# Lucas's Theorem

Chelsea Edmonds

February 23, 2021

**Abstract**

This work presents a formalisation of a generating function proof for Lucas's theorem. We first outline extensions to the existing Formal Power Series (FPS) library, including an equivalence relation for coefficients modulo $n$, an alternate binomial theorem statement, and a formalised proof of the Freshman's dream (mod $p$) lemma.

The second part of the work presents the formal proof of Lucas's Theorem. Working backwards, the formalisation first proves a well known corollary of the theorem which is easier to formalise and then applies induction to prove the original theorem statement. The proof of the corollary aims to provide a good example of a formalised generating function equivalence proof using the FPS library. The final theorem statement is intended to be integrated into the formalised proof of Hilbert's 10th Problem [1].

## Contents



**theory** *Lucas-Theorem*
**imports** *Main HOL−Computational-Algebra.Computational-Algebra*
**begin**

**notation** *fps-nth* (**infixl** $ 75)

# 1 Extensions on Formal Power Series (FPS) Library

This section presents a few extensions on the Formal Power Series (FPS) library, described in [2]

## 1.1 FPS Equivalence Relation

This proof requires reasoning around the equivalence of coefficients mod some prime number. This section defines an equivalence relation on FPS using the pattern described by Paulson in [4], as well as some basic lemmas for reasoning around how the equivalence holds after common operations are applied

**definition** *fpsmodrel p* ≡ { (*f*, *g*). ∀ *n*. (*f* $ *n*) *mod p* = (*g* $ *n*) *mod p* }

**lemma** *fpsrel-iff* [*simp*]: (*f*, *g*) ∈ *fpsmodrel p* ⟷ (∀ *n*. (*f* $ *n*) *mod p* = (*g* $ *n*) *mod p*)
  **by** (*simp add*: *fpsmodrel-def*)

**lemma** *fps-equiv*: *equiv UNIV* (*fpsmodrel p*)
**proof** (*rule equivI*)
  **show** *refl* (*fpsmodrel p*) **by** (*simp add*: *refl-on-def fpsmodrel-def*)
  **show** *sym* (*fpsmodrel p*) **by** (*simp add*: *sym-def fpsmodrel-def*)
  **show** *trans* (*fpsmodrel p*) **by** (*intro transI*) (*simp add*: *fpsmodrel-def*)
**qed**

Equivalence relation over multiplication

**lemma** *fps-mult-equiv-coeff*:
  **fixes** *f g* :: ('*a* :: {*euclidean-ring-cancel*}) *fps*
  **assumes** (*f*, *g*) ∈ *fpsmodrel p*
  **shows** (*f*∗*h*)$*n mod p* = (*g*∗*h*)$*n mod p*
**proof** −
  **have** ((*f*∗*h*) $ *n*) *mod p* =(∑ *i*=*0*..*n*. (*f*$*i mod p* ∗ *h*$(*n* − *i*) *mod p*) *mod p*) *mod p*
    **using** *mod-sum-eq mod-mult-left-eq*
    **by** (*simp add*: *fps-mult-nth mod-sum-eq mod-mult-left-eq*)
  **also have** ... = (∑ *i*=*0*..*n*. (*g*$*i mod p* ∗ *h*$(*n* − *i*) *mod p*) *mod p*) *mod p*
    **using** *assms* **by** *auto*
  **also have** ... = ((*g*∗*h*) $ *n*) *mod p*
    **by** (*simp add*: *mod-mult-left-eq mod-sum-eq fps-mult-nth*)
  **thus** *?thesis* **by** (*simp add*: *calculation*)
**qed**

**lemma** *fps-mult-equiv*:
  **fixes** *f g* :: ('*a* :: {*euclidean-ring-cancel*}) *fps*
  **assumes** (*f*, *g*) ∈ *fpsmodrel p*
  **shows** (*f*∗*h*, *g*∗*h*) ∈ *fpsmodrel p*

**using** *fpsmodrel-def fps-mult-equiv-coeff assms* **by** *blast*

Equivalence relation over power operator

**lemma** *fps-power-equiv*:
**fixes** $f\ g :: ('a :: \{euclidean\text{-}ring\text{-}cancel\})\ fps$
**fixes** $x :: nat$
**assumes** $(f, g) \in fpsmodrel\ p$
**shows** $(f \hat{} x, g \hat{} x) \in fpsmodrel\ p$
**using** *assms*
**proof** (*induct x*)
**case** *0*
**thus** *?case* **by** (*simp add*: *fpsmodrel-def*)
**next**
**case** (*Suc x*)
**then have** *hyp*: $\forall n.\ f \hat{} x\ \$\ n\ mod\ p = g \hat{} x\ \$\ n\ mod\ p$
**using** *fpsrel-iff* **by** *blast*
**thus** *?case*
**proof** −
**have** *fact*: $\forall n\ h.\ (g * h)\ \$\ n\ mod\ p = (f * h)\ \$\ n\ mod\ p$
**by** (*metis assms fps-mult-equiv-coeff*)
**have** $\forall n\ h.\ (g \hat{} x * h)\ \$\ n\ mod\ p = (f \hat{} x * h)\ \$\ n\ mod\ p$
**by** (*simp add*: *fps-mult-equiv-coeff hyp*)
**then have** $\forall n\ h.\ (h * g \hat{} x)\ \$\ n\ mod\ p = (h * f \hat{} x)\ \$\ n\ mod\ p$
**by** (*simp add*: *mult.commute*)
**thus** *?thesis*
**using** *fact* **by** *force*
**qed**
**qed**

## 1.2 Binomial Coefficients

The *fps-binomial* definition in the formal power series uses the *n gchoose k* operator. It's defined as being of type *'a fps*, however the equivalence relation requires a type *'a* that supports the modulo operator. The proof of the binomial theorem based on FPS coefficients below uses the choose operator and does not put bounds on the type of *fps-X*.

**lemma** *binomial-coeffs-induct*:
**fixes** $n\ k :: nat$
**shows** $(1 + fps\text{-}X) \hat{} n\ \$\ k = of\text{-}nat(n\ choose\ k)$
**proof** (*induct n arbitrary*: *k*)
**case** *0*
**thus** *?case*
**by** (*metis binomial-eq-0-iff binomial-n-0 fps-nth-of-nat not-gr-zero of-nat-0 of-nat-1 power-0*)
**next**
**case** *h*: (*Suc n*)
**fix** *k*
**have** *start*: $(1 + fps\text{-}X) \hat{} (n + 1) = (1 + fps\text{-}X) * (1 + fps\text{-}X) \hat{} n$ **by** *auto*

**show** *?case*
**using** *One-nat-def Suc-eq-plus1 Suc-pred add.commute binomial-Suc-Suc binomial-n-0*
*fps-mult-fps-X-plus-1-nth h.hyps neq0-conv start* **by** (*smt of-nat-add*)
**qed**

## 1.3 Freshman's Dream Lemma on FPS

The Freshman's dream lemma modulo a prime number $p$ is a well known proof that $(1 + x^p) \equiv (1 + x)^p \mod p$

First prove that $\binom{p^n}{k} \equiv 0 \mod p$ for $k \geq 1$ and $k < p^n$. The eventual proof only ended up requiring this with $n = 1$

**lemma** *pn-choose-k-modp-0*:
**fixes** *n k*::*nat*
**assumes** *prime p*
*k* ≥ *1* ∧ *k* ≤ *p*^*n* − *1*
*n* > *0*
**shows** (*p*^*n choose k*) *mod p* = *0*
**proof** −
**have** *inequality*: *k* ≤ *p*^*n* **using** *assms* (*2*) **by** *arith*
**have** *choose-take-1*: ((*p*^*n* − *1*) *choose* ( *k* − *1*))= *fact* (*p*^*n* − *1*) *div* (*fact* (*k* − *1*) ∗ *fact* (*p*^*n* − *k*))
**using** *binomial-altdef-nat diff-le-mono inequality assms*(*2*) **by** *auto*
**have** *k* ∗ (*p*^*n choose k*) = *k* ∗ ((*fact* (*p*^*n*)) *div* (*fact k* ∗ *fact*((*p*^*n*) − *k*)))
**using** *assms binomial-fact'*[*OF inequality*] **by** *auto*
**also have** ... = *k* ∗ *fact* (*p*^*n*) *div* (*fact k* ∗ *fact*((*p*^*n*) − *k*))
**using** *binomial-fact-lemma div-mult-self-is-m fact-gt-zero inequality mult.assoc mult.commute*
*nat-0-less-mult-iff* **by** *smt*
**also have** ... = *k* ∗ *fact* (*p*^*n*) *div* (*k* ∗ *fact* (*k* − *1*) ∗ *fact*((*p*^*n*) − *k*))
**by** (*metis assms*(*2*) *fact-nonzero fact-num-eq-if le0 le-antisym of-nat-id*)
**also have** ... = *fact* (*p*^*n*) *div* (*fact* (*k* − *1*) ∗ *fact*((*p*^*n*) − *k*))
**using** *assms* **by** *auto*
**also have** ... = ((*p*^*n*) ∗ *fact* (*p*^*n* − *1*)) *div* (*fact* (*k* − *1*) ∗ *fact*((*p*^*n*) − *k*))
**by** (*metis assms*(*2*) *fact-nonzero fact-num-eq-if inequality le0 le-antisym of-nat-id*)
**also have** ... = (*p*^*n*) ∗ (*fact* (*p*^*n* − *1*) *div* (*fact* (*k* − *1*) ∗ *fact*((*p*^*n*) − *k*)))
**by** (*metis assms*(*2*) *calculation choose-take-1 neq0-conv not-one-le-zero times-binomial-minus1-eq*)
**finally have** *equality*: *k* ∗ (*p*^*n choose k*) = *p*^*n* ∗ ((*p*^*n* − *1*) *choose* (*k* − *1*))
**using** *assms*(*2*) *times-binomial-minus1-eq* **by** *auto*
**then have** *dvd-result*: *p*^*n dvd* (*k* ∗ (*p*^*n choose k*)) **by** *simp*
**have** ¬ (*p*^*n dvd k*)
**using** *assms* (*2*) *binomial-n-0 diff-diff-cancel nat-dvd-not-less neq0-conv* **by** *auto*

**then have** *p dvd* (*p*^*n choose k*)
**using** *mult.commute prime-imp-prime-elem prime-power-dvd-multD assms dvd-result* **by** *metis*
**thus** *?thesis* **by** *simp*
**qed**

Applying the above lemma to the coefficients of $(1+X)^p$, it is easy to show that all coefficients other than the 0th and $p$th will be 0

**lemma** *fps-middle-coeffs*:
**assumes** *prime p*
$n \neq 0 \wedge n \neq p$
**shows** $((1 + fps\text{-}X :: int\ fps)\ \hat{}\ p)\ \$\ n\ mod\ p = 0\ mod\ p$
**proof** −
**let** *?f* = $(1 + fps\text{-}X :: int\ fps)\ \hat{}\ p$
**have** $\forall\ n.\ n > 0 \wedge n < p \longrightarrow (p\ choose\ n)\ mod\ p = 0$ **using** *pn-choose-k-modp-0*
**by** (*metis* (*no-types*, *lifting*) *add-le-imp-le-diff assms*(*1*) *diff-diff-cancel diff-is-0-eq′*

*discrete le-add-diff-inverse le-numeral-extra*(*4*) *power-one-right zero-le-one zero-less-one*)
**then have** *middle-0*: $\forall\ n.\ n > 0 \wedge n < p \longrightarrow (?f\ \$\ n)\ mod\ p = 0$
**using** *binomial-coeffs-induct* **by** (*metis of-nat-0 zmod-int*)
**have** $\forall\ n.\ n > p \longrightarrow ?f\ \$\ n\ mod\ p = 0$
**using** *binomial-eq-0-iff binomial-coeffs-induct mod-0* **by** (*metis of-nat-eq-0-iff*)
**thus** *?thesis* **using** *middle-0 assms*(*2*) *nat-neq-iff* **by** *auto*
**qed**

It follows that $(1 + X)^p$ is equivalent to $(1 + X^p)$ under our equivalence relation, as required to prove the freshmans dream lemma.

**lemma** *fps-freshmans-dream*:
**assumes** *prime p*
**shows** $(((1 + fps\text{-}X :: int\ fps\ )\ \hat{}\ p), (1 + (fps\text{-}X)\ \hat{}\ (p))) \in fpsmodrel\ p$
**proof** −
**let** *?f* = $(1 + fps\text{-}X :: int\ fps)\ \hat{}\ p$
**let** *?g* = $(1 + (fps\text{-}X :: int\ fps)\ \hat{}\ p)$
**have** *all-f-coeffs*: $\forall\ n.\ n \neq 0 \wedge n \neq p \longrightarrow ?f\ \$\ n\ mod\ p = 0\ mod\ p$
**using** *fps-middle-coeffs assms* **by** *blast*
**have** *?g* $\$\ 0 = 1$ **using** *assms* **by** *auto*
**then have** *?g* $\$\ 0\ mod\ p = 1\ mod\ p$
**using** *int-ops*(*2*) *zmod-int assms* **by** *presburger*
**then have** *?g* $\$\ p\ mod\ p = 1\ mod\ p$ **using** *assms* **by** *auto*
**then have** $\forall\ n\ .\ ?f\ \$\ n\ mod\ p = ?g\ \$\ n\ mod\ p$
**using** *all-f-coeffs* **by** (*simp add*: *binomial-coeffs-induct*)
**thus** *?thesis* **using** *fpsrel-iff* **by** *blast*
**qed**

# 2 Lucas's Theorem Proof

A formalisation of Lucas's theorem based on a generating function proof using the existing formal power series (FPS) Isabelle library

## 2.1 Reasoning about Coefficients Helpers

A generating function proof of Lucas's theorem relies on direct comparison between coefficients of FPS which requires a number of helper lemmas to prove formally. In particular it compares the coefficients of $(1+X)^n \mod p$ to $(1+X^p)^N * (1+X)^r n \mod p$, where $N = n/p$, and $rn = n \mod p$. This section proves that the $k$th coefficient of $(1+X^p)^N * (1+X)^r n = (NchooseK) * (rnchooserk)$

Applying the $(oo)$ operator enables reasoning about the coefficients of $(1+X^p)^n$ using the existing binomial theorem proof with $X^p$ instead of $X$.

**lemma** *fps-binomial-p-compose*:
**assumes** $p \neq 0$
**shows** $(1 + (fps\text{-}X :: ('a :: \{idom\}\ fps))\hat{}p)\hat{}n = ((1 + fps\text{-}X)\hat{}n)\ oo\ (fps\text{-}X\hat{}p)$
**proof** −
**have** $(1::'a\ fps) + fps\text{-}X\ \hat{}\ p = 1 + fps\text{-}X\ oo\ fps\text{-}X\ \hat{}\ p$
**by** (*simp add*: *assms fps-compose-add-distrib*)
**then show** *?thesis*
**by** (*simp add*: *assms fps-compose-power*)
**qed**

Next the proof determines the value of the $k$th coefficient of $(1+X^p)^N$.

**lemma** *fps-X-pow-binomial-coeffs*:
**assumes** *prime p*
**shows** $(1 + (fps\text{-}X ::int\ fps)\hat{}p)\hat{}N\ \$k = (if\ p\ dvd\ k\ then\ (N\ choose\ (k\ div\ p))\ else\ 0)$
**proof** −
**let** *?fx* = (*fps-X* :: *int fps*)
**have** $(1 + ?fx\hat{}p)\hat{}N\ \$\ k\ = (((1 + ?fx)\hat{}N)\ oo\ (?fx\hat{}p))\ \$k$
**by** (*metis assms fps-binomial-p-compose not-prime-0*)
**also have** ... = $(\sum i{=}0..k.((1 + ?fx)\hat{}N)\$i * ((?fx\hat{}p)\hat{}i\$k))$
**by** (*simp add*: *fps-compose-nth*)
**finally have** *coeffs*: $(1 + ?fx\hat{}p)\hat{}N\ \$\ k = (\sum i{=}0..k.\ (N\ choose\ i) * ((?fx\hat{}(p{*}i))\$k))$
**using** *binomial-coeffs-induct sum.cong* **by** (*metis* (*no-types*, *lifting*) *power-mult*)

**thus** *?thesis*
**proof** (*cases p dvd k*)
**case** *False* — $p$ does not divide $k$ implies the $k$th term has a coefficient of 0
**have** $\forall\ i.\ \neg(p\ dvd\ k) \longrightarrow (?fx\hat{}(p{*}i))\ \$\ k = 0$
**by** *auto*
**thus** *?thesis* **using** *coeffs* **by** (*simp add*: *False*)
**next**
**case** *True* — $p$ divides $k$ implies the $k$th term has a non-zero coefficient
**have** *contained*: $k\ div\ p \in \{0..\ k\}$ **by** *simp*
**have** $\forall\ i.\ i \neq k\ div\ p \longrightarrow (?fx\hat{}(p{*}i))\ \$\ k = 0$ **using** *assms* **by** *auto*
**then have** *notdivpis0*: $\forall\ i \in (\{0\ ..\ k\} - \{k\ div\ p\}).\ (?fx\hat{}(p{*}i))\ \$\ k = 0$ **by** *simp*
**have** $(1 + ?fx\hat{}p)\hat{}N\ \$\ k = (N\ choose\ (k\ div\ p)) * (?fx\hat{}(p * (k\ div\ p)))\ \$\ k + (\sum i{\in}(\{0..k\} - \{k\ div\ p\}).\ (N\ choose\ i) * ((?fx\hat{}(p{*}i))\$k))$

**using** *contained coeffs sum.remove* **by** (*metis* (*no-types*, *lifting*) *finite-atLeastAtMost*)
**thus** *?thesis* **using** *notdivpis0 True* **by** *simp*
**qed**
**qed**

The final helper lemma proves the $k$th coefficient is equivalent to $\binom{?N}{?K} * \binom{?rn}{?rk}$ as required.

**lemma** *fps-div-rep-coeffs*:
**assumes** *prime p*
**shows** $((1 + (fps\text{-}X::int\ fps)\hat{}p)\hat{}(n\ div\ p) * (1 + fps\text{-}X)\hat{}(n\ mod\ p))\ \$\ k =$
$((n\ div\ p)\ choose\ (k\ div\ p)) * ((n\ mod\ p)\ choose\ (k\ mod\ p))$
(**is** $((1 + (fps\text{-}X::int\ fps)\hat{}p)\hat{}?N * (1 + fps\text{-}X)\hat{}?rn)\ \$\ k = (?N\ choose\ ?K) * (?rn\ choose\ ?rk)$)
**proof** −
— Initial facts with results around representation and 0 valued terms
**let** *?fx* = *fps-X* :: *int fps*
**have** *krep*: $k - ?rk = ?K*p$
**by** (*simp add*: *minus-mod-eq-mult-div*)
**have** *rk-in-range*: $?rk \in \{0..k\}$ **by** *simp*
**have** $\forall\ i \geq p.\ (?rn\ choose\ i) = 0$
**using** *binomial-eq-0-iff*
**by** (*metis assms*(*1*) *leD le-less-trans linorder-cases mod-le-divisor mod-less-divisor prime-gt-0-nat*)
**then have** *ptok0*: $\forall\ i \in \{p..k\}.\ ((?rn\ choose\ i) * (1 + ?fx\hat{}p)\hat{}?N\ \$\ (k - i)) = 0$
**by** *simp*
**then have** *notrkis0*: $\forall i \in \{0..\ k\}.\ i \neq ?rk \longrightarrow (?rn\ choose\ i) * (1 + ?fx\hat{}p)\hat{}?N\ \$\ (k - i) = 0$
**proof** (*cases* $k < p$)
**case** *True* — When $k < p$, it presents a side case with regards to range of reasoning
**then have** *k-value*: $k = ?rk$ **by** *simp*
**then have** $\forall\ i < k.\ \neg\ (p\ dvd\ (k - i))$
**using** *True* **by** (*metis diff-diff-cancel diff-is-0-eq dvd-imp-mod-0 less-imp-diff-less less-irrefl-nat mod-less*)
**then show** *?thesis* **using** *fps-X-pow-binomial-coeffs assms*(*1*) *k-value* **by** *simp*
**next**
**case** *False*
**then have** $\forall\ i < p.\ i \neq ?rk \longrightarrow \neg(p\ dvd\ (k - i))$
**using** *mod-nat-eqI* **by** *auto*
**then have** $\forall\ i \in \{0..<p\}.\ i \neq ?rk \longrightarrow (1 + ?fx\hat{}p)\hat{}?N\ \$\ (k - i) = 0$
**using** *assms fps-X-pow-binomial-coeffs* **by** *simp*
**then show** *?thesis* **using** *ptok0* **by** *auto*
**qed**
— Main body of the proof, using helper facts above
**have** $((1 + fps\text{-}X\hat{}p)\hat{}?N * (1 + fps\text{-}X)\hat{}?rn)\ \$\ k = (((1 + fps\text{-}X)\hat{}?rn) * (1 + fps\text{-}X\hat{}p)\hat{}?N)\ \$\ k$
**by** (*metis* (*no-types*, *hide-lams*) *distrib-left distrib-right fps-mult-fps-X-commute fps-one-mult*(*1*)

*fps-one-mult*(*2*) *power-commuting-commutes*)
**also have** ... = ($\sum$ *i*=*0*..*k*.(*of-nat*(*?rn choose i*)) * ((*1* + (*fps-X*)^*p*)^*?N* $ (*k* − *i*)))
**by** (*simp add*: *fps-mult-nth binomial-coeffs-induct*)
**also have** ... = ((*?rn choose ?rk*) * (*1* + *?fx*^*p*)^*?N* $ (*k* − *?rk*)) + ($\sum$ *i*∈({*0*..*k*} − {*?rk*}). (*?rn choose i*) * (*1* + *?fx*^*p*)^*?N* $ (*k* − *i*))
**using** *rk-in-range sum.remove* **by** (*metis* (*no-types*, *lifting*) *finite-atLeastAtMost*)
**finally have** ((*1* + *?fx*^*p*)^*?N* * (*1* + *?fx*)^*?rn*) $ *k* = ((*?rn choose ?rk*) * (*1* + *?fx*^*p*)^*?N* $ (*k* − *?rk*))
**using** *notrkis0* **by** *simp*
**thus** *?thesis* **using** *fps-X-pow-binomial-coeffs assms krep* **by** *auto*
**qed**

## 2.2 Lucas Theorem Proof

The proof of Lucas's theorem combines a generating function approach, based off [3] with induction. For formalisation purposes, it was easier to first prove a well known corollary of the main theorem (also often presented as an alternative statement for Lucas's theorem), which can itself be used to backwards prove the the original statement by induction. This approach was adapted from P. Cameron's lecture notes on combinatorics [5]

### 2.2.1 Proof of the Corollary

This step makes use of the coefficient equivalence arguments proved in the previous sections

**corollary** *lucas-corollary*:
**fixes** *n k* :: *nat*
**assumes** *prime p*
**shows** (*n choose k*) *mod p* = (((*n div p*) *choose* (*k div p*)) * ((*n mod p*) *choose* (*k mod p*))) *mod p*
(**is** (*n choose k*) *mod p* = ((*?N choose ?K*) * (*?rn choose ?rk*)) *mod p*)
**proof** −
**let** *?fx* = *fps-X* :: *int fps*
**have** *n-rep*: *n* = *?N* * *p* + *?rn*
**by** *simp*
**have** *k-rep*: *k* = *?K* * *p* + *?rk* **by** *simp*
**have** *rhs-coeffs*: ((*1* + *?fx*^*p*)^(*?N*) * (*1* + *?fx*)^(*?rn*)) $ *k* = (*?N choose ?K*) * (*?rn choose ?rk*)
**using** *assms fps-div-rep-coeffs k-rep n-rep* **by** *blast* — Application of coefficient reasoning
**have** ((((*1* + *?fx*)^*p*)^(*?N*) * (*1* + *?fx*)^(*?rn*)),
((*1* + *?fx*^*p*)^(*?N*) * (*1* + *?fx*)^(*?rn*))) ∈ *fpsmodrel p*
**using** *fps-freshmans-dream assms fps-mult-equiv fps-power-equiv* **by** *blast* — Application of equivalence facts and freshmans dream lemma
**then have** *modrel2*: ((*1* + *?fx*)^*n*, ((*1* + *?fx*^*p*)^(*?N*) * (*1* + *?fx*)^(*?rn*)))
∈ *fpsmodrel p*

**by** (*metis* (*mono-tags*, *hide-lams*) *mult-div-mod-eq power-add power-mult*)
**thus** *?thesis*
**using** *fpsrel-iff binomial-coeffs-induct rhs-coeffs* **by** (*metis of-nat-eq-iff zmod-int*)

**qed**

### 2.2.2 Proof of the Theorem

The theorem statement requires a formalised way of referring to the base $p$ representation of a number. We use a definition that specifies the $i$th digit of the base $p$ representation. This definition is originally from the Hilbert's 10th Problem Formalisation project [1] which this work contributes to.

**definition** *nth-digit-general* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* **where**
*nth-digit-general num i base* = (*num div* (*base* ^ *i*)) *mod base*

Applying induction on $d$, where $d$ is the highest power required in either $n$ or $k$'s base $p$ representation, *prime ?p* ⟹ (*?n choose ?k*) *mod ?p* = (*?n div ?p choose ?k div ?p*) ∗ (*?n mod ?p choose ?k mod ?p*) *mod ?p* can be used to prove the original theorem.

**theorem** *lucas-theorem*:
**fixes** *n k d*::*nat*
**assumes** *n* < *p* ^ (*Suc d*)
**assumes** *k* < *p* ^ (*Suc d*)
**assumes** *prime p*
**shows** (*n choose k*) *mod p* = (∏ *i*≤*d*. ((*nth-digit-general n i p*) *choose* (*nth-digit-general k i p*))) *mod p*
**using** *assms*
**proof** (*induct d arbitrary*: *n k*)
**case** *0*
**thus** *?case* **using** *nth-digit-general-def assms* **by** *simp*
**next**
**case** (*Suc d*)
— Representation Variables
**let** *?N* = *n div p*
**let** *?K* = *k div p*
**let** *?nr* = *n mod p*
**let** *?kr* = *k mod p*
— Required assumption facts
**have** *Mlessthan*: *?N* < *p* ^ (*Suc d*)
**using** *less-mult-imp-div-less power-Suc2 assms*(*3*) *prime-ge-2-nat Suc.prems*(*1*)
**by** *metis*
**have** *Nlessthan*: *?K* < *p* ^ (*Suc d*)
**using** *less-mult-imp-div-less power-Suc2 prime-ge-2-nat Suc.prems*(*2*) *assms*(*3*)
**by** *metis*
**have** *shift-bounds-fact*: (∏ *i*=(*Suc 0*)..(*Suc* (*d* )). ((*nth-digit-general n i p*) *choose* (*nth-digit-general k i p*))) =
(∏ *i*=*0*..(*d*). (*nth-digit-general n* (*Suc i*) *p*) *choose* (*nth-digit-general k* (*Suc i*) *p*))

**using** *prod.shift-bounds-cl-Suc-ivl* **by** *blast* — Product manipulation helper fact
**have** (*n choose k* ) *mod p* = ((*?N choose ?K*) ∗ (*?nr choose ?kr*)) *mod p*
**using** *lucas-corollary assms*(*3*) **by** *blast* — Application of corollary
**also have** ...= ((∏ *i*≤*d*. ((*nth-digit-general ?N i p*) *choose* (*nth-digit-general ?K i p*))) ∗ (*?nr choose ?kr*)) *mod p*
**using** *Mlessthan Nlessthan Suc.hyps mod-mult-cong assms*(*3*) **by** *blast* — Using Inductive Hypothesis
— Product manipulation steps
**also have** ... = ((∏ *i*=*0*..(*d*). (*nth-digit-general n* (*Suc i*) *p*) *choose* (*nth-digit-general k* (*Suc i*) *p*)) ∗ (*?nr choose ?kr*)) *mod p*
**using** *atMost-atLeast0 nth-digit-general-def div-mult2-eq* **by** *auto*
**also have** ... = ((∏ *i*=*1*..(*d*+*1*). (*nth-digit-general n i p*) *choose* (*nth-digit-general k i p*)) ∗
((*nth-digit-general n 0 p*) *choose* (*nth-digit-general k 0 p*))) *mod p*
**using** *nth-digit-general-def shift-bounds-fact* **by** *simp*
**finally have** (*n choose k* ) *mod p* = ((∏ *i*=*0*..(*d*+*1*). (*nth-digit-general n i p*) *choose* (*nth-digit-general k i p*))) *mod p*
**using** *One-nat-def atMost-atLeast0 mult.commute prod.atLeast1-atMost-eq prod.atMost-shift*
**by** (*smt Suc-eq-plus1 shift-bounds-fact*)
**thus** *?case*
**using** *Suc-eq-plus1 atMost-atLeast0* **by** *presburger*
**qed**

**end**

# References


[1] J. Bayer, M. David, A. Pal, B. Stock, and D. Schleicher. The DPRM Theorem in Isabelle (Short Paper). In J. Harrison, J. O'Leary, and A. Tolmach, editors, *10th International Conference on Interactive Theorem Proving (ITP 2019)*, volume 141 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:7, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[2] A. Chaieb. Formal power series. *Journal of Automated Reasoning*, 47(3):291–318, Oct. 2011.

[3] N. J. Fine. Binomial coefficients modulo a prime. *The American Mathematical Monthly*, 54(10):589–592, 1947.

[4] L. C. Paulson. Defining Functions on Equivalence Classes. *ACM Transactions on Computational Logic (TOCL)*, 7(4):658–675, Oct. 2006.

[5] Peter Cameron. Notes on Combinatorics. http://www.maths.qmul.ac.uk/~pjc/notes/comb.pdf, 2007.