

On Constructive Proofs for Termination of Normal Form for a Multivariate Polynomial with Respect to a Polynomial List

Sergei D. Meshveliani ¹

Ailamazyan Program Systems Institute of RAS,
Pereslavl-Zalessky, Russia.

“Computer Algebra” seminar, CMC faculty of Moscow State University,

February 16, 2022.

¹This investigation was partially supported by Russian Academy of Sciences, research project No AAAA-A19-119020690043-9.

Introduction

Let

$$R[x_1, \dots, x_n]$$

be a polynomial domain over a commutative ring R having a decidable equality R .

Apply a sparse representation of a polynomial as a list of *nonzero* monomials, ordered decreasingly by $<_e$ on the *exponents* (or “power products” or “tuples”).

Each exponent is represented as a tuple of n natural numbers.

Choosing an appropriate ordering $<_e$ it is often possible to reach the needed property for the chosen algorithm on polynomials.

Admissible Ordering on Exponents

- ▶ $\text{STO-}e$ — a strict total order ($\text{IsStrictTotalOrder } <_e$),
- ▶ $+_e \text{ mono}$ — $\forall e \in \text{PP } (e +_e)$ preserves $<_e$,
- ▶ $>_0$ — $\forall e \neq 0_e, e >_e 0_e$.

Examples of admissible orderings on \mathbb{N}^n :

lexicographic ordering, degree-lexicographic ordering, orderings presented by an integer matrix of weights (*flag*).

Abstract $<_e$ is needed.

Multiplying monomials corresponds to addition $+_e$ of the exponent vectors. For example,

$$(x^2y^0) * (x^1y^5) \quad \text{has the exponent } [3, 5].$$

Denote $<_p$ the *pointwise ordering* on exponents.

It is not total (for $n > 1$).

Due to $+_e$ *mono* for $<_e$, it holds

$$<_p \implies <_e \quad (I)$$

We assume here that any nonzero monomial coefficient is *invertible*. So, for any monomials m, m'

$$m \text{ divides } m' \iff \exp(m) \leq_p \exp(m')$$

And we say in such a case “ $\exp(m)$ divides $\exp(m')$ ”.

In such a case it holds $\exp(m) \leq_e \exp(m')$ — due to (I).

The notions of

leading monomial lm , *leading exponent* lExp ,

leading coefficient lc

are defined relatively to $<_e$.

Normal form Algorithm for Polynomials

This method is related to solving membership of a polynomial to an ideal:

$$f \in \text{Ideal}(g_s)$$

That is finding a decomposition

$$f = \phi_1 g_1 + \dots + \phi_m g_m$$

It is solved by the Gröbner basis algorithm [2].

The normal form algorithm NF for bringing a polynomial f to its normal form by the list g_s of polynomials is a one of the basic parts of the Gröbner basis method. It is as follows

$f = 0$ is put normal form.

For nonzero f , search for the first g in gs such that $e = \text{lExp}(g)$ divides $e' = \text{lExp}(f)$.

If there is not such, then f is normal.

If such g is found then $m = \text{lm}(g)$ divides $m' = \text{lm}(f)$.

Let m_q be the quotient monomial.

Then $\text{NF } f' \text{ } gs$ applies, where

$$f' = f - m_q g \quad (II)$$

$\text{lm}(f)$ is canceled at this step, and either $f' = 0$ or it holds

$$\text{lExp}(f') <_e \text{lExp}(f)$$

This is easy to derive from the axioms for $<_e$.

NF stops when none in gs reduces the current remainder f .

This produces a descending chain

$$e_0 >_e e_1 >_e \dots$$

of the leading exponents of the polynomial being reduced.

It is known that any admissible ordering $<_e$ on power products is *descending-halt*.

Hence NF is terminating.

On Constructive Proofs for NF Termination

We deal with *provable programming* of scientific computation. It is based, by default, on *constructive* formal proofs which are checked by a machine.

To implement algorithms and proofs we use the Agda language [1, 6].

The most popular proof for the descending-halt property of $<_e$ bases on Dickson's lemma.

A constructive definition of the descending-halt property is

$$\forall(f : \mathbb{N} \rightarrow X) \exists n, f(1 + n) \not\prec f(n).$$

But first, it is not simple to compose a full constructive proof for (Descending-halt $<_e$), and for Dickson's lemma too.

Second, usual mathematical practice of referring to the descending-halt property of a relation does not work in the constructive proof program systems based on the *dependent types* theory (Coq, Agda).

Instead the best termination proofs need there a reference to a proof for a certain

well-founded

property of a relation. It is a bit stronger than descending-halt.

This restriction is somewhat fundamental for the above programming systems.

For example, try to implement NF as

```
<e-descending-halt : DescendingHalt _<e_
```

```
<e-descending-halt = <proof>
```

```
NF : Pol -> List Pol -> Pol
```

```
NF f gs =
```

```
  aux f (lExp f) _ _
```

```
  where
```

```
  aux : Pol -> PP -> _ -> _ -> Pol
```

```
  aux f e _ _ =
```

```
    case reduce f gs -- search one step reduction
```

```
    of
```

```
    { nothing          -> f -- normal form f
```

```
    ; (just (pol [])) -> pol [] -- zero normal form
```

```
    ; (just f')       -> let e' = lExp f'
```

```
                        e'<e = <proof>
```

```
    in
```

```
    aux f' e' e'<e <e-descending-halt }
```

— the code is contrived.

Then the type checker would report an error of not seeing a termination proof.

But if we prove `<e-wellFounded : WellFounded _<e_`
and set this proof reference `<e-wellFounded`
to the above NF program instead of `<e-descending-halt`,
then the type checker will confirm termination.
Let us call the obtained program `Program-I`.

Way-outs. Other works. Attempts

The nicest way to implement NF is given above as Program-I.

But to constructively prove `WellFounded <e_`

is even more difficult than to prove Dickson's lemma and

`<e-descending-halt`.

There are visible the four main relevant works

(all in the segment of 1998 - 2003).

[8] (by Théry) describes a machine-checked implementation in Coq of Buchberger's algorithm. But termination is taken only as a *hypothesis*.

[3] (by Coquand and Persson) is about provable Gröbner bases in type theory (in particular in the Coq system). It also describes a certain constructive proof of Dickson's lemma.

But I failed to understand this proof!

[9] (by Coquand and Vytiniotis) pretends to constructively derive well-foundedness of an order relation from the descent-halt property (well-quasi-order) joint with a certain additional small condition. It gives a hope that this will allow to prove constructively well-foundedness of $<_e$ from Dickson's lemma and from the axioms for the $<_e$ relation.

For this, we still need first to constructively prove Dickson's lemma.

[7] (by S.A. Romanenko) describes an Agda program for a proof for Higman's lemma (for two letters). A proof for Dickson's lemma can be derived from this by coding each natural number in unary system using one letter and using another letter as a delimiter. So that Higman's word inclusion relation corresponds to inequality \leq_p on tuples in \mathbb{N}^n .

This is the easiest way for us to prove Dickson's lemma in Agda.

Finally, [5] (Martin–Mateos, Alonso, Hidalgo, Ruiz–Reina) describes a formal proof of Dickson's lemma in ACL2.

There is one page there that shows a nice geometric-like description of an idea of a constructive proof of Dickson's lemma. Its construction of the pattern multiset, and a well-founded ordering on pattern multisets do provide a proof for Dickson's lemma. But we also find that it can be used as a direct proof for termination for the normal form algorithm NF.

Therefore we are currently trying to implement in Agda this method with the multiset of patterns and to apply it to composing a termination proof for NF.

Currently we are close to accomplish this idea implementation in Agda.

Also we suggest a direct constructive proof for well-foundedness of $<_e$ for $n = 2$ (two variables).

It is implemented in Agda.

But strangely, I failed, so far, to generalize it to arbitrary n .

Also we describe below the proof for Dickson's lemma following the idea by [5].

Well-foundedness and Accessibility

Definition.

A binary relation $<$ on X is called well-founded if for any predicate P on X satisfying the properties

- ▶ $P(x)$ holds for all x minimal by $<$ AND
- ▶ $\forall x$, if $P(y)$ holds for all $y < x$, then $P(x)$

it holds $P(x)$ for all x .

This is the *transfinite induction* rule.

Example: the lexicographic ordering on $\mathbb{N} \times \mathbb{N}$ is well-founded.

As in the above definition the property P is arbitrary, let us formulate well-foundedness in a bit simpler way by replacing P with the *accessibility* predicate Acc .

Definition.

A binary relation $<$ on X is called well-founded if

$\text{Acc}(x)$ holds for all x in X ,

where the predicate Acc is defined by the following properties.

- ▶ (1) Each minimal element x is accessible: $\text{Acc}(x)$.
- ▶ (2) $\forall x$, if $\text{Acc}(y)$ for all $y < x$, then $\text{Acc}(x)$.

In our constructive proof system, the only way to prove well-foundedness of $<$ is to do a finite number of steps (1) or (2).

Examples for the well-founded property

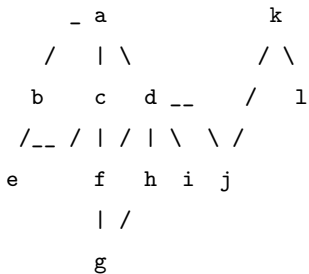
1. $<$ on \mathbb{N}
2. $<$ on \mathbb{Z} is neither descending-halt nor well-founded:

... -2 -1 0 1 2 ...

3. Prove constructively that lexicographic ordering on $\mathbb{N} \times \mathbb{N}$ is well-founded:

```
Y ~ ~ ~
  | | |
  p | |
2 | r |
1 | | |
0 |__q__|_____
  0 1 2 ...   > X
```

4. Prove constructively that any partial order $<$ on any finite set is well-founded (?)

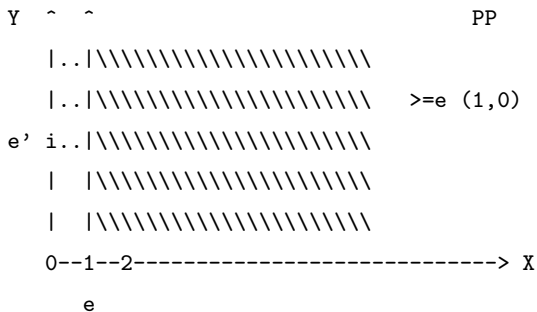


Induction by depth of a node starting from the lower node ...

Admissible Ordering for $n = 2$ is Well-founded

It is needed to prove that all integer points in the quadrant PP drawn below are accessible.

To develop an intuition, prove first that $(0,0)$ and $(1,0)$ are accessible.



Acc $(0,0)$ is because $(0,0)$ is minimal.

Prove Acc $(1,0)$: ... \square

Lemma I. The axis OX is accessible.

Prove $\text{Acc } a$ for $a = (i, 0)$ inductively on i .

The case of $i = 0$ is already proved.

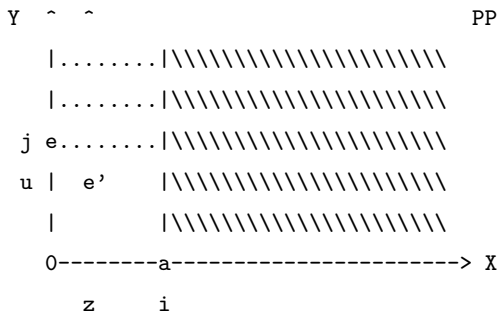
Let $i > 0$.

It is sufficient to prove $\text{Acc } e$ for all points

$$e = (x, j), \quad e <_e a, \quad j < i.$$

Prove it by induction on x .

Lemma 1, the case $x = 0$.



It suffices to prove $\text{Acc } e'$ for all $e' <_e e = (0, j)$.

All of them satisfy $e' <_e (0, j)$, a .

Hence all of such points $e' = (z, u)$ reside in the rectangle

$$R = [0, i) \times [0, j)$$

R is a finite set.

Sieve algorithmically all points of R filtering-in all the points

e' , $e' <_e e$,

and order this list increasingly by $<_e$.

This is possible because $<_e$ is total and decidable.

This produces the list

$$L = [0 = e_1 < \dots < e_m]$$

It is sufficient now to prove that all the points in L are accessible.

.....

Applying induction, all the points in L are accessible.

So, all the points smaller than e belong to L , and all the points in L are accessible. Therefore $e = (0, j)$ is accessible.

The case $x = 0$ is proved.

Lemma I, the case $0 < x < i$

It is similar to the case of $x = 0$, only induction is added, and a slight complication done.

We skip this part.

Theorem.

Any admissible ordering $<_e$ is well-founded on $\mathbb{N} \times \mathbb{N}$.

The proof is by induction on the number j of the horizontal level $y = j$ on the plane quadrant.

It proves accessibility of all points in such a level.

We skip this proof.

A machine-checked proof of this theorem is implemented in Agda.

About Dickson's Lemma

Denote

$$\text{PP} = \mathbb{N}^n$$

an additive monoid of *tuples* (which in our case are also called exponents of monomials, or power-products).

Dickson's lemma.

For any (infinite) sequence $f : \mathbb{N} \rightarrow \text{PP}$ of tuples there exist $i < j$ such that $f(i) \leq_p f(j)$ (divides).

Corollary for any admissible ordering $<_e$.

It can be proved constructively that any admissible ordering $<_e$ is descending-halt.

Because $f(i) \leq_p f(j)$ implies $f(i) \leq_e f(j)$.

The below constructive proof for Dickson's lemma follows the idea taken from [5].

We present this idea in our words.

Call the list $[e_0 \dots e_m]$ of tuples in PP *good*, if there are $i < j \leq m$ such that $e_i \leq_p e_j$.

Let $f(i) = e_i$ be a sequence in PP .

It is needed to prove that there exists a good initial segment in f .

Let $S(m) = [e_0, \dots, e_m]$ be bad.

Then none of these tuples divides any following tuple in this list.

Denote $Sh(m)$ the subset of all $e \in PP$ divisible by any member in $S(m)$.

Regions, patterns, multisets

Call $\text{Sh}(m)$ a *shaded* region of $S(m)$.

Call $W(m) = \text{PP} \setminus \text{Sh}(m)$ a *white* region.

If e_{m+1} is divisible by any element in $S(m)$

(is in the shaded region), then good $S(m+1)$ is found,
the goal is reached.

Otherwise, $S(m+1)$ is bad, $\text{Sh}(m)$ is a proper subset in
 $\text{Sh}(m+1)$,

and $W(m+1)$ is a proper subset in $W(m)$.

That is preserving badness in S is possible only while the white
region sequence is strictly decreasing by the set inclusion:

$$\dots \subset S(m) \subset \dots \subset S(1) \subset S(0)$$

Generally, to prove that $W(i)$ terminates,, represent each $W = W(i)$ symbolically as a certain multiset PM of *patterns*, so that each point in W is “covered” by some pattern in PM. Also it is defined a well-founded ordering $<_{pm}$ on the pattern multisets such that for any bad segment $S(m+1)$ the pattern multiset for $S(m+1)$ is smaller in $<_{pm}$ than the multiset for $S(m)$.

The algorithm producing bad segments $S(i)$ is joined with producing the pattern multisets $PM(i)$ representing $W(i)$. $PM(i)$ decreases in $<_{pm}$, and as this ordering is well-founded, this algorithm terminates, and the last $PM(m)$ is minimal, so that it represents a finite list L in PP, and the further termination proof is easy.

Let $l = \text{length } L$. It follows then that e_{m+l+1} is divisible by

some previous e_i — which is the goal.

Now define *pattern*, ordering on the pattern multisets, reduction of a pattern multiset by the current tuple in PP.

Denote

$$\mathbb{N}^* = \mathbb{N} \cup \ast$$

A *pattern* is a vector in $(\mathbb{N}^*)^n$.

Occurrences of \ast in a pattern are called *freedoms*.

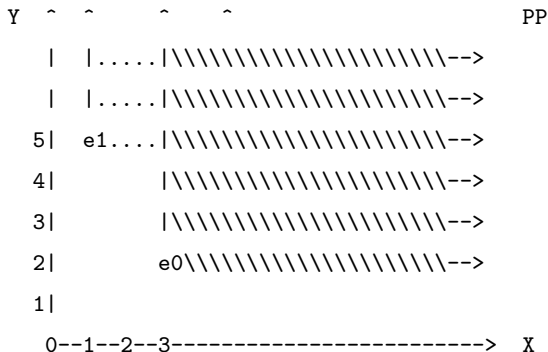
The number of freedoms in a pattern is called *dimension* of a pattern.

A pattern *represents* the set of tuples obtained by replacing every freedom with all natural natural numbers.

A white region is represented by a *multiset of patterns*.

For example, for $n = 2$, the pattern $[*,*]$ represents the whole PP .

Example. Represent $W[e_0, e_1]$ as a pattern multiset:



Its representation can be this pattern multiset (PM):

$$\{ (1, [0*, *0, *1]), (0, [10, 11, 12, 13, 14, 20, 21, 22, 33, 24]) \}$$

Here ... levels ...

A level is a pair (d, ps) ...

In PM, patterns are gathered into levels and these levels are listed decreasingly by dimension.

Multiplicity of a level is called the *length* of its pattern list.

Ordering $<_{pm}$ on PM :

first — by head dimension, then — by head multiplicity,
then — compare tails recursively.

The multiset ordering is well-founded.

It is easy to prove for our case, because a level dimension is not greater than n , so, $<_{pm}$ is actually a lexicographic ordering on \mathbb{N}^{n+1} .

Pattern reduction, multiset reduction

Obtaining the pattern set representation for the white regions $W(i)$ for any given tuple sequence e_i and its related bad segments $S(i)$ is as follows.

$W(0) = PP$. It is represented by a single pattern $\{**...*\}$.

Let $S(m) = [e_0, \dots, e_m]$ be the current bad segment,

$W(m)$ its white area,

$PM = PM(m)$ the pattern multiset representing $W(m)$,

$S(m+1) = [e_0, \dots, e_m, e_{m+1}]$ the next bad segment.

So, none in $S(m)$ divides $e = e_{m+1}$.

$W(m+1)$ is obtained by finding all the patterns in PM reducible by e , reducing these patterns by e , and inserting the redexes into PM according to the ordering, with producing the pattern multiset PM' , this PM' to represent $W(m+1)$.

An inequality \leq_* between \mathbb{N} and \mathbb{N}^* is defined so that

$i \leq_* *$ for any natural i ,

and $i \leq_* j$ means $i \leq j$ for a natural j .

An inequality \leq_{ep} between tuples PP and patterns Pattern is defined as \leq_* applied coordinate-wise.

This is a partial order.

$p \in \text{Pattern}$ is called *reducible* by $e \in \text{PP}$

if p has a freedom in it and $e \leq_{ep} p$.

For a pattern p reducible by e the list of redexes is produced as follows.

If p has $*$ at some position and e has j at this position, then the redexes for this free position are all the patterns e'_0, \dots, e'_{j-1} , where e'_k is obtained from e by replacing $*$ at this position with $0 \leq k < j$.

The exception is for the case of $j = 0$ for which this $*$ is replaced only with 0 (a deviation from [5]).

These redexes are generated for all free positions in p , the pattern p is removed from PM , and its redexes are inserted to PM to form PM' .

Note that ground patterns (the ones without freedoms) are not reducible.

Any reducible pattern has dimension $d > 0$, and all its redexes ps have dimension $d - 1$.

These redexes form the level $lev = (d - 1, ps)$, and lev is inserted into PM as follows.

If PM has not a level of dimension $d - 1$ then lev is added to PM according to the dimension ordering.

If PM has a level $(d - 1, ps')$ then this level is replaced with $(d - 1, ps + + ps')$ — the two multiplicities are summed.

Example of reducing white regions and pattern multiset.

It is taken from [5], but the result is brought to our representation.

`level0 : DimLevel`

`level0 = (2 , [generalPattern])`

`e0 = [3, 2]; e1 = [1, 5]; e2 = [2, 1]`

`levels1 = reduceLevel e0 level0`

`levels2 = reduceLevels e1 levels1`

`levels3 = reduceLevels e2 levels2`

reduce 21 -->

```
levels3 = { (1, [*0, 0*]),  
            (0, [11, 01, 24, 23, 22, 21, 20,  
                ~~~~~~  
                14, 13, 12, 11, 10]) }
```

By repeating `reduceLevels`, the current bad segment

$S = [e_0, \dots, e_m]$ is converted to the pattern multiset representing the white region for $S(m)$:

`expsToWhiteRegionLevels` : List PP \rightarrow DimLevels

`expsToWhiteRegionLevels es = foldr reduceLevels [initialLevel] es`

Theorems about Reduction, Insertion, Ordering

To complete this constructive proof of Dickson's lemma, it is needed to formally prove several lemmas about this method. In particular, reduction of the current pattern multiset must agree with reduction of the corresponding white region. For example, it is proved a

Theorem: If a tuple e is not divisible by any tuple in the list es , then e is covered by some of the patterns in the pattern multiset produced by `(expsToWhiteRegionLevels es)`.

This is formulated in Agda as

```
es-divides-e $\Rightarrow$ e $\in$ -regionLevels :
```

```
 $\forall e es \rightarrow \text{All } (\_ \not\prec e) es \rightarrow$ 
```

```
e  $\in$  ls (expsToWhiteRegionLevels es)
```

Dickson's Lemma Usage

It occurs now that a constructive proof for termination of the normal form function for polynomials can be implemented a bit more directly than by using Dickson's lemma.

It is intended to be as follows (the code is contrived):

```
<pm-wellFounded : WellFounded <pm
                -- for pattern multiset ordering
<pm-wellFounded = <... proof>
```

```
NF : Pol -> (List Pol) -> Pol
```

```
NF f gs =
```

```
  aux f (pMultiset (lExp f)) <pm-wellFounded _
```

```
                -- initiate pattern multiset
```

```
where
```

```
aux : Pol -> PMultiset -> _ -> _ ->Pol
```

```
aux f pm _ _ =
```



```

case (dim pm , reduce f gs)
of
    -- check for zero dimension
    -- and search one step reduction
{ (0 , _          ) -> finish f (listRegion pm)
; (_ , nothing    ) -> f
; (_ , just (pol []) -> pol []          -- zero normal form
; (_ , just f'    ) -> let e' = lExp f'
                        pm' = reducePMultiset e' pm

                        pm'<pm : pm' <pm pm
                        pm'<pm = <... proof>
in
    aux f' pm' <pm-wellFounded pm'<pm
}

```

This multiset reduction stops when the dimension of the head level becomes zero.

And this program must be so that the code devoted to processing pm and its proofs would not effect the run-time performance any essentially.


This is possible due to the “lazy” evaluation in Agda by default and due to other reasons.

Currently we are close to accomplish implementation in Agda for the whole method described above.


Intention:

it is desirable to prove constructively well-foundedness of any admissible ordering.

This will simplify provable programs for many methods for polynomials.

 Agda. A proof assistant. A dependently typed functional programming language and its system.

<http://wiki.portal.chalmers.se/agda/pmwiki.php>.

 B. Buchberger. “Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory”.

CAMP. Publ. No.83–29.0 November 1983.

 Th. Coquand and H. Persson. “Gröbner Bases in Type Theory”.

??

 Per Martin-Loef. “Intuitionistic type theory”.

Bibliopolis, ISBN 88-7088-105-9 (1984), 91 pages.

 F.J. Martin–Mateos, J.A. Alonso, M.J. Hidalgo, and J.L. Ruiz–Reina.

“A Formal Proof of Dickson’s Lemma in ACL2”.

M.Y. Vardi and A. Voronkov (Eds.): LPAR 2003, LNAI 2850, pp. 49–58, 2003.



U. Norell. “Dependently Typed Programming in Agda”.
AFP 2008: Advanced Functional Programming, Lecture Notes
in Computer Science, vol.5832, Springer, Berlin–Heidelberg,
2008, pages 230–266.



S.A. Romanenko. “Proof of Higman’s Lemma (for two letters)
Formalized in Agda”. In Russian. July 2017.

[https://pat.keldysh.ru/~roman/doc/talks/2017_
Romanenko__Higman's_lemma_for_2_letters_in_Agda_
ru__slides.pdf](https://pat.keldysh.ru/~roman/doc/talks/2017_Romanenko__Higman's_lemma_for_2_letters_in_Agda_ru__slides.pdf)

Agda program for the proof:

[https:](https://github.com/sergei-romanenko/agda-Higman-lemma)

[//github.com/sergei-romanenko/agda-Higman-lemma,](https://github.com/sergei-romanenko/agda-Higman-lemma)

in the folder Berghofer.



L. Théry.

“A Machine-Checked Implementation of Buchberger’s Algorithm”.

Journal of Automated Reasoning 26: 107–137, 2001.



D. Vytiniotis and Th. Coquand. “Stop when you are Almost-Full. Adventures in constructive termination”.

International Conference on Interactive Theorem Proving ITP 2012: Interactive Theorem Proving, pages 250–265.