let: (lr, rr) := bumprow r l in (lr, l0 :: rr)
  else (None, [:: l]).

Fixpoint instab t l : seq (seq T) :=
  if t is t0 :: t' then
    let: (lr, rr) := bumprow t0 l in
    if lr is Some ll then rr :: (instab t' ll) else rr :: t'
  else [:: [:: l]].

Fixpoint RS_rev w : seq (seq T) :=
  if w is w0 :: wr then instab (RS_rev wr) w0 else [::].
Definition RS w := RS_rev (rev w).

Lemma bump_dominate r1 r0 l : is_row r0 -> is_row r1 -> bump r0 l ->
  dominate r1 r0 -> dominate (ins r1 (bumped r0 l)) (ins r0 l).
Theorem is_tableau_instab t l : is_tableau t -> is_tableau (instab t l).
Theorem is_tableau_RS w : is_tableau (RS w).
```

Going back

If we remember which cell was added we can recover the letter and the previous tableau:

$$\ydiagram{1,4,6} \xrightarrow{?} \begin{ytableau} c & d \\ b & b & b & b \\ a & a & a & a & a & b \end{ytableau}$$

```
RSmap : forall T : ordType, seq T -> seq (seq T) * seq nat
RSmapinv2 : forall T : ordType, seq (seq T) * seq nat -> seq T

Definition is_RSpair pair := let: (P, Q) := pair
  in [&& is_tableau P, is_yam Q & (shape P == shape_rowseq Q)].
Theorem RSmap_spec w : is_RSpair (RSmap w).

Theorem RS_bij_1 w : RSmapinv2 (RSmap w) = w.
Theorem RS_bij_2 pair : is_RSpair pair -> RSmap (RSmapinv2 pair) = pair.
```

Going back

If we remember which cell was added we can recover the letter and the previous tableau:

$$\begin{array}{|c|c|c|c|c|c|}\hline c \\ \cline{1-4} b & b & b & d \\ \hline a & a & a & a & b & b \\ \hline \end{array} \xrightarrow{a} \begin{array}{|c|c|c|c|c|c|}\hline c & d \\ \cline{1-4} b & b & b & b \\ \hline a & a & a & a & a & b \\ \hline \end{array}$$

```
RSmap : forall T : ordType, seq T -> seq (seq T) * seq nat
RSmapinv2 : forall T : ordType, seq (seq T) * seq nat -> seq T

Definition is_RSpair pair := let: (P, Q) := pair
  in [&& is_tableau P, is_yam Q & (shape P == shape_rowseq Q)].
Theorem RSmap_spec w : is_RSpair (RSmap w).

Theorem RS_bij_1 w : RSmapinv2 (RSmap w) = w.
Theorem RS_bij_2 pair : is_RSpair pair -> RSmap (RSmapinv2 pair) = pair.
```

Robinson-Schensted's bijection (Yamanouchi version)

$$\emptyset,\emptyset \xrightarrow{a} \begin{array}{|c|}\hline {\color{red}a} \\ \hline\end{array},0 \xrightarrow{b} \begin{array}{|c|c|}\hline a & {\color{red}b} \\ \hline\end{array},00 \xrightarrow{a} \begin{array}{|c|c|}\cline{1-1} {\color{red}b} \\ \hline a & {\color{red}a} \\ \hline\end{array},100 \xrightarrow{b}$$

$$\begin{array}{|c|c|c|}\cline{1-1} b \\ \hline a & a & {\color{red}b} \\ \hline\end{array},0100 \xrightarrow{c} \begin{array}{|c|c|c|c|}\cline{1-1} b \\ \hline a & a & b & {\color{red}c} \\ \hline\end{array},00100 \xrightarrow{a} \begin{array}{|c|c|c|c|}\cline{1-2} b & {\color{red}b} \\ \hline a & a & {\color{red}a} & c \\ \hline\end{array},100100 \xrightarrow{b}$$

$$\begin{array}{|c|c|c|c|}\cline{1-3} b & b & {\color{red}c} \\ \hline a & a & a & {\color{red}b} \\ \hline\end{array},1100100 \xrightarrow{b} \begin{array}{|c|c|c|c|c|}\cline{1-3} b & b & c \\ \hline a & a & a & b & {\color{red}b} \\ \hline\end{array},01100100 \xrightarrow{a}$$

$$\begin{array}{|c|c|c|c|c|}\cline{1-1} {\color{red}c} \\ \cline{1-3} b & b & {\color{red}b} \\ \hline a & a & a & {\color{red}a} & b \\ \hline\end{array},201100100 \xrightarrow{d} \begin{array}{|c|c|c|c|c|c|}\cline{1-1} c \\ \cline{1-3} b & b & b \\ \hline a & a & a & a & b & {\color{red}d} \\ \hline\end{array},0201100100$$

Robinson-Schensted's bijection (Tableau version)

$$\emptyset, \emptyset \xrightarrow{a} \ytableaushort{a}, \ytableaushort{0} \xrightarrow{b} \ytableaushort{ab}, \ytableaushort{01} \xrightarrow{a} \ytableaushort{b,aa}, \ytableaushort{2,01} \xrightarrow{b}$$

$$\ytableaushort{b,aab}, \ytableaushort{2,013} \xrightarrow{c} \ytableaushort{b,aabc}, \ytableaushort{2,0134} \xrightarrow{a} \ytableaushort{bb,aaac}, \ytableaushort{25,0134} \xrightarrow{b}$$

$$\ytableaushort{bbc,aaab}, \ytableaushort{256,0134} \xrightarrow{b} \ytableaushort{bbc,aaabb}, \ytableaushort{256,01347} \xrightarrow{a}$$

$$\ytableaushort{c,bbb,aaaab}, \ytableaushort{8,256,01347} \xrightarrow{d} \ytableaushort{c,bbb,aaaabd}, \ytableaushort{8,256,013479}$$

Idea of the proof: the non commutative lifting

Fix the second tableau Q in the bijection

$$L_Q := \{w \mid RS(w)_2 = Q\}$$

Then clearly:

$$S_{\mathrm{shape}(Q)} = \sum_{w \in L_Q} \mathrm{comm}(w)$$

The crucial fact is:

Theorem

There exists an explicit set $\Omega(Q, R)$ such that

$$L_Q L_R = \sum_{T \in \Omega(Q,R)} L_T.$$

Idea of the proof: the non commutative lifting

Fix the second tableau Q in the bijection

$$L_Q := \{w \mid RS(w)_2 = Q\}$$

Then clearly:

$$S_{\mathrm{shape}(Q)} = \sum_{w \in L_Q} \mathrm{comm}(w)$$

The crucial fact is:

Theorem

There exists an explicit set $\Omega(Q, R)$ such that

$$L_Q L_R = \sum_{T \in \Omega(Q,R)} L_T.$$

Idea of the proof: the non commutative lifting

```
Variable R : comRingType.
Fixpoint multpoly n :=
  if n is n'.+1 then poly_comRingType (multpoly n') else R.

Definition vari n (i : 'I_n) : multpoly n.
Definition commword (w : seq 'I_n) : multpoly n := \prod_(i <- w) vari i.

Definition polyset d (s : {set d.-tuple 'I_n}) := \sum_(w in s) commword w.
Definition catset d1 d2 (s1 : {set d1.-tuple 'I_n}) (s2 : {set d2.-tuple 'I_n})
           : {set (d1 + d2).-tuple 'I_n} :=
 [set cat_tuple w1 w2 | w1 in s1, w2 in s2].

Lemma multcatset d1 d2 (s1 : {set d1.-tuple 'I_n}) (s2 : {set d2.-tuple 'I_n})
  polyset s1 * polyset s2 = polyset (catset s1 s2).
```

The free LR-rule

```
Definition freeSchur (Q : stdtabn d) :=
  [set t : d.-tuple 'I_n | (RStabmap t).2 == Q].

Lemma Schur_freeSchurE d (Q : stdtabn d) :
  Schur (shape_deg Q) = polyset R (freeSchur Q).

Definition predLRTriple (t1 t2 : seq (seq nat)) : pred (t : (seq (seq nat))).

Variables (d1 d2 : nat).
Variables (Q1 : stdtabn d1) (Q2 : stdtabn d2).
Definition LR_support :=
  [set Q : stdtabn (d1 + d2) | predLRTriple Q1 Q2 Q ].

Lemma catset_LR_rule :
  catset (freeSchur Q1) (freeSchur Q2) =
   \bigcup_(Q in LR_support) (freeSchur Q).
```

Proving the free LR-rule

```
Definition freeSchur (Q : stdtabn d) :=
  [set t : d.-tuple 'I_n | (RStabmap t).2 == Q].

Lemma catset_LR_rule :
  catset (freeSchur Q1) (freeSchur Q2) =
    \bigcup_(Q in LR_support) (freeSchur Q).
```

One needs to understand the execution of the RS algorithms on the concatenation of two words !

Question ?

What does the Robinson-Schensted algorithm compute ?

Green's Theorem

disjoint support increasing subsequences:

$$ababcabbadbab$$

Theorem

For any word w, and integer k

- *The sum of the length of the k-first row of $RS(w)$ is the maximum sum of the length of k disjoint support increasing subsequences of w;*
- *The sum of the length of the k-first column of $RS(w)$ is the maximum sum of the length of k disjoint support strictly decreasing subsequences of w.*

Green's Theorem

disjoint support increasing subsequences:

$$ababcabbadbab$$

Theorem

For any word w, and integer k

- *The sum of the length of the k-first row of $RS(w)$ is the maximum sum of the length of k disjoint support increasing subsequences of w;*
- *The sum of the length of the k-first column of $RS(w)$ is the maximum sum of the length of k disjoint support strictly decreasing subsequences of w.*

Green's Theorem

disjoint support increasing subsequences:

$$ababcabbadbab$$

Theorem

For any word w, and integer k

- *The sum of the length of the k-first row of $RS(w)$ is the maximum sum of the length of k disjoint support increasing subsequences of w;*
- *The sum of the length of the k-first column of $RS(w)$ is the maximum sum of the length of k disjoint support strictly decreasing subsequences of w.*

```
(* From mathcomp: cover P     == the union of the set of sets P. *)
(*                 trivIset P <=> the elements of P are pairwise disjoint.  *)
Definition cover P := \bigcup_(B in P) B.
Definition trivIset P := \sum_(B in P) #|B| == #|cover P|.

Variable N : nat. Variable wt : N.-tuple Alph.
Definition extractpred (P : pred 'I_N) := [seq tnth wt i | i <- enum P].
Definition extract := [fun s : {set 'I_N} => extractpred wt (mem s)].

Variable comp : rel Alph. Hypothesis Hcomp : transitive comp.
Definition is_seq := [fun r => (sorted comp r)].
Definition ksupp k (P : {set {set 'I_N}}) :=
  [&& #|P| <= k, trivIset P & [forall (s | s \in P), is_seq (extract s)]].

Definition green_rel_t k := \max_(P | ksupp k P) #|cover P|].
Definition green_rel u := [fun k => green_rel_t comp (in_tuple u) k].

Variable Alph : ordType.
Definition greenRow := green_rel (leqX Alph).
Definition greenCol := green_rel (gtnX Alph).
```

Green's Theorem

```
(* take n s == the sequence containing only the first n items of s *)
(*              (or all of s if size s <= n).                        *)

Definition part_sum s k := (\sum_(l <- (take k s)) l).

Corollary greenRow_RS k w :
  greenRow w k = part_sum (shape (RS w)) k.

(* conj_part : partition conjugate *)
Corollary greenCol_RS k w :
  greenCol w k = part_sum (conj_part (shape (RS w))) k.
```

Proof:

- Show that theorems `green(Row|Col)_RS` hold whenever w is the row reading of a tableau;
- Knuth's relations: given a tableau t, describe all the word w such that $RS(w) = t$;
- Show that Green's invariant are indeed invariants.

Green's Theorem

```
(* take n s == the sequence containing only the first n items of s *)
(*            (or all of s if size s <= n).                         *)

Definition part_sum s k := (\sum_(l <- (take k s)) l).

Corollary greenRow_RS k w :
  greenRow w k = part_sum (shape (RS w)) k.

(* conj_part : partition conjugate *)
Corollary greenCol_RS k w :
  greenCol w k = part_sum (conj_part (shape (RS w))) k.
```

Proof:

- Show that theorems `green(Row|Col)_RS` hold whenever w is the row reading of a tableau;
- Knuth's relations: given a tableau t, describe all the word w such that $RS(w) = t$;
- Show that Green's invariant are indeed invariants.

Step 1: Green's invariant of a tableau

- Green Columns : Upper bound using concatenation;
- Green Rows : Upper bound intersection with columns;
- Construction of an explicit k-sub sequence.

```
(* Bound from concatenation *)
Lemma green_rel_cat k v w :
   green_rel (v ++ w) k <= green_rel v k + green_rel w k.

Lemma greenCol_inf_tab k t :
  is_tableau t -> greenCol (to_word t) k <= \sum_(l <- (shape t)) minn l k.

(* Note : conj_part p == conjugate partition of p *)
Definition part_sum s k := (\sum_(l <- (take k s)) l).
Lemma sum_conj sh k : \sum_(l <- sh) minn l k = part_sum (conj_part sh) k.
```

Construction of an explicit k-sub sequence:

A dependant type nightmare !

```
(* lshift n j == the i : 'I_(m + n) with value j : 'I_m.        *)
(* rshift m k == the i : 'I_(m + n) with value m + k, k : 'I_n. *)
Let sym_cast m n (i : 'I_(m + n)) : 'I_(n + m) := cast_ord (addnC m n) i.
Definition shiftset s0 sh :=
  [fun s : {set 'I_(sumn sh)} =>
     ((@sym_cast _ _) \o (@lshift (sumn sh) s0)) @: s].

Fixpoint shrows sh : seq {set 'I_(sumn sh)} :=
  if sh is s0 :: sh then
    [set ((@sym_cast _ _) \o (@rshift (sumn sh) s0)) i | i in 'I_s0] ::
    map (@shiftset s0 sh) (shrows sh)
  else [::].

Lemma lcast_com :
  (cast_ord (size_to_word (t0 :: t)))
    \o (@sym_cast _ _) \o (@lshift (sumn (shape t)) (size t0))
  =1  linj_rec \o (cast_ord (size_to_word t)).
```

```
Lemma size_to_word T (t : seq (seq T)) : size_tab t = size (to_word t)

Let cast_set_tab t :=
  [fun s : {set 'I_(sumn (shape t))} => (cast_ord (size_to_word t)) @: s].

Definition tabrows t := map (cast_set_tab t) (shrows (shape t)).
Definition tabrowsk t := [fun k => take k (tabrows t)].

Lemma ksupp_leqX_tabrowsk k t : is_tableau t ->
  ksupp (leqX Alph) (in_tuple (to_word t)) k [set s | s \in (tabrowsk t k)].

Lemma scover_tabcolsk k t : is_part (shape t) ->
  scover [set s | s \in (tabcolsk t k)] = \sum_(l <- (shape t)) minn l k.
```

Step 2: Knuth relation

$$acb \equiv cab \quad \text{if } a \leq b < c$$

$$bac \equiv bca \quad \text{if } a < b \leq c$$

```
Definition plact1 :=
  fun s => match s return seq word with
    | [:: a; c; b] =>
        if (a <= b < c)%Ord then [:: [:: c; a; b]] else [::]
    | _ => [::]
  end.
[...]

Definition plactrule :=
  [fun s => plact1 s ++ plact1i s ++ plact2 s ++ plact2i s].

Lemma plactrule_sym u v : v \in (plactrule u) -> u \in (plactrule v).
Lemma plact_homog : forall u : word, all (perm_eq u) (plactrule u).
```

Step 2: The plactic congruence

```
Definition congruence_rel (r : rel word) :=
  forall a b1 c b2, r b1 b2 -> r (a ++ b1 ++ c) (a ++ b2 ++ c).

CoInductive Generated_EquivCongruence (grel : rel word) :=
  GenCongr : equivalence_rel grel ->
             congruence_rel grel ->
             ( forall u v, v \in rule u -> grel u v ) ->
             ( forall r : rel word,
                   equivalence_rel r -> congruence_rel r ->
                   (forall x y, y \in rule x -> r x y) ->
                   forall x y, grel x y -> r x y
             ) -> Generated_EquivCongruence grel.

Theorem gencongrP : Generated_EquivCongruence gencongr.
Definition plactcongr := (gencongr plact_homog).

Lemma plactcongr_homog u v : v \in plactcongr u -> perm_eq u v.
Lemma size_plactcongr u v : v \in plactcongr u -> size u = size v.

Notation "a =Pl b" := (plactcongr a b) (at level 70).
```

Step 2: plactic congruence and RS algorithm

$$\boxed{a}\boxed{a}\boxed{a}\boxed{c}\boxed{c}\boxed{d} \cdot b \equiv_{\mathrm{Pl}} c \cdot \boxed{a}\boxed{a}\boxed{a}\boxed{b}\boxed{c}\boxed{d}$$

$$aaaccd \cdot b \to aaaccdb \to aaaccbd \to aaacbcd \to$$
$$aacabcd \to acaabcd \to c \cdot acaabcd$$

```
Lemma congr_bump r l :
  r != [::] -> is_row r -> bump r l ->
  r ++ [:: l] =Pl [:: bumped r l] ++ ins r l.

Theorem congr_RS w : w =Pl (to_word (RS w)).
Corollary Sch_plact u v : RS u == RS v -> u =Pl v .
```

Step 3: Green's numbers are plactic invariants

$$u \equiv_{Pl} v \quad \Longrightarrow \quad \forall k \in \mathbb{N},\ \mathrm{Green}_k(u) = \mathrm{Green}_k(v)$$

```
Lemma ksupp_cast (T : ordType) R (w1 w2 : seq T) (H : w1 = w2) k Q :
  ksupp R (in_tuple w1) k Q ->
  ksupp R (in_tuple w2) k ((cast_set (eq_size H)) @: Q).

Definition ksupp_inj k (u1 : seq T1) (u2 : seq T2) :=
  forall s1, ksupp R1 (in_tuple u1) k s1 ->
    exists s2, (scover s1 == scover s2) && ksupp R2 (in_tuple u2) k s2.

Lemma leq_green k (u1 : seq T1) (u2 : seq T2) :
  ksupp_inj k u1 u2 -> green_rel R1 u1 k <= green_rel R2 u2 k.
```

Dependent type + too many hypothesis nightmare !

```
Record hypRabc (Alph : ordType) (R : rel Alph) (a b c : Alph) :
Type := HypRabc
  { Rtrans : transitive R;
    Hbc : is_true (R b c);
    Hba : is_true (~~ R b a);
    Hxba : forall l : Alph, R l a -> R l b;
    Hbax : forall l : Alph, R b l -> R a l }

SetContainingBothLeft.exists_Qy
     : forall (Alph : ordType) (R : rel Alph) (u v : seq Alph) (a b c : Alph),
       SetContainingBothLeft.hypRabc R a b c ->
       forall (k : nat) (P : {set {set 'I_(size (u ++ [:: b; a; c] ++ v))}}),
       ksupp R (in_tuple (u ++ [:: b; a; c] ++ v)) k P ->
       forall S0 : {set 'I_(size (u ++ [:: b; a; c] ++ v))},
       S0 \in P ->
       Swap.pos1 u (c :: v) b a \in S0 ->
       Ordinal (SetContainingBothLeft.u2lt u v a b c) \in S0 ->
       exists Q : {set {set 'I_(size ((u ++ [:: b]) ++ [:: c; a] ++ v))}},
         scover Q = scover P /\
         ksupp R (in_tuple ((u ++ [:: b]) ++ [:: c; a] ++ v)) k Q
```

Green's Plactic invariants

```
Theorem greenRow_invar_plactic u v :
  u =Pl v -> forall k, greenRow u k = greenRow v k.
Theorem greenCol_invar_plactic u v :
  u =Pl v -> forall k, greenCol u k = greenCol v k.

Corollary greenRow_RS k w :
  greenRow w k = part_sum (shape (RS w)) k.

Corollary greenCol_RS k w :
  greenCol w k = part_sum (conj_part (shape (RS w))) k.

Corollary plactic_shapeRS_row_proof u v :
  u =Pl v -> shape (RS u) = shape (RS v).
```

Main plactic theorem

```
(* rembig w : remove the last occurrence of the largest letter of w *)
(* append_nth T b i : append b to the i-th row of T *)

Theorem rembig_RS a v :
  {i | RS (a :: v) = append_nth (RS (rembig (a :: v))) (maxL a v) i}.
(* Proof by Bi-simulation *)

Theorem rembig_plactcongr u v : u =Pl v -> (rembig u) =Pl (rembig v).
(* Proof by induction on the rules *)

Lemma plactic_shapeRS u v : u =Pl v -> shape (RS u) = shape (RS v).
(* Proof by green invariant *)

Theorem plactic_RS u v : u =Pl v <-> RS u == RS v.
(* Induction removing the last occurrence of the largest letter *)
(* same shape => the largest letter is appended in the same row *)

(* Could be proved much earlier *)
Corollary RS_tabE (t : seq (seq Alph)) : is_tableau t -> RS (to_word t) = t.
```

… and then

7. standardization; symmetry of RS;
8. Lifting to non commutative polynomials : Free quasi-symmetric function and shuffle product;
9. non-commutative lifting the LR-rule : The free/tableau LR-rule;
10. Back to Yamanouchi words: a final bijection.

```
(* inverse word permutations *)
Definition linvseq s :=
  [fun t => all (fun i => nth (size s) t (nth (size t) s i) == i)
    (iota 0 (size s))].
Definition invseq s t := linvseq s t && linvseq t s.

Corollary invseqRSE s t :
  invseq s t -> RStabmap s = ((RStabmap t).2, (RStabmap t).1).
```

8 Lifting to non commutative polynomials : Free quasi-symmetric function and shuffle product;

```
Fixpoint shuffle_from_rec a u shuffu' v {struct v} :=
  if v is b :: v' then
    [seq a :: w | w <- shuffu' v] ++
    [seq b :: w | w <- shuffle_from_rec a u shuffu' v']
  else [:: (a :: u)].
Fixpoint shuffle u v {struct u} :=
  if u is a :: u' then
    shuffle_from_rec a u' (shuffle u') v
  else [:: v].
Definition shiftn    n := map (addn n).
Definition shsh u v := shuffle u (shiftn (size u) v).

(* This is essentially the product rule of FQSym *)
Theorem invstd_cat_in_shsh u v :
  invstd (std (u ++ v)) \in shsh (invstd (std u)) (invstd (std v)).
```

9 non-commutative lifting the LR-rule : The free/tableau LR-rule;

```
Record plactLRTriple t1 t2 t : Prop :=
  PlactLRTriple :
    forall p1 p2 p, RS p1 = t1 -> RS p2 = t2 -> RS p = t ->
                    p \in shsh p1 p2 -> plactLRTriple t1 t2 t.
(* reflect (plactLRTriple t1 t2 t) (predLRTriple t1 t2 t). *)
Definition LR_support :=
  [set Q : stdtabn (d1 + d2) | predLRTriple Q1 Q2 Q ].
Definition LR_coeff d1 d2
  (P1 : intpartn d1) (P2 : intpartn d2) (P : intpartn (d1 + d2)) :=
  #|[set Q | Q in (LR_support (hyper_std P1) (hyper_std P2)) &
                (shape Q == P)]|.

Theorem LR_coeffP d1 d2 (P1 : intpartn d1) (P2 : intpartn d2) :
  Schur P1 * Schur P2 =
   \sum_(P : intpartn (d1 + d2)) (Schur P) *+ LR_coeff P1 P2 P.
```

The final bijection

10 Back to Yamanouchi words: a final bijection:

Tableau version:

$$C^{7542}_{431,4321} = 4$$

15	17					
7	12	13	16			
4	5	6	8	14		
0	1	2	3	9	10	11

15	17					
7	8	12	16			
4	5	6	13	14		
0	1	2	3	9	10	11

12	17					
7	8	15	16			
4	5	6	13	14		
0	1	2	3	9	10	11

8	17					
7	12	15	16			
4	5	6	13	14		
0	1	2	3	9	10	11

Yamanouchi word version:

$$C^{7542}_{431,4321} = 4$$

2	3					
	1	1	2			
			0	1		
				0	0	0

2	3					
	0	1	2			
			1	1		
				0	0	0

1	3					
	0	2	2			
			1	1		
				0	0	0

0	3					
	1	2	2			
			1	1		
				0	0	0

+ a fast algorithm to enumerate those

Easy / Hard points

- SSReflect is good at automatically dealing with trivial cases (size $* = \frac{1}{3}$);
- Few missing basic lemmas;
- Lack of end user documentation for the class/mixin/canonical paradigm;
- Tuple and dependant types;
- More generally, I feel that SSReflect is too much oriented toward finite;
- Dealing with a lot of hypothesis.

Easy / Hard points

- SSReflect is good at automatically dealing with trivial cases (size $* = \frac{1}{3}$);
- Few missing basic lemmas;
- Lack of end user documentation for the class/mixin/canonical paradigm;
- Tuple and dependant types;
- More generally, I feel that SSReflect is too much oriented toward finite;
- Dealing with a lot of hypothesis.